

MBS Xojo Encryption Kit

Version 1.0, © 2014-2019 by Christian Schmitz

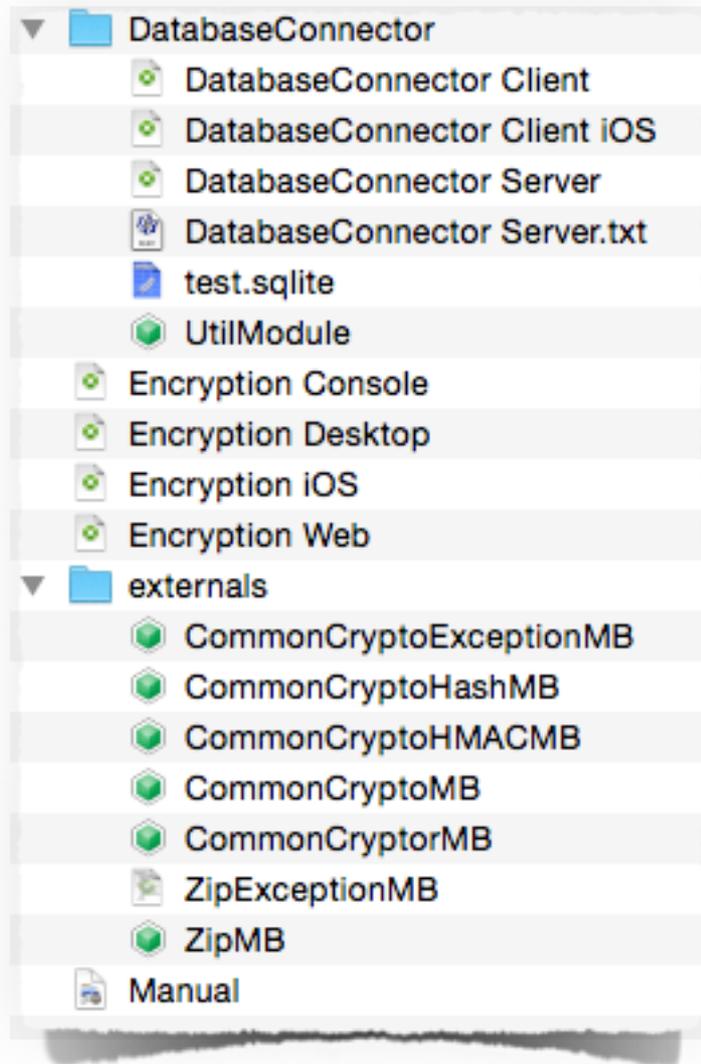
About the MBS Xojo Encryption Kit	2
CommonCrypto	3
Zip	4
Database Connector	5
Interfaces	6
CommonCryptoExceptionMB	6
CommonCryptoHashMB	6
CommonCryptoHMACMB	7
CommonCryptoMB	8
CommonCryptorMB	13
ZipMB	18
ZipExceptionMB	18
AES with 28 rounds	19
Example for AES CBC	20
Version History	21
Installation	22
Requirements	23
License	24
Contact	25

About the MBS Xojo Encryption Kit

The MBS Xojo Encryption Kit provides you with a few useful classes and modules to easily add encryption to your Xojo iOS application.

The Kit contains:

- Wrapper for OS X and iOS CommonCrypto framework
- Compression/Decompression functions using zlib
- Database Connector for encrypted database queries from iOS to server app



CommonCrypto

The Encryption Kit contains a complete wrapper for CommonCrypto, Apple's encryption library on OS X and iOS:

Features

- Hashes with various algorithms:
MD2, MD4, MD5, SHA1, SHA224, SHA384, SHA256 and SHA512
- Key derivation with PBKDF2
- Encryption with AES, Blowfish, DES and others
- Hardware accelerated
- Key wrapping with AES

Wrapper Features

- For old and new Xojo framework
- Using exception handling to track error
- Using enums for safer constant passing
- Test code included
- Convenience functions to pass values as text or string instead of MemoryBlocks
- Parameter validation
- All classes with MB postfix to avoid name conflicts.
- Includes EncodeHex function
- Compiles for all targets
- All module definitions are protected to avoid conflicts
- Inline documentation
- Full Source code, no encryption
- Works for 32bit and 64bit targets.
- Projects for Desktop, iOS, Web and Console using our classes.

Zip

For our database connector we added compression functions, so we can compress our recordsets for the transmission.

- Zlib wrapper for Compression and Decompression of memory blocks in memory.
- for Win/Linux with MBS Compression Plugin

Database Connector

As Xojo for iOS does not support direct database server queries, we provide a sample implementation for a secure connection. Database query is encoded with JSON and sent to server compressed and encrypted. Server will connect to database, run the query, pack the recordset and send it back to client.

Features Server:

- Console application for running in background on server
- Reads preferences file for database name, password, user and server port
- Using ServerSocket for taking queries
- Code to Encrypt & Decrypt, Compress & Uncompress messages

Features Client Desktop:

- Using Sockets for synchronous queries, 3 attempts
- Classes for RecordSet in memory
- Query window to run queries.

Features Client iOS:

- Socket for connecting to server and running queries asynchronously
- Query view to run queries and show result in table.
- Delegate to inform about query finished/failed
- Classes for RecordSet in memory

Possible future features:

- Automatic Key exchange instead of shared key.
- Reuse connection
- Logins

Interfaces

CommonCryptoExceptionMB

Class CommonCryptoExceptionMB **Inherits** RuntimeException
Sub Constructor(m **as** string, e **as** Integer = 0)
The exception class if something went wrong with encryption
Sub Constructor(m **as** text, e **as** Integer = 0)
The exception class if something went wrong with encryption
End Class

CommonCryptoHashMB

Class CommonCryptoHashMB
Sub Constructor(h **as** CommonCryptoMB.hashes)
Sub Destructor()
Explicit Cleanup
Function Final1() **As** xojo.Core.MemoryBlock
Returns final hash
Function Final2() **As** MemoryBlock
Returns final hash
Function Update(Data **as** MemoryBlock) **As** MemoryBlock
Process some data.

data Data to process.

