MarshallSoft AES

(Advanced Encryption Standard)

User's Manual

(AES_REF)

Version 6.0

February 14, 2022

This software is provided as-is.
There are no warranties, expressed or implied.

MarshallSoft Computing, Inc.
Post Office Box 4543
Huntsville AL 35815 USA

Web: http://www.marshallsoft.com

TABLE OF CONTENTS

1 Introduction

The **MarshallSoft Advanced Encryption Standard Library (AES)** is a toolkit that allows software developers to easily implement strong encryption and decryption in a Windows application.

The **MarshallSoft Advanced Encryption Standard Library (AES)** is a component library of functions used to perform encryption and decryption using the 256-bit "Advanced Encryption Standard" (AES) as specified by the U.S. National Institute of Standards and Technology (NIST). See http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

AES is considered "strong encryption" and replaces the previous US encryption standard "Data Encryption Standard" (DES). AES is commonly used by many financial entities such as banks to protect their customer's sensitive information.

Our implementation of the Advanced Encryption Standard has been verified by running the "Advanced Encryption Standard Algorithm Validation Suite" (AESAVS), which can be found at http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf

The **MarshallSoft Advanced Encryption Standard DLL's (AES32.DLL** and **AES64.DLL)** will work under all 32-bit and 64-bit versions of Windows through Windows 11. Both Win32 and Win64 DLL's are included.

The User's Manual applies to the **MarshallSoft Advanced Encryption Standard Library** (**AES**) component library for all supported programming languages. It discusses encryption and decryption processing as well as language independent programming issues and provides purchasing and licensing information.

We have versions of the **MarshallSoft Advanced Encryption Standard** SDK (AES) for C/C++ (AES4C), Delphi (AES4D), Visual Basic (AES4VB), PowerBasic (AES4PB), Visual FoxPro (AES4FP), Visual dBase (AES4DB) and Alaska Xbase++ (AES4XB). The **MarshallSoft AES** DLLs (AES32.DLL and AES64.DLL) can also be used from any language (Power Basic, Visual Basic, ACCESS, EXCEL, VBA, Delphi, Visual FoxPro, COBOL, Xbase++, Visual dBase, Microsoft Office, etc.) capable of calling the Windows API.

Purchase a developer license for one programming language and use it with all others. All versions of the MarshallSoft AES component use the same DLLs (AES32.DLL or AES64.DLL). However, the examples provided for each version are written and tested for the specified programming development language. Development time is shortened because programmers need only to learn one interface.

For the latest version of our **AES** software, see http://www.marshallsoft.com/aes.htm

The best way to get familiar with the **MarshallSoft AES** library is to try out one of the example programs. The example programs are described in the AES4x Programmer's Manual. Where the "x" in AES_4x specifies the host programming language: C for C/C++, VB for Visual Basic, 'PB' for Power Basic, 'D' for Delphi, 'FP' for FoxPro, 'DB' for dBase, and 'X' for XBase.

Legalities

It is illegal to possess strong encryption software in some countries in the world. Do not download or use this software if it is illegal to do so in your country.

In addition, this software cannot be sold to countries on the US Embargo List. See http://www.pmddtc.state.gov/embargoed_countries/index.html

1.1 Documentation Set

The complete set of documentation is provided in Adobe PDF format. This is the third manual (AES_REF.PDF) in the set.

- AES_4x Programmer’s Manual (AES_4x.PDF) where x = C,D,VB,PB,FP,DB,XB
- AES User’s Manual (AES_USR.PDF)
- AES Reference Manual (AES_REF.PDF)

The AES_4x Programmer’s Manual is the programming language specific manual. All language dependent programming issues including installation, compiling and example programs are discussed in this manual. The language specific manuals are as follows:

[NAME]		[DESCRIPTION]
AES_4C	:	AES Programmer's Manual for C/C++
AES_4VB	:	AES Programmer's Manual for Visual Basic
AES_4PB	:	AES Programmer's Manual for Power Basic
AES_4D	:	AES Programmer's Manual for Delphi
AES_4FP	:	AES Programmer's Manual for Visual FoxPro
AES_4DB	:	AES Programmer's Manual for Visual dBase
AES_4XB	:	AES Programmer's Manual for XBase++

The AES User’s Manual (AES_USR.PDF) discusses encryption/decryption programming issues. Technical support, licensing and purchasing information is also provided. Read this manual after reading the AES Programmer’s Manual.

