AxCrypt Version 1 Algorithms and File Format

Introduction

This document provides information on the file format used by the AxCrypt family of software, which provides confidentiality and integrity of stored data using encryption.

General Terms

- AxCrypt - The name of the original software version 1.x as well as a general term for the whole family of software which is compatible with AxCrypt-encrypted files.
- AxCrypt.NET - The version 2.x level software, which is a complete rewrite of the original software.
- AxDecrypt - A stand-alone utility which can only decrypt, and also implements self-decryption capabilities.
- AxCrypt2Go - A stand-alone utility which can decrypt and encrypt, but does not integrate with application launching.

Technical Terms

- Passphrase - A string of characters entered by a user, which indirectly provides access to encrypted data. The passphrase is never directly used to encrypt data. A fixed-length binary Key Encrypting Key is derived from the passphrase using a key derivation algorithm.
- Key Encrypting Key - A key used to encrypt a master key, abbreviated KEK. The KEK is typically derived from a variable length passphrase.
- Master Encrypting Key - A randomly generated key, which is used as the basis for several derived keys which are used to encrypt headers, calculate an HMAC and to actually encrypt data. Abbreviated MEK. The MEK is protected by a Key Wrap,using the KEK as the key.
- Sub Key - A key derived from the MEK by encrypting a constant in ECB mode using the MEK as the encryption key.
- HMAC - Message authentication code as defined by RFC 2104.
- AxCrypt-HMAC-SHA1-128, An HMAC using SHA-1 with a block size of 160 with the result truncated to 128 bits.

General functions

The AxCrypt file format provides support for storing data and meta data in a way to ensure both confidentiality and integrity. It also provides support for compression according to RFC 1950 ZLIB and RFC 1951 DEFLATE .

AxCrypt 1.x and 2.x

The AxCrypt common parts of 1.x and 2.x file format are described here.

Overview

An AxCrypt 1.x or 2.x file consists of a unique GUID preamble, a number of headers and finally the encrypted and optionally compressed data.

The AxCrypt GUID

```
  0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 2e  07  b9  c0  93  4f  46  f1  a0  15  79  2c  a1  d9  e8  21|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

The value of the GUID is thus 0x2e07b9c0934f46f1a015792ca1d9e821 . This is used to determine if the following data does follow the AxCrypt file format standard. The same GUID is used for version 1 and version 2, although the subsequent format differs slightly.

Header and Header Blocks

Following the AxCrypt GUID, are a number of Header Blocks which together make up the Header. The header contains various meta data as well as the actual Master Key, wrapped with the Key Encrypting Key derived from the passphrase.

A header block follows the following general format:

```
  0   1   2   3   4   5 ......... Length-1
+---+---+---+---+---+====================+
|LSB  Length MSB|HBT| Length-5 bytes data |
+---+---+---+---+---+====================+
```

The Length is the total length of the data block, including the length itself. The length is stored as a 32-bit unsigned integer in little endian order, i.e. with the least significant byte (LSB) first and the most significant byte (MSB) last.

The Header Block Type (HBT) is a one-byte integer specifying what kind of data is stored in the block.

The following header block types are recognized by AxCrypt 1.x and 2.x:

2 - Preamble, Must be first.
3 - Version, Version information etc.

63 - Data, The data, compressed and/or encrypted. This header does not actually contain data in AxCrypt 2.x. See AxCrypt 2.x for details.

64 - Encrypted,Start of headers containing encrypted header data. There is no actual header with this value.

68 - FileInfo, Time stamps and size of the original file
69 - Compression, Indicates if the data is compressed. 1.2.2.
70 - UnicodeFileNameInfo, Original file name in Unicode. 1.6.3.3

All block types except those listed are undefined. Blocks with these types should be skipped and silently ignored, since they may have been defined by future versions of the software.

AxCrypt 1.x

The following header block types are recognized by AxCrypt 1.x:

2 - Preamble, Must be first.
3 - Version, Version information etc.
4 - KeyWrap1, A 128-bit Data Enc Key wrapped with 128-bit KEK
5 - KeyWrap2, Some other kind of KEK, DEK, IV scheme... Future use.
6 - IdTag, An arbitrary string encoded as Ansi Code Page 1252 defined by the caller.

63 - Data, The data, compressed and/or encrypted.

64 - Encrypted,Start of headers containing encrypted header data. There is no actual header with this value.

65 - FileNameInfo, Original file name
66 - EncryptionInfo, Sizes of the original data file before encryption
67 - CompressionInfo, Indicates that the data is compressed and the sizes.
68 - FileInfo, Time stamps and size of the original file
69 - Compression, Indicates if the data is compressed. 1.2.2.
70 - UnicodeFileNameInfo, Original file name in Unicode. 1.6.3.3

All block types except those listed are undefined. Blocks with these types should be skipped and silently ignored, since they may have been defined by future versions of the software.

Header Blocks Encryption

Header blocks with a Header Block Type >= 64 are encrypted using AES in CBC mode, with a separate sub key, and a zero IV.

The sub key is generated by taking the Master Encrypting Key, and encrypting a data block in Electronic Code Book mode where the first byte has a value of ‘2’, and the remaining 15 bytes are zero.

Header Blocks

Header Block 2 - Preamble

The first header block must always be of type 2, Preamble.

The format is:

```
  0   1   2   3   4   5 ...................... 20
+---+---+---+---+---+================================+
| 21  0   0   0 | 2 | 16 bytes AxCrypt-HMAC-SHA1-128 |
+---+---+---+---+---+================================+
```

This is the AxCrypt-version of HMAC-SHA-1, a 128-bit truncated HMAC of the following headers and all encrypted data, where the block size for the key is 160 bits, instead of the standard 512 bits. The key is a sub key generated from the Master Encrypting Key.

The sub key is generated by taking the Master Encrypting Key, and encrypting a data block in Electronic Code Book mode where the first byte has a value of ‘0’, and the remaining 15 bytes are zero.

Header Block 3 - Version, Version information etc

The version header block records information about the File Format version as well as the software generator version.

The format is:

```
  0   1   2   3   4
+---+---+---+---+---+---+---+---+---+---+
| 10  0   0   0 | 3 | a   b   x   y   z |
+---+---+---+---+---+---+---+---+---+---+
```

Where:

- a => File Major Version
- b => File Minor Version
- x => Program Major Version
- y => Program Minor Version
- z => Program Minor Version

The Program version is mostly for informative purposes. The File Minor Version can be used to control behavior for newer versions when reading older data.

The most important value is the File Major Version. A reading program that detects a higher File Major Version than it knows it can interpret, should stop reading and present a message to the effect that while the file is an AxCrypt file, the current program must be upgraded.

The most recent File Major.Minor Version at the time of writing is 3.2 and corresponds to AxCrypt 1.7.

HeaderBlock 4 - KeyWrap1, A 128-bit Data Enc Key wrapped with 128-bit KEK

This is the master encryption key, wrapped using the NIST AES Key Wrap Algorithm with a slight tweak due to a byte ordering issue in AxCrypt.

The format is:

```
  0   1   2   3   4   5 ....... 28   29 .. 44   45 ....... 48
+---+---+---+---+---+==============+==========+================+
| 49  0   0   0 | 4 | Key Wrap(24) | Salt(16) | Iterations(4) |
+---+---+---+---+---+==============+==========+================+
```

The Key Wrap consists of 3 blocks of 64-bits, which is the result of the algorithm when wrapping a 128-bit key. The Key Wrap uses the suggested IV 0xA6A6A6A6A6A6A6A6 to ensure the integrity of the wrapped key, and also to test for correctness. A best effort brute force attack would thus focus on the Key Wrap, not the data which is why the iterations are dynamically determined and calculated to take approximately 100ms on the encrypting system.

The iteration counter is stored as a 32-bit little endian unsigned integer.

The master encryption key is never used directly. Sub keys are derived by encrypting a data block with a specific first byte, the rest zero in Electronic Code Book mode. The sub keys used are respectively:

0 - HMAC
2 - Headers
3 - Data

The AxCrypt byte ordering tweak

The byte ordering issue changes the algorithm as follows. The original states:

B = AES(K) (A | Ri)
A = MSB64 (B) XOR t where t = (n * j)+ i
Ri = LSB64 (B)

The issue is that the original assumes that all values are stored big endian including ‘t’. The AxCrypt bug causes the algorithm to work as follows instead:

B = AES(K) (A | Ri)
A = MSB64 (B) XOR LittleEndian64(t) where t = (n * j)+ i
Ri = LSB64 (B)

This should not cause any change in the security of the wrap, since it just moves the change introduced by the XOR to different input bit positions in the next round of encryption.

HeaderBlock 5 - KeyWrap2. Future use. Ignored.

This is a reserved header block type which is ignored by all AxCrypt 1.x versions.

The format is:

```
  0   1   2   3   4   5 ... n-1
+---+---+---+---+---+===========+
|       n       | 5 | data(n-5) |
+---+---+---+---+---+===========+
```

HeaderBlock 6 - IdTag

This is a header block that can be used to store an arbitrary nul-terminated ANSI code page 1252 string. It is currently ignored.

The format is:

```
  0   1   2   3   4   5 ..... n-1
+---+---+---+---+---+=============+
|        n      | 6 | string(n-5) |
+---+---+---+---+---+=============+
```

Header Block 63 - Data, The data

This header block signals the start of the actual data which is encrypted, and optionally compressed as specified in preceeding headers.

This header is always the last header block.

The format is:

```
  0   1   2   3   4   5   6   7   8   9   10  11  12
+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 13  0   0   0 | 63| Total length of data(8)       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

This is the last header block. Directly following this block are the number of bytes of data specified in the total length of data field. This field is stored as a 64-bit unsigned little endian integer.