This can be called multiple times.
Sub Update(Data **as** string)
Process some data.

data Data to process.

This can be called multiple times.
Sub Update(Data **as** text)
Process some data.

data Data to process.

This can be called multiple times.
Sub Update(Data **as** xojo.Core.MemoryBlock)
Process some data.

data Data to process.

This can be called multiple times.
End Class

CommonCryptoHMACMB

Class CommonCryptoHMACMB

Sub Constructor(type **as** CommonCryptoMB.HMacAlgorithm, key **as** MemoryBlock)

Initialize an CCHmacContext with provided raw key bytes.
algorithm HMAC algorithm to perform.
key Raw key bytes.

Sub Constructor(type **as** CommonCryptoMB.HMacAlgorithm, key **as string**)

Initialize an CCHmacContext with provided raw key bytes.
algorithm HMAC algorithm to perform.
key Raw key bytes.

Sub Constructor(type **as** CommonCryptoMB.HMacAlgorithm, key **as text**)

Initialize an CCHmacContext with provided raw key bytes.
algorithm HMAC algorithm to perform.
key Raw key bytes.

Sub Constructor(type **as** CommonCryptoMB.HMacAlgorithm, key **as** xojo.Core.MemoryBlock)

Initialize an CCHmacContext with provided raw key bytes.
algorithm HMAC algorithm to perform.
key Raw key bytes.

Sub Destructor()

Explicit Cleanup

Function Final1() **As** xojo.Core.MemoryBlock

Obtain the final Message Authentication Code.

Function Final2() **As** MemoryBlock

Obtain the final Message Authentication Code.

Sub Update(Data **as** MemoryBlock)

Process some data.

data Data to process.

This can be called multiple times.

Sub Update(Data **as string**)

Process some data.

data Data to process.

This can be called multiple times.

Sub Update(Data **as text**)

Process some data.

data Data to process.

This can be called multiple times.

Sub Update(Data **as** xojo.Core.MemoryBlock)

Process some data.

data Data to process.

This can be called multiple times.

End Class

CommonCryptoMB

Module CommonCryptoMB

```
Const CC_MD2_DIGEST_LENGTH = 16
Const CC_MD4_DIGEST_LENGTH = 16
Const CC_MD5_DIGEST_LENGTH = 16
Const CC_SHA1_DIGEST_LENGTH = 20
Const CC_SHA224_DIGEST_LENGTH = 28
Const CC_SHA256_DIGEST_LENGTH = 32
Const CC_SHA384_DIGEST_LENGTH = 48
Const CC_SHA512_DIGEST_LENGTH = 64
Const kCCAlignmentError = -4303
Const kCCBufferTooSmall = -4301
Const kCCDecodeError = -4304
Const kCCMemoryFailure = -4302
Const kCCOptionECBMode = 2
Const kCCOptionPKCS7Padding = 1
Const kCCOverflow = -4306
Const kCCParamError = -4300
Const kCCRNGFailure = -4307
Const kCCSuccess = 0
Const kCCUnimplemented = -4305
Const kLibrary =
```

Enum CryptBlockSize

```
kCCBlockSizeAES128 = 16
kCCBlockSizeDES = 8
kCCBlockSize3DES = 8
kCCBlockSizeCAST = 8
kCCBlockSizeRC2 = 8
kCCBlockSizeBlowfish = 8
```

End Enum

Enum CryptKeySize

```
kCCKeySizeAES128 = 16
kCCKeySizeAES192 = 24
kCCKeySizeAES256 = 256
kCCKeySizeDES = 8
kCCKeySize3DES = 24
kCCKeySizeMinCAST = 5
kCCKeySizeMaxCAST = 16
kCCKeySizeMinRC4 = 1
kCCKeySizeMaxRC4 = 512
kCCKeySizeMinRC2 = 1
kCCKeySizeMaxRC2 = 128
kCCKeySizeMinBlowfish = 8
kCCKeySizeMaxBlowfish = 56
```

End Enum

Enum CryptoAlgorithm

```
AES128 = 0
AES = 0
DES = 1
ThreeDES = 2
CAST = 3
RC4 = 4
RC2 = 5
Blowfish = 6
```

End Enum

Enum CryptoMode

```
kCCModeECB = 1
kCCModeCBC = 2
kCCModeCFB = 3
kCCModeCTR = 4
kCCModeF8 = 5
kCCModeLRW = 6
kCCModeOFB = 7
kCCModeXTS = 8
kCCModeRC4 = 9
```

kCCModeCFB8 = 10

End Enum

Enum CryptoOperation

Encrypt = 0

Decrypt = 1

End Enum

Enum CryptoPadding

No = 0

PKCS7 = 1

End Enum

Enum HMacAlgorithm

SHA1 = 0

MD5 = 1

SHA256 = 2

SHA384 = 3

SHA512 = 4

SHA224 = 5

End Enum

Enum Hashes

MD2 = 2

MD4 = 4

MD5 = 5

SHA1 = 1

SHA224 = 224

SHA384 = 384

SHA256 = 256

SHA512 = 512

End Enum

Enum PseudoRandomAlgorithm

SHA1 = 1

SHA224 = 2

SHA256 = 3

SHA384 = 4

SHA512 = 5

End Enum

Enum WrappingAlgorithm

AES = 1

End Enum

Protected Function CalibratePBKDF2(passwordLength as Integer, SaltLength as Integer, algorithm as

PseudoRandomAlgorithm, msec as Integer = 1000) As Integer

Determine the number of PRF rounds to use for a specific delay on the current platform.

algorithm Currently only PBKDF2 is available via kCCPBKDF2

passwordLen The length of the text password in bytes.

saltLen The length of the salt in bytes.

prf The Pseudo Random Algorithm to use for the derivation iterations.

derivedKeyLen The expected length of the derived key in bytes.

msec The targetted duration we want to achieve for a key derivation with these parameters.

the number of iterations to use for the desired processing time.