The AES Reference Manual (AES_REF.PDF) contains details on each individual AES function.

All documentation can also be accessed online at `http://www.marshallsoft.com/advanced-encryption-standard.htm`.

1.2 Technical Support

We want you to be successful in developing applications using the **MarshallSoft Advanced Encryption Standard Library**! We are committed to providing the best, most robust library that we can. If you have any suggestions or comments, please let us know.

If you are having a problem using AES, see section 5.0 "Resolving Problems". If the problem cannot be resolved, email us at

`info@marshallsoft.com`

with subject `"AES HELP"`

To avoid having your email **deleted** by our Spam scanners, begin the subject of your email with "AES4C", “AES4VB”, “AES4D” or “MSC HELP”. Zip up any attachments and send plain ASCII text email only.

The latest versions of our products are available on our web site at

`http://www.marshallsoft.com`

Registered users can update (for a period of one year) to the latest AES DLL at

`http://www.marshallsoft.com/update.htm`

1. How to Purchase

A developer license for the **MarshallSoft Advanced Encryption Standard Library** toolkit may be purchased for $119 (USD) for electronic (email) delivery, or $199 (USD) with ANSII C source code for the DLLs. This price is good for one year from the release date.

Order with a credit card or through PayPal at

http://www.marshallsoft.com/order.htm

We accept American Express, VISA, MasterCard, Discover, PayPal, checks in US dollars drawn on a US bank and International Postal Money Orders (such as Western Union).

The registered package includes:

- AES32 and AES64 (for 64-bit programming) libraries without the “evaluation info” screen.
- Free downloadable updates to the registered DLLs for one (1) year.
- Free technical support by email and telephone for one (1) year.

1.3.1 Academic Discount

We offer an "academic price" of 40% off the normal price for prepaid email orders to faculty and full time students currently enrolled in any accredited high school, college, or university. The software must be used for educational purposes. The academic discount does **not** apply to source code.

To qualify for the discount, your school must have a web site and you must have an email address at your school that is not forwarded. When ordering, ask for the "academic discount" or enter "student at" (or "faculty at") and your schools web site address (URL) in the comments field of the order form on our web site order page. Your order will be sent to your email address at your school.

This offer is not retroactive and cannot be used with any other discount. Products bought with academic pricing cannot be used for any commercial purpose nor can the AES DLLs be distributed.

1.3.2 Source Code

Source code is available for the purpose of re-compiling AES32.DLL/AES64.DLL. Source code for the DLL library is standard ANSI C. The source code for AES32.DLL/AES64.DLL is copyrighted by MarshallSoft Computing and may not be released in whole or in part.

There are two ways to order Source Code for the **MarshallSoft Advanced Encryption Standard Library SDK.**

(1) Source Code can be ordered at the same time as the Developer’s License for $195 (for both).

(2) Source Code can be ordered within one year of purchasing a Developer's License for $100. After one year, a Developer’s License update must be purchased prior to purchasing the source code.

1.4 Updates

When a developer license is purchased for the **MarshallSoft Advanced Encryption Standard Library SDK**, the developer will receive the registered DLLs plus a license file (AESxxxx.LIC) that can be used to update the registered DLLs (does not include source code) for a period of one year from purchase. Updates can be downloaded from

    http://www.marshallsoft.com/update.htm

After one year, the developer license must be updated to be able to download updates and receive technical support. The license can be updated for:

- $33 if the update is ordered within one year of the original purchase (or previous update).
- $55 if the update is ordered between one and three years of the original purchase (or previous update).
- $77 if the update is ordered after three years of the original purchase (or previous update).

A license update includes an additional year of technical support and downloadable updates.

Source code previously purchased may be updated for $40 in addition to the cost of the update ($33, $55 or $77).

Note that the registered AES DLLs **do not** expire.

Also see the file UPDATES.TXT.

