

FathCrypt ActiveX control

By Fath Software

Contents

License agreement	3
Technical support	4
Internet Mail	4
World Wide Web	4
Encryption	5
A Password Based File Encryption	5
Overall Structure	5
The Components	5
Diffie-Hellman	6
RSA	7
AES	7
FathCrypt ASP / scripting usage	8
FathCrypt component reference	8
Constants	9
DigestType	9
Properties	9
Password	9
RsaKeyLen	10
Version	10
Methods	10
Base64Decode	10
Base64Encode	10
DecryptData	11
DecryptFile	11
DecryptFileToMemory	11
DHComputeKey	12
DHSetupAgreement	12

EncryptData.....	12
EncryptFile	13
EncryptFileFromMemory.....	13
GetDigest.....	13
GetFileDigest	14
MakeSFX	14
RsaDecrypt	14
RsaEncrypt	15
RsaGenerateKeys	15
ShredFile	15
Events	15
Status	15

License agreement

This Limited Use Software License Agreement (the "Agreement") is a legal agreement between you, the end-user ("Licensee"), and author. By using this software or storing this program ("FathCrypt component") on a computer hard drive or other media, you are agreeing to be bound by the terms of this Agreement.

You may install trial version of this program to test and evaluate for 30 days; after that time you must either register this program or delete it from your computer hard drive.

The trial version of software may be distributed freely on online services, bulletin boards, or other electronic media as long as the files are distributed in their entirety.

You may not alter this software in any way, including changing or removing any messages or windows.

You may not decompile, reverse engineer, disassemble or otherwise reduce this software to a human perceivable form. You may not modify, rent or resell for profit this software. You may not publicize or distribute any registration code algorithms, information, or registration codes used by this software without permission of author.

Author grants a license to use the enclosed software to the original purchaser.

Customer written applications containing embedded FathCrypt component may be freely distributed, without royalty payments to author, provided that such distributed product is bound into these applications in such a way so as to prohibit separate use in design mode, and that such product is distributed only in conjunction with the customer's own software product.

This control may be used as a constituent control only if the compound control thus created is distributed with and as an integral part of an application. This license may be transferred to a third party only if all existing copies of the software and its documentation are also transferred.

This product is licensed for use by only one developer at a time. Author expressly prohibits installing this product on more than one computer if there is any chance that both copies will be used simultaneously. This restriction also extends to installation on a network server, if more than one workstation will be accessing the product. All developers working on a project which includes an FathCrypt component product, even though not working directly with the product, are required to purchase a license for that product.

This software is provided "as is". Author makes no warranty, expressed or implied, with regard to the software. All implied warranties, including the warranties of merchantability and fitness for a particular use, are hereby excluded. AUTHOR'S LIABILITY IS LIMITED TO THE PURCHASE PRICE. Under no circumstances shall author of this product be liable for any incidental or consequential damages, nor for any damages in excess of the original purchase price.

Plain English version:

We require that you purchase one copy of a control per developer on a project. If this is met, you may distribute the control with your application royalty free. You may never distribute the LIC file. You may not change the product in any way that removes or changes the requirement of a license file.

We encourage the use of our controls as constituent controls when the compound controls you create are an integral part of your application. But we don't allow distribution of our controls as constituents of other controls when the compound control is not part of an application. The reason we need to have this restriction is that without it someone might decide to use our control as a constituent, add some trivial (or even non-trivial) enhancements and then sell the compound control. Obviously there would be little difference between that and just plain reselling our control.

If you have purchased the source code, you may not re-distribute the source code either (nor may you copy it into your own project). Author retains the copyright to the source code.

Your license is transferable. The original purchaser of the product must make the transfer request. Contact us for further information.

The sample versions of our products are intended for evaluation purposes only. You may not use the sample version to develop completed applications.

Technical support

Internet Mail

You can send E-mail to technical support via the Internet. Messages should be addressed to support@fathsoft.com .

World Wide Web

The Fath Software web site is located at <http://www.fathsoft.com> . You can access our web site for up-to-date information about the product, support forum, knowledgebase and news.