Protected Function Crypt(Operation as CryptoOperation, Algorithm as CryptoAlgorithm, Options as Integer, key as MemoryBlock, Data as MemoryBlock, IV as MemoryBlock = nil) As MemoryBlock

Protected Function Crypt(Operation as CryptoOperation, Algorithm as CryptoAlgorithm, Options as Integer, key as xojo.core.MemoryBlock, Data as xojo.core.MemoryBlock, IV as xojo.core.MemoryBlock = nil) As xojo.core.MemoryBlock

Protected Function EncodeHex(data as xojo.Core.MemoryBlock) As text

encodes MemoryBlock as text

Protected Function HMAC(HashType as HMacAlgorithm, Key as MemoryBlock, Data as MemoryBlock) As

MemoryBlock

Stateless, one-shot HMAC function.

Returns hash as memoryblock

Protected Function HMAC(HashType as HMacAlgorithm, Key as String, Data as string) As MemoryBlock

Stateless, one-shot HMAC function.

Returns hash as memoryblock

Protected Function HMAC(HashType as HMacAlgorithm, Key as Text, Data as text) As

xojo.Core.MemoryBlock

Stateless, one-shot HMAC function.

Returns hash as memoryblock

Protected Function HMAC(HashType as HMacAlgorithm, Key as Xojo.Core.MemoryBlock, Data as xojo.Core.MemoryBlock) As xojo.Core.MemoryBlock

Stateless, one-shot HMAC function.

Returns hash as memoryblock

Protected Function HMacHashSize(h as HMacAlgorithm) As integer

size of hash in bytes

Protected Function Hash(HashType as Hashes, Data as MemoryBlock) As MemoryBlock

one shot hash function

calculates hash for given data

Protected Function Hash(HashType as Hashes, Data as string) As MemoryBlock

one shot hash function

calculates hash for given data

Protected Function Hash(HashType as Hashes, Data as text) As xojo.Core.MemoryBlock

one shot hash function

calculates hash for given data

Protected Function Hash(HashType as Hashes, Data as xojo.Core.MemoryBlock) As xojo.Core.MemoryBlock

one shot hash function

calculates hash for given data

Protected Function KeyDerivationPBKDF2(password as MemoryBlock, salt as MemoryBlock, algorithm as PseudoRandomAlgorithm, rounds as Integer) As MemoryBlock

Derive a key from a text password/passphrase

algorithm Currently only PBKDF2 is available via kCCPBKDF2

password The text password used as input to the derivation function. The actual octets present in this string will be used with no additional processing. It's extremely important that the same encoding and normalization be used each time this routine is called if the same key is expected to be derived.

salt The salt byte values used as input to the derivation function.

prf The Pseudo Random Algorithm to use for the derivation iterations.

rounds The number of rounds of the Pseudo Random Algorithm to use.

The following values are used to designate the PRF:

- * PseudoRandomAlgorithm.SHA1
- * PseudoRandomAlgorithm.SHA224
- * PseudoRandomAlgorithm.SHA256
- * PseudoRandomAlgorithm.SHA384
- * PseudoRandomAlgorithm.SHA512

Raises exception with Parameter error in case of bad values for the password, salt, and unwrapped key pointers as well as a bad value for the prf function.

PS: Maybe before putting password or salt here, push them through hash function first?

Protected Function KeyDerivationPBKDF2(password as xojo.core.MemoryBlock, salt as xojo.core.MemoryBlock, algorithm as PseudoRandomAlgorithm, rounds as Integer) As xojo.core.MemoryBlock

Derive a key from a text password/passphrase

algorithm Currently only PBKDF2 is available via kCCPBKDF2

password The text password used as input to the derivation function. The actual octets present in this string will be used with no additional processing. It's extremely important that the same encoding and normalization be used each time this routine is called if the same key is expected to be derived.

salt The salt byte values used as input to the derivation function.

prf The Pseudo Random Algorithm to use for the derivation iterations.

rounds The number of rounds of the Pseudo Random Algorithm to use.

The following values are used to designate the PRF:

- * PseudoRandomAlgorithm.SHA1
- * PseudoRandomAlgorithm.SHA224
- * PseudoRandomAlgorithm.SHA256
- * PseudoRandomAlgorithm.SHA384
- * PseudoRandomAlgorithm.SHA512

Raises exception with Parameter error in case of bad values for the password, salt, and unwrapped key pointers as well as a bad value for the prf function.

PS: Maybe before putting password or salt here, push them through hash function first?

Protected Function `SymmetricKeyUnwrap(algorithm as WrappingAlgorithm, IV as MemoryBlock, kek as MemoryBlock, wrappedKey as MemoryBlock) As MemoryBlock`
Unwrap a symmetric key with a Key Encryption Key (KEK).

`algorithm` Currently only AES Keywrapping (rfc3394) is available via `WrappingAlgorithm.AES`
`iv` The initialization value to be used. `rfc3394_iv1` is available as a constant for the standard IV to use.

`kek` The Key Encryption Key to be used to unwrap the raw key.
`wrappedKey` The wrapped key bytes.