1.5 Customer ID

The customer ID is the number following the product name (AES) in the license file. For example, Customer 12345 would receive license file **AES12345.LIC**. Provide the Customer ID along with product name in the Subject field of an email when contacting us for technical support (AES4C 12345).

1.6 License File

A license file, AESxxxxx.LIC, where "xxxxx" is the 5 digit customer ID is provided with each developer license. The license file is an encrypted binary file used for updating AES as explained in Section 1.5 "Updates". The license file is required in order to create (or update) the registered DLLs. The license file can be found in the /DLLS directory created after SETUP is run.

1.7 Distribution

In order to run an application (that calls MarshallSoft AES functions) on another computer, the file AES32.DLL or AES64.DLL must be copied to the Windows directory of the other computer. The Windows directory is normally \WINDOWS for Windows 95/98/ME/2003/2012/XP/Vista/Windows 7/Windows 8 and \WINNT for Windows NT/2000. Do not attempt to "register" the DLLs.

1.8 Keycode

When a developer license is purchased, the developer will receive a new set of DLLs and a keycode for the AES DLL's. Pass this keycode as the argument to **aesAttach**. The keycode will be found in the file named "KEYCODE". The keycode for the evaluation version is 0. The keycode for the registered version will be a unique 9 or 10 digit number. Note: Your keycode is NOT your Customer ID/Registration number.

1.9 Dynamic Link Library

The **MarshallSoft Advanced Encryption Standard Library SDK** includes a Win32 [AES32.DLL] and Win64 [AES64.DLL] dynamic link libraries (DLL). A DLL is characterized by the fact that it need not be loaded until required by an application program and that only one copy of the DLL is necessary regardless of the number of application programs that use it. Contrast this to a static library that is bound at link time to each and every application that uses it.

2 MarshallSoft AES Library Application Notes

2.1 AES Control Buffer

Most **MarshallSoft AES** functions use the "AES control buffer". This control buffer is for internal use only and is required for the functions that perform encryption and decryption. The control buffer can reside in the caller's data space or in the AES data space. Normally it is best to allocate the control buffer in the AES data space by passing either a NULL pointer or a string whose first character is an asterisk '*' for the control parameter.

In order to use a control buffer in the caller's space, allocate an array of at least 288 bytes, and use this array for the control parameter in AES functions. Using a control buffer in the caller's program space allows concurrent encryption (or decryption).

2.2 Validation

There are several web sites that can do 256-bit AES encryption, such as:

```
https://www.javainuse.com/aesgenerator
https://ieasynote.com/tools/aes
https://the-x.cn/en-us/cryptography/Aes.aspx
```

Remember that the password phrase is not the same thing as the encryption key itself. Although any 32 bytes of data can be used as the encryption key, normally the encryption key is constructed from the password phrase.

Also see file Validation.txt in the DOCS sub-directory.

Validation of our implementation of AES uses the "known answer tests" (KAT) using values published by the Information Technology Laboratory at the National Institute of Standards and Technology (NIST)

See "The Advanced Encryption Standard Algorithm Validation Suite (AESAVS)
at http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf

A KAT test program (kat.c) for our implementation of AES is available for anyone who is interested.

2.3 Encryption Keys

AES is a "symmetric cipher", which means that the same key is used for both encryption and decryption. Our implementation of AES uses 256-bit (32-byte) encryption keys.

To be effective, the encryption key should not be something that has an obvious pattern that might make it easy to guess. For example, using an 8 character ASCII text string as the key with the remaining 24 bytes set to zero does not make for a good encryption key.

Ideally, one would create the encryption key using true random numbers. The significant downside is that such keys would have to be stored or written down as they would certainly not be easily remembered.

2.4 Key Generation

Encryption keys can be created in several ways. Technically, any buffer of 32 bytes (256-bits) will work. However, it is normally easier to create the encryption key from an ASCII text string using a "password generator" function such as **aesMakeUserKey** that can use up to 42 ASCII text characters to create the 32-byte encryption key.

The pass phrase is used to generate the 32 byte encryption key. Choose a strong pass phrase. This is most easily done by choosing a pass phrase rather than just a single password. For example, it one were to choose a 4 character password, say "Mike", a brute force attack on the password is rather easy. Instead, choose multiple words such as "Doc Holliday was born in Georgia". Alternatively, “salt” the password as described in section 2.13 “Pass Phrase Security”.