Encryption

A Password Based File Encryption

The Password Based File Encryption Utility used in FathCrypt combines AES code with HMAC-SHA1 for authentication and key derivation from a password and a salt according to RFC2898.

Users should be aware that this code is not designed to operate in a hostile machine environment (that is, one in which other running processes have to be treated as hostile).

Overall Structure

The essential components in the design are:

AES in Counter (CTR) mode for encryption

HMAC-SHA1 for authentication

Key derivation from a password and a salt according to RFC2898

Salt Length	AES Key Length	HMAC-SHA1 Key Length	Password Verifier Length	Authentication Field Length	File Length Overhead
16	32	32	2	10	28

The encrypted file format is:

- the password salt bytes (16 bytes)
- the password verifier (2 bytes)
- the encrypted file contents (the same length as the file itself)
- the authentication field (10 bytes)

Authentication is performed on the encrypted file contents (that is after encryption).

Please note that the use of HMAC-SHA1 for key derivation places a limit on the resulting key space that can be as low as 160 bits. Hence, although 192 and 256 bit AES keys are supported, users should be aware that these keys will not offer their full theoretical strength against brute force key searches. However brute force key searches over such huge key spaces are of no practical significance. In consequence the design is directed instead to the prevention of searches in the much smaller key spaces that experience suggests are typical for password derived keys.

The Components

The Pseudo Random Data Pool

Pseudo random numbers are needed in the design to provide the bytes of the password salt. However, since the salt is not secret, we are interested in randomness rather than secrecy.

FathCrypt is using the Time Stamp Counter as an entropy source.

Password Based Key Derivation

The supplied password and a salt value are converted into two keys (one for AES and one for HMAC-SHA1) and a password verification value using the approach set out in RFC 2898.

The two byte password verifier is placed in the file after the password salt. This is generated from the password and salt value on encryption and stored in the file. On decryption it is again generated from the password and salt but is then tested against the stored value to determine if the password is correct.

The main aim of the password verifier is to cope with huge files where there may be practical reasons for not wanting to decrypt the whole file before discovering that the password is incorrect.

Diffie-Hellman

Diffie-Hellman (D-H) key exchange is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher (like AES).

Diffie-Hellman is not an encryption mechanism as we normally think of them in that we do not typically use it to encrypt data. Instead, it is a method to securely exchange the keys that encrypt data. Diffie-Hellman accomplishes this secure exchange by creating a “shared secret” (sometimes called a “Key Encryption Key” or KEK) between two devices. The shared secret then encrypts the symmetric key for secure transmittal. The symmetric key is sometimes called a Traffic Encryption Key (TEK) or Data Encryption Key (DEK). Therefore, the KEK provides for secure delivery of the TEK, while the TEK provides for secure delivery of the data itself.

The process begins when each side of the communication generates a private key and public key (using DHSetupAgreement method in FathCrypt). Public key is a derivative of the private key. The two systems then exchange their public keys. Each side of the communication now has their own private key and the other systems public key.

Noting that the public key is a derivative of the private key is important – the two keys are mathematically linked.

Once the key exchange is complete, the process continues. The DHComputeKey method generates “shared secrets” – identical cryptographic key shared by each side of the communication. By running the mathematical operation against your own private key and the other side’s public key, you generate a value. When the distant end runs the same operation against your public key and their own private key, they also generate a value. The important point is that the two values generated are identical. They are the “Shared Secret” that can be used as a key to encrypt information between systems.

More info can be found at: <http://tools.ietf.org/html/rfc2631>

RSA

In cryptography, RSA is an algorithm for public-key cryptography. It was the first algorithm known to be suitable for signing as well as encryption, and one of the first great advances in public key cryptography. RSA is widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys and the use of up-to-date implementations

RSA involves a public key and a private key. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key.

The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

The key length for a secure RSA transmission is typically 1024 bits. 512 bits is now no longer considered secure. For more security or if you are paranoid, use 2048 or even 4096 bits. With the faster computers available today, the time taken to encrypt and decrypt even with a 4096-bit modulus really isn't an issue anymore. In practice, it is still effectively impossible for you or I to crack a message encrypted with a 512-bit key. An organization like the NSA who has the latest supercomputers can probably crack it by brute force in a reasonable time, if they choose to put their resources to work on it. The longer your information is needed to be kept secure, the longer the key you should use. Keep up to date with the latest recommendations in the security journals.

