cipher

package | standard library

Version: go1.20.1 Latest | Published: Feb 14, 2023 | License: BSD-3-Clause | Imports: 6 | Imported by: 24,817

Details Valid go.mod file ? Redistributable license ? Tagged version ? Stable version ?

Learn more

Repository cs.opensource.google/go/go

Links Report a Vulnerability

Documentation

Documentation

Overview

Package cipher implements standard block cipher modes that can be wrapped around low-level block cipher implementations. See https://csrc.nist.gov/groups/ST/toolkit/BCM/current_modes.html and NIST Special Publication 800-38A.

Index

Examples

Constants

This section is empty.

Variables

This section is empty.

Functions

This section is empty.

Types

type AEAD

added in go1.2

```
type AEAD interface {
    // NonceSize returns the size of the nonce that must be passed to Seal
    // and Open.
    NonceSize() int

    // Overhead returns the maximum difference between the lengths of a
    // plaintext and its ciphertext.
    Overhead() int

    // Seal encrypts and authenticates plaintext, authenticates the
    // additional data and appends the result to dst, returning the updated
    // slice. The nonce must be NonceSize() bytes long and unique for all
    // time, for a given key.
    //
    // To reuse plaintext's storage for the encrypted output, use plaintext[:0]
    // as dst. Otherwise, the remaining capacity of dst must not overlap plaintext.
    Seal(dst, nonce, plaintext, additionalData []byte) []byte

    // Open decrypts and authenticates ciphertext, authenticates the
    // additional data and, if successful, appends the resulting plaintext
    // to dst, returning the updated slice. The nonce must be NonceSize()
    // bytes long and both it and the additional data must match the
    // value passed to Seal.
    //
    // To reuse ciphertext's storage for the decrypted output, use ciphertext[:0]
    // as dst. Otherwise, the remaining capacity of dst must not overlap plaintext.
    //
    // Even if the function fails, the contents of dst, up to its capacity,
    // may be overwritten.
    Open(dst, nonce, ciphertext, additionalData []byte) ([]byte, error)
}
```

AEAD is a cipher mode providing authenticated encryption with associated data. For a description of the methodology, see https://en.wikipedia.org/wiki/Authenticated_encryption.

func NewGCM

added in go1.2

```
func NewGCM(cipher Block) (AEAD, error)
```

NewGCM returns the given 128-bit, block cipher wrapped in Galois Counter Mode with the standard nonce length.

In general, the GHASH operation performed by this implementation of GCM is not constant-time. An exception is when the underlying Block was created by aes.NewCipher on systems with hardware support for AES. See the crypto/aes package documentation for details.

▶ Example (Decrypt)

▶ Example (Encrypt)

func NewGCMWithNonceSize

added in go1.5

```
func NewGCMWithNonceSize(cipher Block, size int) (AEAD, error)
```

NewGCMWithNonceSize returns the given 128-bit, block cipher wrapped in Galois Counter Mode, which accepts nonces of the given length. The length must not be zero.

Only use this function if you require compatibility with an existing cryptosystem that uses non-standard nonce lengths. All other users should use NewGCM, which is faster and more resistant to misuse.

func NewGCMWithTagSize

added in go1.11

```
func NewGCMWithTagSize(cipher Block, tagSize int) (AEAD, error)
```

NewGCMWithTagSize returns the given 128-bit, block cipher wrapped in Galois Counter Mode, which generates tags with the given length.

Tag sizes between 12 and 16 bytes are allowed.

Only use this function if you require compatibility with an existing cryptosystem that uses non-standard tag lengths. All other users should use NewGCM, which is more resistant to misuse.

type Block

```
type Block interface {
    // BlockSize returns the cipher's block size.
    BlockSize() int

    // Encrypt encrypts the first block in src into dst.
    // Dst and src must overlap entirely or not at all.
    Encrypt(dst, src []byte)

    // Decrypt decrypts the first block in src into dst.
    // Dst and src must overlap entirely or not at all.
    Decrypt(dst, src []byte)
}
```

A Block represents an implementation of block cipher using a given key. It provides the capability to encrypt or decrypt individual blocks. The mode implementations extend that capability to streams of blocks.