There are three algorithms for generating the encryption/decryption key: (1) The "nibble" method (2) the "Sha256" method, and the shared method.

The nibble method creates the key by generating three 8-bit bytes from four 6-bit characters.

```
  aesMakeUserKey(PasswordPhrase, KeyBuffer, AES_NIBBLE_METHOD)
```

The "Sha256" method uses the 256-bit (32-byte) SHA cryptographic hash algorithm.

```
  aesMakeUserKey(PasswordPhrase, KeyBuffer, AES_SHA256_METHOD)
```

The mixed method consists of first applying the nibble method then the SHA 256 method.

```
  aesMakeUserKey(PasswordPhrase, KeyBuffer, AES_MIXED_METHOD)
```

2.5 Session Key

Suppose that one wants to transmit several files across the internet. One possibility would be to encrypt each file using the same encryption key. The biggest disadvantages of this approach is that (1) an adversary could have all the files (that is, a lot of material) for his crypto-analysis attack, and (2) encrypting a file is significantly slower than performing XOR operations on a file.

Another possibility would be to encrypt each file using a different encryption key. The biggest disadvantage to this approach is that many encryption keys must be kept.

However, common practice is to create a "session key" of random values for each file. The session key is then encrypted using a single encryption key. The encrypted session key is then transmitted followed by the file in which each block of 16-bytes (from the file) is XOR'ed with the original (un-encrypted) 16-byte session key.

At the receiving end, the encrypted session key is decrypted so that the file can be recovered by XOR'ing each block of 16 bytes with the session key.

<u>Transmit Side</u>

1. Create random 16-byte session key (SK).
2. Encrypt session key giving encrypted session key (ESK).
3. Transmit ESK
4. For each 16-byte block (FBi) in the file, transmit (FBi XOR SK)

<u>Receive Side</u>

1. Receive encrypted session key ESK.
2. Decrypt giving the original session key SK.
3. For each 16-byte block (FBi XOR SK) received, decrypt Fbi = (Fbi XOR SK) XOR SK.

2.6 Block Padding

The AES block cipher works on units of 16-bytes, known as a "block". A message to be encrypted is first broken into 16-byte blocks. Before encryption with **aesEncryptBlocks**, the final block must be padded if it is less than 16 bytes.

Blocks can be padded with zero’s, spaces, random bytes or using the PKCS7 padding algorithm (in which the pad byte is the number of pad bytes added).

The **aesPadBuffer** function can be used to append padding as necessary. The 2nd argument of the aesAttach() function specifies if zero padding or PKCS7 padding is done when calling **aesEncryptFile**, **aesDecryptFile**, **aesEncryptWrite**, or **aesReadDecrypt** functions.

2.7 Encryption Modes

AES supports two modes of operation: **ECB** and **CBC**

2.7.1 ECB - Electronic Code Book (ECB) Mode

The simplest of the encryption modes is the electronic codebook (ECB) mode. The message is divided into 16-byte blocks and each block is encrypted separately.

2.7.2 CBC - Cipher Block Chaining (CBC) Mode

In CBC mode, each block of plaintext is XOR'ed with the previous ciphertext block before being encrypted. In this way, each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block.

```
  C[i] = EK(P[i] ^ C[i-1]), C[0] = IV  // Encryption (first block = 1)
  P[i] = DK(C[i] ^ C[i-1]), C[0] = IV  // Decryption
```

Initialization Vector

An initialization vector (IV) is a 16-byte block of data that is used in the CBC mode to randomize encryption and therefore produce distinct ciphertexts even if the same plaintext is encrypted multiple times.

The initialization vector usually does not need to be secret. However, it is important that an initialization vector not be reused using the same key. Reusing an initialization vector leaks some information about the first block of plaintext. Furthermore, the initialization vector must be unpredictable at encryption time.

2.8 Encryption Logic

The logic for encrypting data or files is as follows:

1. Call **aesAttach.**
2. Create the 32-byte encryption key or use **aesMakeUserKey** to make it.
3. Initialize AES for encryption using **aesInitAES**.
4. Encrypt your buffer with **aesEncryptBlocks** or **aesEncryptBuffer**,
   or encrypt your file with **aesEncryptFile**.