AES

In cryptography, the Advanced Encryption Standard (AES), also known as Rijndael, is a block cipher adopted as an encryption standard by the U.S. government. It has been analyzed extensively and is now used worldwide, as was the case with its predecessor, the Data Encryption Standard (DES). AES was announced by National Institute of Standards and Technology (NIST) as U.S. FIPS PUB 197 (FIPS 197) on November 26, 2001 after a 5-year standardization process (see Advanced Encryption Standard process for more details). It became effective as a standard May 26, 2002. As of 2006, AES is one of the most popular algorithms used in symmetric key

cryptography. It is available by choice in many different encryption packages. This marks the first time that the public has had access to a cipher approved by NSA for top secret information.

As of 2006, the only successful attacks against AES implementations have been side channel attacks. The National Security Agency (NSA) reviewed all the AES finalists, including Rijndael, and stated that all of them were secure enough for US Government non-classified data. In June 2003, the US Government announced that AES may be used for classified information:

The design and strength of all key lengths of the AES algorithm (i.e., 128, 192 and 256) are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 key lengths.

More info can be found at http://en.wikipedia.org/wiki/Advanced_Encryption_Standard .

FathCrypt ASP / scripting usage

Here's VBScript example:

```
Dim c,s1,s2
Set c=CreateObject("FathCrypt.CFathCrypt.1")
c.Password="123"
s1=c.EncryptString("test string")
s2=c.DecryptString(s1)
MsgBox s2

c.EncryptFile srcfile, destfile
c.DecryptFile srcfile, destfile
```

FathCrypt component reference

DigestType	Digest type constants
Password	Password to use.
RsaKeyLen	Key length used for RSA and DH methods. Supported values are: 1024, 2048 and 4096.
Version	Return control version string.
Base64Decode	Decode a Base64-encoded string into VARIANT (Byte Array)
Base64Encode	Encode a VARIANT (Byte Array) into string using Base64 scheme.
DecryptData	Decrypt string or byte array
DecryptFile	Decrypt file.
DecryptFileToMemory	Decrypt file into memory variable (String)
DHComputeKey	Computer secret key out of other party public value and your private

DHSetupAgreement	value. DHParams parameter is the same one returned by DHSetupAgreement call. AgreedKey return value is Base64 encoded. Create public/private value pair for Diffie-Hellman key-agreement protocol. After that, PrivateKey and DHParams can be used in a call to DHComputeKey method to compute a secret key between peers.
EncryptData	Returns Base64-encoded encrypted string or byte array
EncryptFile	Encrypt a file using AES-256.
EncryptFileFromMemory	Encrypt memory variable (String) into a file
GetDigest	Returns message digest. DigestType can be MD5, SHA-1, SHA2-256, SHA2-512. Data argument can be of type String or Byte Array.
GetFileDigest	Returns file digest. DigestType can be MD5, SHA-1, SHA2-256, SHA2-512.
MakeSFX	Create self-decrypting file.
RsaDecrypt	Decrypt a string using private key.
RsaEncrypt	Encrypt a string using public key.
RsaGenerateKeys	Generate public/private key pair for RSA. Key length is specified by RsaKeyLen property.
ShredFile	Reliably destroys file data.
Status	Reports encryption/decryption progress.

Constants

DigestType

MD5	0
SHA1	1
SHA2_256	2
SHA2_512	3

Properties

Password

Property Password As String

Password to use.

FathCrypt use this string as a password input parameter to RFC-2898 key-derivation. Key length must be in range 1 to 128 chars.

See <http://www.ietf.org/rfc/rfc2898.txt> for more info.

Crypt.Password = "hardtoguess"

RsaKeyLen

Property RsaKeyLen As Long

Key length used for RSA and DH methods. Supported values are: 1024, 2048 and 4096.

Version

Property Version As String

Return control version string.

Methods

Base64Decode

Function Base64Decode (Data As String)

Decode a Base64-encoded string into VARIANT (Byte Array)

For more info about base64, see <http://en.wikipedia.org/wiki/Base64> .