The algorithm chosen is determined by the algorithm parameter and the size of the key being wrapped (ie aes128 for 128 bit keys).

May raise `CommonCryptoExceptionMB` with `BufferTooSmall` indicates insufficient space in the `rawKey` buffer.

`ParamError` can result from bad values for the `kek`, `rawKey`, and `wrappedKey` key pointers.

Protected Function `SymmetricKeyUnwrap(algorithm as WrappingAlgorithm, IV as xojo.Core.MemoryBlock, kek as xojo.Core.MemoryBlock, wrappedKey as xojo.Core.MemoryBlock) As xojo.Core.MemoryBlock`
Unwrap a symmetric key with a Key Encryption Key (KEK).

`algorithm` Currently only AES Keywrapping (rfc3394) is available via `WrappingAlgorithm.AES`
`iv` The initialization value to be used. `rfc3394_iv1` is available as a constant for the standard IV to use.

`kek` The Key Encryption Key to be used to unwrap the raw key.
`wrappedKey` The wrapped key bytes.

The algorithm chosen is determined by the algorithm parameter and the size of the key being wrapped (ie aes128 for 128 bit keys).

May raise `CommonCryptoExceptionMB` with `BufferTooSmall` indicates insufficient space in the `rawKey` buffer.

`ParamError` can result from bad values for the `kek`, `rawKey`, and `wrappedKey` key pointers.

Protected Function `SymmetricKeyWrap(algorithm as WrappingAlgorithm, IV as MemoryBlock, kek as MemoryBlock, rawKey as MemoryBlock) As MemoryBlock`
Wrap a symmetric key with a Key Encryption Key (KEK).

`algorithm` Currently only AES Keywrapping (rfc3394) is available via `WrappingAlgorithm.AES`
`iv` The initialization value to be used. `rfc3394iv1` is available as a constant for the standard IV to use.

`kek` The Key Encryption Key to be used to wrap the raw key.
`rawKey` The raw key bytes to be wrapped.

Returns the wrapped key as memoryblock.

Can raise `CommonCryptoExceptionMB` with `BufferTooSmall` indicates insufficient space in the `wrappedKey` buffer.

`ParamError` can result from bad values for the `kek`, `rawKey`, and `wrappedKey` key pointers.

Protected Function `SymmetricKeyWrap(algorithm as WrappingAlgorithm, IV as xojo.Core.MemoryBlock, kek as xojo.Core.MemoryBlock, rawKey as xojo.Core.MemoryBlock) As xojo.Core.MemoryBlock`
Wrap a symmetric key with a Key Encryption Key (KEK).

`algorithm` Currently only AES Keywrapping (rfc3394) is available via `WrappingAlgorithm.AES`
`iv` The initialization value to be used. `rfc3394iv1` is available as a constant for the standard IV to use.

`kek` The Key Encryption Key to be used to wrap the raw key.
`rawKey` The raw key bytes to be wrapped.

Returns the wrapped key as memoryblock.

Can raise `CommonCryptoExceptionMB` with `BufferTooSmall` indicates insufficient space in the `wrappedKey` buffer.

`ParamError` can result from bad values for the `kek`, `rawKey`, and `wrappedKey` key pointers.

Protected Sub `TestRandom()`
test random function

Protected Sub `checkResult(e as Int32)`

raises exception in case something went wrong

Protected Function generateBytes1(count **as integer**) **As** xojo.core.MemoryBlock

Return random bytes in a buffer allocated by the caller.

The PRNG returns cryptographically strong random bits suitable for use as cryptographic keys, IVs,

nonces etc.

count Number of random bytes to return.

Returns MemoryBlock with data or raises exception

available in Mac OS X 10.10 or iOS 8.0

Protected Function generateBytes2(count **as integer**) **As** MemoryBlock

Return random bytes in a buffer allocated by the caller.

The PRNG returns cryptographically strong random bits suitable for use as cryptographic keys, IVs,

nonces etc.

count Number of random bytes to return.

Returns MemoryBlock with data or raises exception

available in Mac OS X 10.10 or iOS 8.0

Protected Function rfc3394iv1() **As** xojo.Core.MemoryBlock

returns RFC 3394 Initial Vector

Protected Function rfc3394iv2() **As** MemoryBlock

returns RFC 3394 Initial Vector

Protected Sub test()

Protected Sub testCrypt()

test encryption

Protected Sub testHMAC()

test hmac functions

Protected Sub testHash()

test hash functions

Protected Sub testKeyGen()

test key derivation

Protected Sub testKeyWrap()

Note "Info"

This is a complete wrapper for CommonCrypto, Apple's encryption library on OS X and iOS

Features

- * Hashes with various algorithms
- * Key derivation with PBKDF2
- * Encryption with AES, Blowfish, DES and others
- * Hardware accelerated
- * Key wrapping with AES

Wrapper Features

- * For old and new framework
- * using exception handling to track error
- * using enums for safer constant passing
- * test code included
- * convenience functions to pass values as text or string instead of MemoryBlocks
- * parameter validation
- * all classes with MB postfix to avoid conflicts
- * EncodeHex function
- * Compiles for all targets
- * All module definitions are protected to avoid conflicts
- * Inline documentation
- * Full Source code, no encryption

Note "License"

todo

End Module

CommonCryptorMB

Class CommonCryptorMB

Sub Constructor(Operation **as** CommonCryptoMB.CryptoOperation, Algorithm **as** CommonCryptoMB.CryptoAlgorithm, Options **as Integer**, key **as** MemoryBlock, iv **as** MemoryBlock = **nil**)
Create a cryptographic context.

op Defines the basic operation: Encrypt or Decrypt.
alg Defines the algorithm.
options A word of flags defining options. can be 0, kCCOptionPKCS7Padding and/or

kCCOptionECBMode.

key Raw key material. Must be appropriate in length for the selected operation and algorithm.