See the TestAES example program.

2.9 Decryption Logic

The logic for decrypting data or files is as follows:

1. Call **aesAttach.**
2. Create the 32-byte decryption key or use **aesMakeUserKey** to make it.
3. Initialize AES for decryption using **aesInitAES**.
4. Decrypt your buffer with **aesDecryptBlocks** or **aesDecryptBuffer**,
   or decrypt your file with **aesDecryptFile**.

See the TestAES example program.

2.10 Hash Functions

A cryptographic hash function is a mathematical algorithm that maps data to a string of fixed size. Hash functions are one-way functions in that it is infeasible to invert the hashed data.

Refer to functions **aesSha256**, **aesSha256Data**, and **aesSha256File** in the AES Reference Manual **aes_ref.pdf**, which maps buffers of bytes to 256 bit (32-byte) strings.

2.10.1 Message Integrity

Hash functions can be used to verify the integrity of a message. For example, if you email a message along with the hash of the message, the recipient can compute the hash of the received message then compare it to the hash sent along with the message. If the message has been changed, the two hashes will not match.

2.10.2 User Authentication

Hash functions can be used to authenticate a user's password (or pass phrase) if the hash of each legitimate user has been previously stored. The hash of the user's presented password is computed then compared to the previously stored hash value. If they match, the user is authenticated. Thus a user can be authenticated without having to store his password.

2.11 PKCS7 Padding

AES encrypts data in blocks of 16-bytes. If the buffer to be encrypted is a multiple of 16, then a new 16-byte block is appended containing 10H (16 decimal) repeated 16 times.

If the buffer to be encrypted is not a multiple of 16, then N additional bytes are appended to the last block (which is 16 - N encrypted bytes) where each appended byte is the value of N

The padding will thus be one of:

```
01                (1 pad byte)
02 02             (2 pad bytes)
03 03 03          (3 pad bytes)
...
10 10 10 ... 10  (16 pad bytes)
```

Padding is done before encrypting. The last decrypted 16-byte block is always padded, and the last byte in the last block is always the number of padding characters in the last block.

The use of PKCS7 padding is recommended.

2.12 Pass Phrase Selection

It is very important to choose a strong pass phrase rather than just a single password. For example, if one were to choose a 4 character password such as "Mike", a brute force attack on the password (trying all 4 character passwords) is rather easy. Instead, choose multiple words such as "Doc Holliday was born in Georgia". Further, pass phrases are often easier to remember than more cryptic passwords.

Another alternative is to extend the length of the user's short password by "salting" it with randomly generated characters. Refer to function **aesSaltPass**.

2.13 Pass Phrase Security

To date, 256-bit AES is considered unbreakable. However, pass phrase security is always the weak point in any encryption implementation.

Embedding a password or pass phrase in your code or writing it to disk is a security risk if an adversary dumps your executable or disk file. The optimal solution is to first “salt” the password or pass phrase (using **aesSaltPass**) then compute (using **aesSha256Data**) and embed the hash digest of the salted password or pass phrase in your code, or read the hash digest from a disk file. Note that the hash digest can even be public.

See section 2.10 “Hash Functions” above and look through the HashDigest example program.

2.14 Cryptographically Secure RNG

A cryptographically secure pseudo-random number generator (CSPRNG) is one in which, given a sequence of numbers, it is computationally infeasible to predict the next number in the sequence.

The **aesSecureRandom** function employs the **ISAAC** algorithm as based on the public domain code released by Bob Jenkins in March 1996.

https://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator
http://burtleburtle.net/bob/rand/isaac.html

Also refer to the AES Reference Manual (`http://www.marshallsoft.com/aes_ref.pdf`) for details on the **aesSeureRandom** function.

2.15 Base64 Encoding

Base64 is an encoding method that replaces groups of 3 binary bytes with 4 ASCII text bytes. Base64 encoding is a convenient way to express a 32 byte binary encryption key.

Some AES online encryption sites will allow the AES encryption/decryption key to be entered as a Base64 text string.