Header Block 65 - Original file name

This header block is encrypted. No particular padding is used.
The header block contains the original file name in a nul-terminated ASCII encoding.

The decrypted format is:

```
  0   1   2   3   4   5                               n-1
+---+---+---+---+---+===============================+
|       n       | 65| Encrypted File Name<nul>(?..?)|
+---+---+---+---+---+===============================+
```

The value n is the total length of the header block, a multiple of 16 + 5 bytes. The data must first be decrypted as described above, and the result is the nul-terminated ASCII encoding of the original file name. If the file name data did not fit into an even multiple of the data block size, the last block is extended with undefined data.

Header Block 66 - Encryption Information

This header block is encrypted. No particular padding is used.
The block contains the original plain size of the data, as well as the data encryption IV.

The decrypted format is:

```
  0   1   2   3   4   5   .......... 12  13 ... 28   29 .. 36
+---+---+---+---+---+==================+=========+=========+
| 37  0   0   0 | 66| PlaintextLength(8)| IV(16)   | extra(8)|
+---+---+---+---+---+==================+=========+=========+
```

The plain text length is a 64-bit unsigned integer stored in little endian format. The IV is a vector of bytes used for the data encryption.
The plain text length is the number of bytes input to the encryption stage, the data may or may not be compressed.
Extra are padding bytes required to even out the data to an even multiple of the block length.

Header Block 67 - Compression information.

This header block is encrypted. No particular padding is used.
Implies the data is compressed, and contains the uncompressed size of the data.

The decrypted format is:

```
  0   1   2   3   4   5   6   7   8   9   10  11  12  13 .. 20
+---+---+---+---+---+---+---+---+---+---+---+---+---+=========+
| 21  0   0   0 | 67| Uncompressed Length(8)        | extra(8)|
+---+---+---+---+---+---+---+---+---+---+---+---+---+=========+
```

The uncompressed length is stored as a signed 64-bit little endian integer. The resulting compressed length will be stored as the plain text length field in the Encryption Information header.
Extra are padding bytes required to even out the data to an even multiple of the block length.

Header Block 68 - File Information

This header block is encrypted. No particular padding is used.
Original time stamps are stored in this header.

The decrypted format is:

```
  0   1   2   3   4   5   12  13  20  21  28 29 ... 36
+---+---+---+---+---+=======+=======+=======+=========+
| 37  0   0   0 | 68| CRT(8)| LAT(8)|LWT(8) | extra(8)|
+---+---+---+---+---+=======+=======+=======+=========+
```

The time stamps are as follows:

- CRT - Creation Time
- LAT - Last Access Time
- LWT - Last Write Time

The time is encoded as a 64-bit field, representing the number of ticks since January 1, 1601, UTC. One single tick is one hundred nanoseconds or one ten-millionth of a second. There are 10,000 ticks in a millisecond.
Extra are padding bytes required to even out the data to an even multiple of the block length.

Header Block 69 - Compression flag

This header block is encrypted. No particular padding is used.
This header is a simple flag, indicating if the data is compressed or not. If it is compressed, the format is according to the zlib format with the deflate algorithm.

The decrypted format is:

```
  0   1   2   3   4   5                8   9      20
+---+---+---+---+---+================+==========+
| 21  0   0   0 | 69| Is Compressed(4)| extra(12)|
+---+---+---+---+---+================+==========+
```

The is compressed flag is a signed 32-bit little endian integer, where a value different than zero is a 'true' value, i.e. indicates that the data is in fact compressed.
Extra are padding bytes required to even out the data to an even multiple of the block length.

Header Block 70 - Unicode File Name

This header block is encrypted. No particular padding is used.

The decrypted format is:

```
  0   1   2   3   4   5                          n-1
+---+---+---+---+---+==============================+
|       n       | 70| Encrypted File Name<nul>(?..?)|
+---+---+---+---+---+==============================+
```

The value n is the total length of the header block, a multiple of 16 + 5 bytes. The data must first be decrypted as described above, and the result is the double-nul-terminated UTF-16 encoding of the original file name. If the file name data did not fit into an even multiple of the data block size, the last block is extended with undefined data.

The encrypted data

The encrypted data itself follows all the headers, and particularly after the data header block, which is always the last one.

The data is encrypted using CBC with a key derived from the Master Encryption Key, an IV from the encryption info header, using PKCS #7 padding.

The sub key is generated by taking the Master Encrypting Key, and encrypting a data block in Electronic Code Book mode where the first byte has a value of '3', and the remaining 15 bytes are zero.
If the data stream is an exact number of Data Blocks, the last block will have n-5 == 0.