Some algorithms provide for varying key lengths.

iv Initialization vector, optional. Used by block ciphers when Cipher Block Chaining (CBC)

mode is enabled. If present, must be the same length as the selected algorithm's block size.

If CBC mode is selected (by the absence of the kCCOptionECBMode bit in the options flags) and no IV is present, a NULL (all zeroes) IV will be used.

This parameter is ignored if ECB mode is used or if a stream cipher algorithm is selected.

Sub Constructor(Operation **as** CommonCryptoMB.CryptoOperation, Algorithm **as** CommonCryptoMB.CryptoAlgorithm, Options **as Integer**, key **as** Xojo.Core.MemoryBlock, iv **as** xojo.core.MemoryBlock = **nil**)
Create a cryptographic context.

op Defines the basic operation: Encrypt or Decrypt.
alg Defines the algorithm.
options A word of flags defining options. can be 0, kCCOptionPKCS7Padding and/or

kCCOptionECBMode.

key Raw key material. Must be appropriate in length for the selected operation and algorithm.

Some algorithms provide for varying key lengths.

iv Initialization vector, optional. Used by block ciphers when Cipher Block Chaining (CBC)

mode is enabled. If present, must be the same length as the selected algorithm's block size.

If CBC mode is selected (by the absence of the kCCOptionECBMode bit in the options flags) and no IV is present, a NULL (all zeroes) IV will be used.

This parameter is ignored if ECB mode is used or if a stream cipher algorithm is selected.

Sub Constructor(Operation **as** CommonCryptoMB.CryptoOperation, Algorithm **as** CommonCryptoMB.CryptoAlgorithm, Options **as Integer**, key **as string**, iv **as** MemoryBlock = **nil**)
Create a cryptographic context.

op Defines the basic operation: Encrypt or Decrypt.
alg Defines the algorithm.
options A word of flags defining options. can be 0, kCCOptionPKCS7Padding and/or

kCCOptionECBMode.

key Raw key material. Must be appropriate in length for the selected operation and algorithm.

Some algorithms provide for varying key lengths.

iv Initialization vector, optional. Used by block ciphers when Cipher Block Chaining (CBC)

mode is enabled. If present, must be the same length as the selected algorithm's block size.

If CBC mode is selected (by the absence of the kCCOptionECBMode bit in the options flags) and no IV is present, a NULL (all zeroes) IV will be used.

This parameter is ignored if ECB mode is used or if a stream cipher algorithm is selected.

Sub Constructor(Operation **as** CommonCryptoMB.CryptoOperation, Algorithm **as** CommonCryptoMB.CryptoAlgorithm, Options **as Integer**, key **as** text, iv **as** xojo.core.MemoryBlock = **nil**)
Create a cryptographic context.

op Defines the basic operation: Encrypt or Decrypt.
alg Defines the algorithm.
options A word of flags defining options. can be 0, kCCOptionPKCS7Padding and/or

kCCOptionECBMode.

key Raw key material. Must be appropriate in length for the selected operation and algorithm.

Some algorithms provide for varying key lengths.

iv Initialization vector, optional. Used by block ciphers when Cipher Block Chaining (CBC)

mode is enabled. If present, must be the same length as the selected algorithm's block size.

If CBC mode is selected (by the absence of the kCCOptionECBMode bit in the options flags) and no IV is present, a NULL (all zeroes) IV will be used.

This parameter is ignored if ECB mode is used or if a stream cipher algorithm is selected.

Sub Constructor(Operation **as** CommonCryptoMB.CryptoOperation, Mode **as** CommonCryptoMB.CryptoMode, Algorithm **as** CommonCryptoMB.CryptoAlgorithm, Padding **as** CommonCryptoMB.CryptoPadding, key **as** MemoryBlock, Tweak **as** MemoryBlock = **nil**, iv **as** MemoryBlock = **nil**, NumRounds **as Integer** = 0, Options **as Integer** = 0)

Create a cryptographic context.

Sub Constructor(Operation as CommonCryptoMB.CryptoOperation, Mode as CommonCryptoMB.CryptoMode, Algorithm as CommonCryptoMB.CryptoAlgorithm, Padding as CommonCryptoMB.CryptoPadding, key as Xojo.Core.MemoryBlock, Tweak as xojo.core.MemoryBlock = nil, iv as xojo.core.MemoryBlock = nil, NumRounds as Integer = 0, Options as Integer = 0)

Create a cryptographic context.

Sub Constructor(Operation as CommonCryptoMB.CryptoOperation, Mode as CommonCryptoMB.CryptoMode, Algorithm as CommonCryptoMB.CryptoAlgorithm, Padding as CommonCryptoMB.CryptoPadding, key as string, Tweak as MemoryBlock = nil, iv as MemoryBlock = nil, NumRounds as Integer = 0, Options as Integer = 0)

Create a cryptographic context.

Sub Constructor(Operation as CommonCryptoMB.CryptoOperation, Mode as CommonCryptoMB.CryptoMode, Algorithm as CommonCryptoMB.CryptoAlgorithm, Padding as CommonCryptoMB.CryptoPadding, key as text, Tweak as xojo.core.MemoryBlock = nil, iv as xojo.core.MemoryBlock = nil, NumRounds as Integer = 0, Options as Integer = 0)

Create a cryptographic context.

Sub Destructor()

Cleanup

Function Final1() As xojo.Core.MemoryBlock

Finish an encrypt or decrypt operation, and obtain the (possible) final data output.

Returns final bytes of the encrypted/decrypted data. This can be empty memoryblock.