See **aesEncodeBase64** and **aesDecodeBase64**

2.16 Diffie-Hellman KeyExchange

The Diffie-Hellman algorithm allows two users to exchange data (such as an AES encryption key) securely over a public medium such as the Internet.

For example, consider Pam and Joe who communicate over the internet.

1. Pam wants to send an AES encryption/decryption key over the internet to Joe.
2. Pam creates a private/public key pair on her computer.
3. Joe also creates private/public key pair on his computer.
4. Pam sends Joe her public key and Joe sends Pam his public key.
5. Pam computes her shared key from her private key and Joe's public key.
6. Joe also computes his shared key from his private key and Pam's public key.
7. Pam's shared key will be identical to Joe's shared key.

The public/private key pair is created by **aesMakeKeyPair** and the shared key is created by **aesMakeSharedKey**. The "best practice" is to use the shared AES key just once to exchange the AES pass phrase or encryption key that will be used on a regular basis.

Note that the functions **aesMakeKeyPair** and **aesMakeSharedKey** are very slow as they perform 1024 bit arithmetic.

See the TestDH example program.

3.0 Encryption Topics

3.1 Encryption Keys Embedded in Code

There are significant advantages to embedding encryption keys into executable code, such as:

(1) A binary encryption key can be used instead of a pass phrase since the key doesn't have to be remembered by the user.
(2) Since the encryption key is not physically entered, it can never be seen by someone looking over one's shoulder or compromised with a screenshot.
(3) Users who encrypt messages don't need to know the actual encryption key. An encryption key should be entered into one's code as binary data (not text), and modified in a predetermined way at runtime in several different places in the code before actually being used. Thus, an adversary would have to have a copy of your program and then be able to de-compile it in order to figure out the sequence of steps leading to the encryption key.

3.2 Multi-Factor Encryption

The idea of multi-factor encryption is that encryption is performed more than once using different encryption keys. For example, one encryption key could be embedded in your code and another could be entered as a pass phrase. To encrypt a message, the message is encrypted with the first key then the (encrypted) result is encrypted a second time with the second key. An adversary would need both encryption keys in order to decrypt the message.

3.3 Key Management

From time to time, encryption keys will need to be changed. In general, it is best that the "key administrator" change the keys without revealing the new key (or keys) to others who would be doing the actual encryption. Conversely, in a large enough business, the key administrator shouldn't have access to the data being encrypted.

3.4 Key Distribution to Remote Users

If an encryption key set must be provided to remote users, there are several different approaches that are reasonable unless there are a large number of remote users.

Assume two or more encryption keys that make up a multi-factor encryption key set that must be distributed to remote users. For example, some ways to distribute an encryption key are:

(1) BEST: Use public key Diffie-Hellman encryption (see section 2.16 above).
(2) Email via Gmail since Gmail encrypts email and is not often hacked.
(3) Mailed via US postal or FedEx.
(4) Sent as a text message.
(5) Make a voice call.

Obviously, unless using Diffie-Hellman, one would want to use two or three factor encryption using a different method of distribution for each encryption key in the multi-factor encryption key set.

3.5 Separation of Encryptor & Communicator.

A fundamental principle of encryption security is that the software that does the encryption & decryption should be independent of the software that delivers the encrypted message. Thus, the software that delivers the message, such as Gmail or Yahoo, would not know how to decrypt your encrypted message. The fact that both GMail & Yahoo also encrypt your message provides an additional level of security, although both Gmail & Yahoo have been hacked. Google "has Gmail ever been hacked ?".

4 MarshallSoft AES Library Versions

The **MarshallSoft AES Library** is available in three versions. All three versions have identical functionality.

4.1 Evaluation Version

The evaluation version can be differentiated from the other two versions by:

(1) The registration reminder screen is displayed at startup and every 7 minutes thereafter.

(2) The evaluation version may not be used for commercial purposes.

(3) The evaluation version stops working after 30 days.

3.2 Academic Version

The academic version can be differentiated from the other two versions by:

(1) There is no registration reminder screen.

(2) DLL's purchased with the academic discount may not be distributed, and must be used for educational purposes only.

4.3 Professional Version

The professional version can be differentiated from the other two versions by:

(1) There is no registration reminder screen.