type BlockMode

```
type BlockMode interface {
    // BlockSize returns the mode's block size.
    BlockSize() int

    // CryptBlocks encrypts or decrypts a number of blocks. The length of
    // src must be a multiple of the block size. Dst and src must overlap
    // entirely or not at all.
    //
    // If len(dst) < len(src), CryptBlocks should panic. It is acceptable
    // to pass a dst bigger than src, and in that case, CryptBlocks will
    // only update dst[:len(src)] and will not touch the rest of dst.
    //
    // Multiple calls to CryptBlocks behave as if the concatenation of
    // the src buffers was passed in a single run. That is, BlockMode
    // maintains state and does not reset at each CryptBlocks call.
    CryptBlocks(dst, src []byte)
}
```

A BlockMode represents a block cipher running in a block-based mode (CBC, ECB etc).

func NewCBCDecrypter

```
func NewCBCDecrypter(b Block, iv []byte) BlockMode
```

NewCBCDecrypter returns a BlockMode which decrypts in cipher block chaining mode, using the given Block. The length of iv must be the same as the Block's block size and must match the iv used to encrypt the data.

▶ Example

func NewCBCEncrypter

```
func NewCBCEncrypter(b Block, iv []byte) BlockMode
```

NewCBCEncrypter returns a BlockMode which encrypts in cipher block chaining mode, using the given Block. The length of iv must be the same as the Block's block size.

▶ Example

type Stream

```
type Stream interface {
    // XORKeyStream XORs each byte in the given slice with a byte from the
    // cipher's key stream. Dst and src must overlap entirely or not at all.
    //
    // If len(dst) < len(src), XORKeyStream should panic. It is acceptable
    // to pass a dst bigger than src, and in that case, XORKeyStream will
    // only update dst[:len(src)] and will not touch the rest of dst.
    //
    // Multiple calls to XORKeyStream behave as if the concatenation of
    // the src buffers was passed in a single run. That is, Stream
    // maintains state and does not reset at each XORKeyStream call.
    XORKeyStream(dst, src []byte)
}
```

A Stream represents a stream cipher.

func NewCFBDecrypter

```
func NewCFBDecrypter(block Block, iv []byte) Stream
```

NewCFBDecrypter returns a Stream which decrypts with cipher feedback mode, using the given Block. The iv must be the same length as the Block's block size.

▶ Example

func NewCFBEncrypter

```
func NewCFBEncrypter(block Block, iv []byte) Stream
```

NewCFBEncrypter returns a Stream which encrypts with cipher feedback mode, using the given Block. The iv must be the same length as the Block's block size.

▶ Example

func NewCTR

```
func NewCTR(block Block, iv []byte) Stream
```

NewCTR returns a Stream which encrypts/decrypts using the given Block in counter mode. The length of iv must be the same as the Block's block size.

▶ Example

func NewOFB

```
func NewOFB(b Block, iv []byte) Stream
```

NewOFB returns a Stream that encrypts or decrypts using the block cipher b in output feedback mode. The initialization vector iv's length must be equal to b's block size.

▶ Example

type StreamReader

```
type StreamReader struct {
    S Stream
    R io.Reader
}
```

StreamReader wraps a Stream into an io.Reader. It calls XORKeyStream to process each slice of data which passes through.

▶ Example

func (StreamReader) Read

```
func (r StreamReader) Read(dst []byte) (n int, err error)
```

type StreamWriter

```
type StreamWriter struct {
    S   Stream
    W   io.Writer
    Err error // unused
}
```

StreamWriter wraps a Stream into an io.Writer. It calls XORKeyStream to process each slice of data which passes through. If any Write call returns short then the StreamWriter is out of sync and must be discarded. A StreamWriter has no internal buffering; Close does not need to be called to flush write data.

▶ Example

func (StreamWriter) Close

```
func (w StreamWriter) Close() error
```

Close closes the underlying Writer and returns its Close return value, if the Writer is also an io.Closer. Otherwise it returns nil.

func (StreamWriter) Write

```
func (w StreamWriter) Write(src []byte) (n int, err error)
```

Source Files

View all

cbc.go	ctr.go	ofb.go
cfb.go	gcm.go	
cipher.go	io.go	

Google