Raises exceptions on error. kCCBufferTooSmall indicates insufficient space in the dataOut buffer. kCCAlignmentError When decrypting, or when encrypting with a block cipher with padding disabled, kCCAlignmentError will be returned if the total number of bytes provided to Update is not an integral multiple of the current algorithm's block size. kCCDecodeError Indicates garbled ciphertext or the wrong key during decryption. This can only be returned while decrypting with padding enabled.

Except when kCCBufferTooSmall is returned, the Cryptor can no longer be used for subsequent operations unless Reset() is called on it.

It is not necessary to call Final() when performing symmetric encryption or decryption if padding is disabled, or when using a stream cipher.

It is not necessary to call Final() when aborting an operation.

Function Final2() As MemoryBlock

Finish an encrypt or decrypt operation, and obtain the (possible) final data output.

Returns final bytes of the encrypted/decrypted data. This can be empty memoryblock.

Raises exceptions on error. kCCBufferTooSmall indicates insufficient space in the dataOut buffer. kCCAlignmentError When decrypting, or when encrypting with a block cipher with padding disabled, kCCAlignmentError will be returned if the total number of bytes provided to Update is not an integral multiple of the current algorithm's block size. kCCDecodeError Indicates garbled ciphertext or the wrong key during decryption. This can only be returned while decrypting with padding enabled.

Except when kCCBufferTooSmall is returned, the Cryptor can no longer be used for subsequent operations unless Reset() is called on it.

It is not necessary to call Final() when performing symmetric encryption or decryption if padding is disabled, or when using a stream cipher.

It is not necessary to call Final() when aborting an operation.

Sub Reset(iv as MemoryBlock)

Reinitializes an existing CCCryptorRef with a (possibly) new initialization vector. Not implemented for stream ciphers.

iv Optional initialization vector; if present, must be the same size as the current algorithm's block size.

The the only possible errors are kCCParamError and kCCUnimplemented which cause exceptions.

This can be called on a cryptor with data pending (i.e. in a padded mode operation before Final is called); however any pending data will be lost in that case.

Sub Reset(iv as xojo.core.memoryblock = nil)

Reinitializes an existing CCCryptorRef with a (possibly) new initialization vector. Not implemented for stream ciphers.

iv Optional initialization vector; if present, must be the same size as the current algorithm's block size.

The the only possible errors are kCCParamError and kCCUnimplemented which cause exceptions.

This can be called on a cryptor with data pending (i.e. in a padded mode operation before Final is called); however any pending data will be lost in that case.

Function Update(Data as MemoryBlock) As MemoryBlock

Process (encrypt, decrypt) some data. The result, if any, is returned as memoryblock.

data: Data to process.

Returns data. The result memoryblock can be smaller or bigger due to alignment.

Can raise error. kCCBufferTooSmall indicates insufficient space in the dataOut buffer.

This routine can be called multiple times. The caller does not need to align input data lengths to block buffered as necessary for block ciphers.

When performing symmetric encryption with block ciphers, and padding is enabled via kCCOptionPKCS7Padding, the total number of bytes provided by all the calls to this function when encrypting can be arbitrary (i.e., the total number of bytes does not have to be block aligned). However if padding is disabled, or when decrypting, the total number of bytes does have to be aligned to the block size; otherwise Final() will return kCCAlignmentError.

Generally, when all data has been processed, call Final().

In the following cases, the CCCryptorFinal() is superfluous as it will not yield any data nor return an error:

1. Encrypting or decrypting with a block cipher with padding disabled, when the total amount of data provided to Update() is an integral multiple of the block size.
2. Encrypting or decrypting with a stream cipher.

Function Update(Data as string) As MemoryBlock

Process (encrypt, decrypt) some data. The result, if any, is returned as memoryblock.

data: Data to process.

Returns data. The result memoryblock can be smaller or bigger due to alignment.

Can raise error. kCCBufferTooSmall indicates insufficient space in the dataOut buffer.

This routine can be called multiple times. The caller does not need to align input data lengths to block buffered as necessary for block ciphers.

When performing symmetric encryption with block ciphers, and padding is enabled via kCCOptionPKCS7Padding, the total number of bytes provided by all the calls to this function when encrypting can be arbitrary (i.e., the total number of bytes does not have to be block aligned). However if

sizes; input is

sizes; input is

padding is disabled, or when decrypting, the total number of bytes does have to be aligned to the block size; otherwise Final() will return kCCAlignmentError.

Generally, when all data has been processed, call Final().

In the following cases, the CCCryptorFinal() is superfluous as it will not yield any data nor return an error:

1. Encrypting or decrypting with a block cipher with padding disabled, when the total amount of data provided to Update() is an integral multiple of the block size.
2. Encrypting or decrypting with a stream cipher.

Function Update(Data as text) As xojo.Core.MemoryBlock
Process (encrypt, decrypt) some data. The result, if any, is returned as memoryblock.

data: Data to process.

Returns data. The result memoryblock can be smaller or bigger due to alignment.

Can raise error. kCCBufferTooSmall indicates insufficient space in the dataOut buffer.

This routine can be called multiple times. The caller does not need to align input data lengths to block buffered as necessary for block ciphers.

When performing symmetric encryption with block ciphers, and padding is enabled via kCCOptionPKCS7Padding, the total number of bytes provided by all the calls to this function when encrypting can be arbitrary (i.e., the total number of bytes does not have to be block aligned). However if padding is disabled, or when decrypting, the total number of bytes does have to be aligned to the block size; otherwise Final() will return kCCAlignmentError.

Generally, when all data has been processed, call Final().