(2) Your compiled DLL may be distributed with your compiled applications as specified by the software license. However, the Keycode to the DLLs cannot be distributed. The Professional version may be used for commercial purposes. Licensing information is provided in Section 5.1

5 Resolving Problems

(1) First, be sure you are passing the proper key code. See Section 1.9, “Keycode”.

(2) If the registration reminder screen (popup) is still being displayed after purchasing a license, the problem is that Windows is finding the evaluation version of the AES DLL before the registered DLL. The solution is to delete (or zip up) all evaluation versions of AES32.DLL or AES64.DLL. Run SETUP and then recompile.

(3) If "error -202" is received when calling **aesAttach**, the problem is that the keycode passed to **aesAttach** does not match the keycode in the DLL. This is caused by (1) using the evaluation keycode (value = 0) with the registered DLL, or (2) using the registered keycode with the evaluation DLL.

(4) If you cannot get your application to run properly, first compile and run the example programs. If you call us to report a possible bug in the library, the first thing we will ask is if the example programs run correctly.

(5) Run the TestAES program. If it fails to run as expected, zip up the Marshall AES log files and send them to

```
                    info@marshallsoft.com
```

with subject

```
                    AES HELP xxxxx
```

where xxxxx is your (6+ digits) customer ID.

(6) Be sure to test the code returned from **MarshallSoft AES** functions. Then call **aesErrorText** to get the text associated with the error code. (The file, ERRORS.TXT, contains a list of all error codes.)

6 Legal Issues

6.1 License

This license agreement (LICENSE) is a legal agreement between you (either an individual or a single entity) and MarshallSoft Computing, Inc. for this software product (SOFTWARE). This agreement also governs any later releases or updates of the SOFTWARE. By installing and using the SOFTWARE, you agree to be bound by the terms of this LICENSE. If you do not agree to the terms of this LICENSE, do not install or use the SOFTWARE

MarshallSoft Computing, Inc. grants a nonexclusive license to use the SOFTWARE to the original purchaser for the purposes of designing, testing or developing software applications. Copies may be made for back-up or archival purposes only. This product is licensed for use by only one developer at a time. All developers working on a project that includes a MarshallSoft Software SDK, even though not working directly with the MarshallSoft SDK, are required to purchase a license for that MarshallSoft product.

The "academic" registered DLL's may not be distributed under any circumstances, nor may they be used for any commercial purpose.

The "professional" registered DLL's may be distributed (royalty free) in object form only, as part of the user's compiled application provided the value of the Keycode is not revealed. The registered DLL's may NOT be distributed as part of any software development system (compiler or interpreter) without our express written permission.

Note that registered DLL’s do not expire. Registered users may download free updates for a period of one year from the date of purchase.

6.2 Warranty

MARSHALLSOFT COMPUTING, INC. DISCLAIMS ALL WARRANTIES RELATING TO THIS SOFTWARE, WHETHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ALL SUCH WARRANTIES ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. NEITHER MARSHALLSOFT COMPUTING, INC. NOR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION, OR DELIVERY OF THIS SOFTWARE SHALL BE LIABLE FOR ANY INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH SOFTWARE EVEN IF MARSHALLSOFT COMPUTING, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR CLAIMS. IN NO EVENT SHALL MARSHALLSOFT COMPUTING, INC.'S LIABILITY FOR ANY SUCH DAMAGES EVER EXCEED THE PRICE PAID FOR THE LICENSE TO USE THE SOFTWARE, REGARDLESS OF THE FORM OF THE CLAIM. THE PERSON USING THE SOFTWARE BEARS ALL RISK AS TO THE QUALITY AND PERFORMANCE OF THE SOFTWARE.

Some states do not allow the exclusion of the limit of liability for consequential or incidental damages, so the above limitation may not apply to you.

This agreement shall be governed by the laws of the State of Alabama and shall inure to the benefit of MarshallSoft Computing, Inc. and any successors, administrators, heirs and assigns. Any action or proceeding brought by either party against the other arising out of or related to this agreement shall be brought only in a STATE or FEDERAL COURT of competent jurisdiction located in Madison County, Alabama. The parties hereby consent to in personam jurisdiction of said courts.