Base64Encode

Function Base64Encode (Data) As String

Encode a VARIANT (Byte Array) into string using Base64 scheme.

For more info about base64, see <http://en.wikipedia.org/wiki/Base64> .

DecryptData

Function DecryptData (SrcData)

Decrypt string or byte array

SrcData parameter can be a string or a byte array.

This method returns VARIANT of the same data type as SrcData parameter.

```
Text1.Text = FathCrypt.DecryptData(x)
```

DecryptFile

Function DecryptFile (SrcFile As String, DestFile As String) As Long

Decrypt file.

SrcFile is a full path to a encrypted source file.

DestFile is a full path to a file to make.

See also the Status event.

Return values are:

0 Unsuccessful

-1 Success

DecryptFileToMemory

Function DecryptFileToMemory (SrcFile As String) As String

Decrypt file into memory variable (String)

SrcFile is a full path to a encrypted source file.

Return value is a decrypted text.

DHComputeKey

Function DHComputeKey (OtherPublicValue As String, PrivateValue As String, DHParams As String, AgreedKey As String) As Long

Compute secret key out of other party public value and your private value. DHParams parameter is the same one returned by DHSetupAgreement call. AgreedKey return value is Base64 encoded.

For more info, see <http://en.wikipedia.org/wiki/Diffie-Hellman> .

DHSetupAgreement

Function DHSetupAgreement (PublicValue As String, PrivateValue As String, DHParams As String) As Long

Create public/private value pair for Diffie-Hellman key-agreement protocol. After that, PrivateKey and DHParams can be used in a call to DHComputeKey method to compute a secret key between peers.

EncryptData

Function EncryptData (SrcData)

Returns Base64-encoded encrypted string or byte array

SrcData parameter can be a string or a byte array. This method returns VARIANT of the same data type as SrcData parameter.

```
Text2.Text = FathCrypt.EncryptData(Text1.Text)
```

EncryptFile

Function EncryptFile (SrcFile As String, DestFile As String) As Long

Encrypt a file using AES-256.

SrcFile is a full path to a source file.
DestFile is a full path to a encrypted file to make.
See Also the Status event.

Return values

0 Failed
-1 Success

EncryptFileFromMemory

Function EncryptFileFromMemory (SrcString As String, DestFile As String) As Long

Encrypt memory variable (String) into a file

SrcString is a string (text) variable.
DestFile is a full path to a encrypted file to make.

Return values are:

0 Unsuccessful
-1 Success

GetDigest

Function GetDigest (DigestType As DigestType, Data)

Returns message digest. DigestType can be MD5, SHA-1, SHA2-256, SHA2-512. Data argument can be of type String or Byte Array.

GetFileDigest

Function GetFileDigest (DigestType As Long, FilePath As String) As String

Returns file digest. DigestType can be MD5, SHA-1, SHA2-256, SHA2-512.

MakeSFX

Function MakeSFX (CryptedFile As String, DestinationExe As String) As Long

Create self-decrypting file.

CryptedFile is a full path to encrypted file you want to wrap into exe.

DestinationExe is a full path to an exe program file to make.

When decrypting, SFX module will create file of the same name as SFX module, but without .exe extension. So, if you want exe to decrypt myfile.zip, you should name it myfile.zip.exe .

Return values are:

0 Unsuccessful

-1 Success

RsaDecrypt

Function RsaDecrypt (InStr As String, OutStr As String, PrivateKey As String) As Long

Decrypt a string using private key.

RsaEncrypt

Function RsaEncrypt (InStr As String, OutStr As String, PublicKey As String) As Long

Encrypt a string using public key.

RsaGenerateKeys

Function RsaGenerateKeys (PrivateKey As String, PublicKey As String) As Long

Generate public/private key pair for RSA. Key length is specified by RsaKeyLen property.

ShredFile

Function ShredFile (FilePath As String, nTimes As Long) As Long

This method irrecoverably destroys file data by using multiple overwrites as specified in the Department of Defense (DOD 5220.22-M) recommendation for top security.

Events

Status

Sub Status (Progress As Long, Cancel As Long)

Reports encryption/decryption progress.