In the following cases, the CCCryptorFinal() is superfluous as it will not yield any data nor return an error:

1. Encrypting or decrypting with a block cipher with padding disabled, when the total amount of data provided to Update() is an integral multiple of the block size.
2. Encrypting or decrypting with a stream cipher.

Function Update(Data as xojo.Core.MemoryBlock) As xojo.Core.MemoryBlock
Process (encrypt, decrypt) some data. The result, if any, is returned as memoryblock.

data: Data to process.

Returns data. The result memoryblock can be smaller or bigger due to alignment.

Can raise error. kCCBufferTooSmall indicates insufficient space in the dataOut buffer.

This routine can be called multiple times. The caller does not need to align input data lengths to block buffered as necessary for block ciphers.

When performing symmetric encryption with block ciphers, and padding is enabled via kCCOptionPKCS7Padding, the total number of bytes provided by all the calls to this function when encrypting can be arbitrary (i.e., the total number of bytes does not have to be block aligned). However if padding is disabled, or when decrypting, the total number of bytes does have to be aligned to the block size; otherwise Final() will return kCCAlignmentError.

Generally, when all data has been processed, call Final().

sizes; input is

sizes; input is

In the following cases, the CCCryptorFinal() is superfluous as it will not yield any data nor return an error:

1. Encrypting or decrypting with a block cipher with padding disabled, when the total amount of data provided to Update() is an integral multiple of the block size.
2. Encrypting or decrypting with a stream cipher.

Note "About"

Generic interface for symmetric encryption.

This interface provides access to a number of symmetric encryption algorithms. Symmetric encryption algorithms come in two "flavors" - block ciphers, and stream ciphers. Block ciphers process data (while both encrypting and decrypting) in discrete chunks of data called blocks; stream ciphers operate on arbitrary sized data.

The object declared in this interface, CCCryptor, provides access to both block ciphers and stream ciphers with the same API; however some options are available for block ciphers that do not apply to stream ciphers.

The general operation of a CCCryptor is: initialize it with raw key data and other optional fields with Constructor; process input data via one or more calls to Update(), each of which may result in output data being returned as memoryblock; and obtain possible remaining output data with Final functions. The CCCryptor is disposed of via destructor, or it can be reused (with the same key data as provided to Constructor()) by calling Reset().

One option for block ciphers is padding, as defined in PKCS7; when padding is enabled, the total amount of data encrypted does not have to be an even multiple of the block size, and the actual length of plaintext is calculated during decryption.

Another option for block ciphers is Cipher Block Chaining, known as CBC mode. When using CBC mode, an Initialization Vector (IV) is provided along with the key when starting an encrypt or decrypt operation. If CBC mode is selected and no IV is provided, an IV of all zeroes will be used.

CCCryptor also implements block buffering, so that individual calls to Update() do not have to provide data whose length is aligned to the block size. (If padding is disabled, encrypting with block ciphers does require that the *total* length of data input to Update() call(s) be aligned to the block size.)

A given CCCryptor can only be used by one thread at a time; multiple threads can use safely different CCCryptors at the same time.

Property Algorithm **As** CommonCryptoMB.CryptoAlgorithm

Property Operation **As** CommonCryptoMB.CryptoOperation

Property Options **As Integer**

Property Padding **As** CommonCryptoMB.CryptoPadding

End Class

ZipMB

Module ZipMB

Const kEndOfStream = 1

Const kErrorBuffer = -5

Const kErrorData = -3

Const kErrorOK = 0

Const kErrorOutOfMemory = -4

Const kLibrary =

Protected Function Compress(Data **as** MemoryBlock) **As** MemoryBlock

Compresses the source buffer into the destination buffer.

As you see we always use best compression and calculate buffer size with compressBound

compress returns kErrorOK if success, kErrorOutOfMemory if there was not enough memory, kErrorBuffer if there was not enough room in the output buffer. (should never happen)

Protected Function Compress(t **as** string) **As** MemoryBlock

convert string in current encoding to MemoryBlock

Protected Function Compress(t **as** text) **As** xojo.Core.MemoryBlock

Protected Function Compress(Data **as** xojo.Core.MemoryBlock) **As** xojo.Core.MemoryBlock

Compresses the source buffer into the destination buffer.

As you see we always use best compression and calculate buffer size with compressBound

compress returns kErrorOK if success, kErrorOutOfMemory if there was not enough memory, kErrorBuffer if there was not enough room in the output buffer. (should never happen)

Protected Function Decompress(Data **as** MemoryBlock) **As** string

decompress data and return new MemoryBlock

Protected Function Decompress(Data **as** xojo.Core.MemoryBlock) **As** xojo.Core.MemoryBlock

decompress data and return new MemoryBlock

Protected Sub test()

End Module

ZipExceptionMB

Class ZipExceptionMB **Inherits** RuntimeException

Sub Constructor(m **as** string, e **as** Integer = 0)

The exception class if something went wrong with encryption

Sub Constructor(m **as** text, e **as** Integer = 0)