7 MarshallSoft AES Function Summary

Refer to the AES Reference Manual (AES_REF.PDF) for detailed information on the AES functions. A one-line summary of each function follows.

There are 32 functions in the **MarshallSoft AES** library.

aesAttach	Initialize MarshallSoft AES library.
aesByteToHex	Convert bytes to hex characters.
aesDecodeBase64	Decode a base64 encoded data buffer.
aesDecryptBlocks	Decrypt data blocks.
aesDecryptBuffer	Decrypt data buffer.
aesDecryptFile	Decrypt file.
aesEncodeBase64	Base64 encode a data buffer.
aesEncryptBlocks	Encrypt data blocks.
aesEncryptBuffer	Encrypt data buffer.
aesEncryptFile	Encrypt file.
aesEncryptWrite	Encrypt buffer & write to file.
aesErrorText	Get error text.
aesGetInteger	Get Integer parameter for AES processing.
aesGetString	Get AES string parameter.
aesHexToByte	Convert hex characters to bytes.
aesInitAES	Initialize AES for encryption/decryption.
aesMakeKeyPair	Make public/private key pair.
aesMakeRandom	Generate random bytes.
aesMakeSharedKey	Make shared key
aesMakeUserKey	Make encryption key.
aesPadBuffer	Append pad bytes to buffer.
aesReadDecrypt	Read file & decrypt to buffer.
aesRemovePad	Remove PKCS7 padding.
aesSaltPass	Add random characters ("salt") password or pass phrase.
aesSecureRandom	Generate secure random numbers.
aesSetInteger	Set AES parameter.
aesSha256Data	Compute 32-byte SHA-256 hash (from buffer).
aesSha256File	Compute 32-byte SHA-256 hash (from file).
aesShredFile	Shred (overwrite with zeros then delete) file.
aesSleep	Sleep specified number of milliseconds.
aesVerifyControl	Verify integrity of encryption/decryption control buffer.
aesXorBits	Convert bytes to hex characters.

7 AES Error Code List

Negative return codes are errors, as follows:

```
AES_NOT_MULTIPLE       -2 : block not multiple of 16 bytes
AES_BAD_KEY_DIR        -3 : key direction is invalid
AES_BAD_KEY_DATA       -4 : key data is invalid
AES_BAD_CIPHER_MODE    -5 : invalid cipher mode
AES_BAD_CIPHER_STATE   -6 : cipher not initialized
AES_BAD_BLOCK_LENGTH   -7 : invalid block length
AES_NOT_INITIALIZED    -8 : AES control block not initialized
AES_IS_CORRUPTED       -9 : AES control block is corrupted
AES_INTERNAL_ERROR    -10 : AES internal error
AES_BAD_PASS_LEN      -11 : password is too short
AES_CANNOT_OPEN       -12 : cannot open file (for read)
AES_CANNOT_CREATE     -13 : cannot create file
AES_READ_ERROR        -14 : read error
AES_WRITE_ERROR       -15 : write error
AES_BAD_PAD_CHOICE    -16 : not AES_PAD_ZERO, AES_PAD_RANDOM, AES_PAD_SPACE
AES_BAD_HEX_CHAR      -17 : bad hex character
AES_UNEXPECTED_CHAR   -18 : unexpected pass phrase character
AES_ATTACH_CALL       -19 : aesAttach not called
AES_NULL_POINTER      -20 : unexpected null pointer
AES_BAD_METHOD        -21 : not AES_NIBBLE_METHOD, AES_SHA256_METHOD, AES_MIXED_METHOD
AES_BUFFER_TOO_SMALL  -22 : buffer is too small
AES_BUFFER_TOO_BIG    -23 : buffer is too big
AES_PKCS7_ERROR       -24 : PKCS7 padding error
AES_CANNOT_OPEN_WRITE -25 : Cannot open file (for write)
AES_CANNOT_CLOSE      -26 : Cannot close file
AES_CANNOT_DELETE     -27 : Cannot delete file
AES_ABORTED          -201 : AES aborted by user
AES_KEYCODE          -202 : Invalid key code
AES_EXPIRED          -203 : Evaluation version has expired
```