The exception class if something went wrong with encryption

End Class

AES with 28 rounds

AES can work with 14 rounds (default) or 28 rounds (better).

```
// test encryption with AES 28 rounds

dim HelloWorldText as text = "Hello World, this is just a test."
dim HelloWorldXCMem as xojo.Core.MemoryBlock = _
    xojo.core.TextEncoding.UTF8.ConvertTextToData(HelloWorldText)

dim KeyText as text = "secret"
dim KeyXCMem as xojo.Core.MemoryBlock
    KeyXCMem = CommonCryptoMB.Hash(CommonCryptoMB.Hashes.MD5, KeyText)

// Operation as CommonCryptoMB.CryptoOperation, Mode as CommonCryptoMB.CryptoMode,
// Algorithm as CommonCryptoMB.CryptoAlgorithm, Padding as CommonCryptoMB.CryptoPadding,
// key as Xojo.Core.MemoryBlock, Tweak as xojo.core.Memoryblock = nil, iv as xojo.core.MemoryBlock = nil,
// NumRounds as Integer = 0, Options as Integer = 0

dim tweak as xojo.Core.MemoryBlock
dim iv as xojo.Core.MemoryBlock
dim c as new CommonCryptorMB(CommonCryptoMB.CryptoOperation.Encrypt, _
    CommonCryptoMB.CryptoMode.kCCModeECB, _
    CommonCryptoMB.CryptoAlgorithm.AES, _
    CommonCryptoMB.CryptoPadding.PKCS7, _
    KeyXCMem, _
    tweak, _
    iv, _
    28, 0)

dim m as new xojo.Core.MutableMemoryBlock(0)

m.Append c.Update(HelloWorldXCMem)
m.Append c.Final

Break
```

Example for AES CBC

```
// AES 256-bit CBC test code

system.DebugLog "Testing started..."

dim MyVal as text = "If you can read this text then process of encryption and decryption is working well."
dim mbMyVal as xojo.Core.MemoryBlock = xojo.core.TextEncoding.utf8.ConvertTextToData(MyVal)

dim MyPwd as text = "Passphrase goes here"
dim mbMyPwd as xojo.Core.MemoryBlock =
CommonCryptoMB.Hash(CommonCryptoMB.Hashes.SHA256,MyPwd)

System.DebugLog "Test value and passphrase created."

dim iv as xojo.Core.MemoryBlock
dim cryptor as new CommonCryptorMB(CommonCryptoMB.CryptoOperation.Encrypt,
CommonCryptoMB.CryptoMode.kCCModeCBC, _
CommonCryptoMB.CryptoAlgorithm.AES, CommonCryptoMB.CryptoPadding.PKCS7, mbMyPwd, iv)

dim enData1 as xojo.Core.MemoryBlock = Cryptor.Update(mbMyVal)
dim enData2 as xojo.Core.MemoryBlock = cryptor.Final1
dim enData as new xojo.Core.MutableMemoryBlock(enData1)
enData.Append enData2

System.DebugLog "Encrypted data value created."

System.DebugLog "Encrypted data: '" + CommonCryptoMB.EncodeHex(enData) + "'"

dim decryptor as new CommonCryptorMB(CommonCryptoMB.CryptoOperation.Decrypt,
CommonCryptoMB.CryptoMode.kCCModeCBC, _
CommonCryptoMB.CryptoAlgorithm.AES, CommonCryptoMB.CryptoPadding.PKCS7, mbMyPwd, iv)

dim result1 as xojo.Core.MemoryBlock = Cryptor.Update(enData)
dim result2 as xojo.Core.MemoryBlock = cryptor.Final1
dim result as new xojo.Core.MutableMemoryBlock(result1)
result.Append result2

System.DebugLog "Result: '" + xojo.core.TextEncoding.UTF8.ConvertDataToText(result) + "'"
```

Version History

Tip: If you want to update your existing code with new release, you'd best compare projects with Arbed (<http://www.tempel.org/Arbed>) and copy modifications to your project.

1.1, prerelease

- DatabaseConnector Server now has a switch in UtilModule to decide if you want to use SQLiteDatabase or MBS SQL Plugin.
- DatabaseConnector Server defaults now on OS X to not use plugin for encryption.
- Added EncodeBase64 and DecodeBase64
- Added AES example.
- Updated iOS example for Xojo 2018

1.0, first release

Installation

To get your projects working with this Encryption Kit, you need to follow a few steps.

Drop the folder „externals“ into your project and access all the common crypto or zip modules and classes. Or copy from existing example projects what you need.

Requirements

You need Xojo 2015r1 or newer.

We did not test with older versions, but you can if you need.

The encryption parts for desktop can use MBS Plugins to perform similar operations on Mac OS X, Windows and Linux.

The SQL connection used in the DatabaseConnector uses MBS Xojo SQL Plugin to connect to database.

License

Summary:

- You may use Encryption Kit only with one licensed Xojo installation.
- You agree not to share the Encryption Kit or use someone else's Encryption Kit copy.

Christian Schmitz Software GmbH, of Nickenich Germany is the owner, developer and sole copyright holder of this product, which is licensed -not sold- to you on a non-exclusive basis.

You agree not to share your MBS Xojo Encryption Kit with anyone.

You may transfer your license to another person only after receiving written authorization from Christian Schmitz Software GmbH and only if the recipient agrees to be bound by the terms of this agreement.

Christian Schmitz Software GmbH reserves the right to cancel the license key(s) of any user who Christian Schmitz Software GmbH determines is in violation of this agreement.

THE WARRANTIES IN THIS AGREEMENT REPLACE ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE IS PROVIDED "AS IS" AND Christian Schmitz Software GmbH DISCLAIMS AND EXCLUDES ALL OTHER WARRANTIES. IN NO EVENT WILL Christian Schmitz Software GmbH BE LIABLE FOR ANY SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, EVEN IF WE HAVE KNOWLEDGE OF THE POTENTIAL LOSS OR DAMAGE.

If you are located in Germany this agreement is subject to the laws of Germany. If you are located outside Germany local law may apply. Some states do not allow the exclusion of warranties, so the above exclusion may not apply to you.

Christian Schmitz Software GmbH does not charge royalties or deployment fees for Xojo applications.

Access to updates is included for one year. After that time you can order an update or keep using the old version you have.

Contact

Christian Schmitz Software GmbH
Eckertshohl 22
56645 Nickenich
Germany

Email: support@monkeybreadsoftware.de

Phone: +49 26 32 95 89 55 (Office) or +49 17 58 36 37 10 (Mobile)