Go: cifrare e decifrare una stringa

La cifratura e la decifratura di stringhe sono operazioni cruciali per garantire la sicurezza delle informazioni sensibili. Go è un linguaggio di programmazione che offre una libreria standard molto potente per gestire la cifratura e la decifratura di stringhe in modo sicuro. In questo articolo, esploreremo come eseguire queste operazioni utilizzando Go.

Per iniziare a cifrare e decifrare una stringa in Go, dobbiamo importare il pacchetto `crypto` che fornisce numerose funzioni e algoritmi di crittografia. Assicuriamoci di importare il pacchetto necessario nel nostro programma:

```
import (
    "crypto/aes"
    "crypto/cipher"
    "crypto/rand"
    "errors"
    "io"
)
```

Per cifrare una stringa in Go, utilizzeremo l'algoritmo di cifratura AES (Advanced Encryption Standard) con una chiave segreta. Ecco come eseguire la cifratura di una stringa:

```
func EncryptString(key []byte, plaintext string)
(string, error) {
    block, err := aes.NewCipher(key)
    if err != nil {
        return "", err
    }
```

```
    plaintextBytes := []byte(plaintext)
    ciphertext := make([]byte,
aes.BlockSize+len(plaintextBytes))

    iv := ciphertext[:aes.BlockSize]
    if _, err := io.ReadFull(rand.Reader, iv); err !=
nil {
        return "", err
    }

    stream := cipher.NewCFBEncrypter(block, iv)
    stream.XORKeyStream(ciphertext[aes.BlockSize:],
plaintextBytes)

    return string(ciphertext), nil
}
```

Nel codice sopra, generiamo un vettore di inizializzazione (IV) casuale per migliorare la sicurezza. L'IV viene poi utilizzato per inizializzare il cifrario AES. La stringa cifrata risultante viene restituita come una stringa base64 per la facilità di memorizzazione e trasmissione.

Per decifrare una stringa cifrata in Go, è necessario avere la chiave segreta utilizzata per la cifratura. Ecco come decifrare una stringa:

```
func DecryptString(key []byte, ciphertext string)
(string, error) {
    block, err := aes.NewCipher(key)
    if err != nil {
        return "", err
    }
```

```
    ciphertextBytes := []byte(ciphertext)
    if len(ciphertextBytes) < aes.BlockSize {
        return "", errors.New("ciphertext is too
short")
    }

    iv := ciphertextBytes[:aes.BlockSize]
    ciphertextBytes = ciphertextBytes[aes.BlockSize:]

    stream := cipher.NewCFBDecrypter(block, iv)
    stream.XORKeyStream(ciphertextBytes,
ciphertextBytes)

    return string(ciphertextBytes), nil
}
```

In questo caso, forniamo la chiave segreta utilizzata per la cifratura e la stringa cifrata. Dopo aver estratto l'IV, decifriamo la stringa cifrata utilizzando lo stesso algoritmo AES con il metodo `cipher.NewCFBDecrypter`.

Ecco un esempio completo di come cifrare e decifrare una stringa in Go:

```
package main

import (
    "fmt"
    "crypto/aes"
    "crypto/cipher"
    "crypto/rand"
    "encoding/base64"
    "errors"
    "io"
```

```
)

func EncryptString(key []byte, plaintext string)
(string, error) {
    block, err := aes.NewCipher(key)
    if err != nil {
        return "", err
    }

    plaintextBytes := []byte(plaintext)
    ciphertext := make([]byte,
aes.BlockSize+len(plaintextBytes))

    iv := ciphertext[:aes.BlockSize]
    if _, err := io.ReadFull(rand.Reader, iv); err !=
nil {
        return "", err
    }

    stream := cipher.NewCFBEncrypter(block, iv)
    stream.XORKeyStream(ciphertext[aes.BlockSize:],
plaintextBytes)

    return
base64.URLEncoding.EncodeToString(ciphertext), nil
}

func DecryptString(key []byte, ciphertext string)
(string, error) {
    block, err := aes.NewCipher(key)
    if err != nil {
        return "", err
    }
```

```
    ciphertextBytes, err :=
base64.URLEncoding.DecodeString(ciphertext)
    if err != nil {
        return "", err
    }

    if len(ciphertextBytes) < aes.BlockSize {
        return "", errors.New("ciphertext is too
short")
    }

    iv := ciphertextBytes[:aes.BlockSize]
    ciphertextBytes = ciphertextBytes[aes.BlockSize:]

    stream := cipher.NewCFBDecrypter(block, iv)
    stream.XORKeyStream(ciphertextBytes,
ciphertextBytes)

    return string(ciphertextBytes), nil
}

func main() {
    key := []byte("thisisakey123456") // Chiave
segreta di 16, 24 o 32 byte
    plaintext := "Hello, world!"

    encrypted, err := EncryptString(key, plaintext)
    if err != nil {
        fmt.Println("Errore durante la cifratura:",
err)
        return
    }

    fmt.Println("Testo cifrato:", encrypted)
```

```
    decrypted, err := DecryptString(key, encrypted)
    if err != nil {
        fmt.Println("Errore durante la decifratura:",
err)
        return
    }

    fmt.Println("Testo decifrato:", decrypted)
}
```

In questo esempio, definiamo una chiave segreta di 16 byte, cifriamo una stringa di testo e poi la decifriamo con successo utilizzando le funzioni `EncryptString` e `DecryptString`. Assicurati di sostituire la chiave con una chiave segreta forte per scopi di sicurezza reali.

La cifratura e la decifratura di stringhe in Go possono sembrare complesse, ma utilizzando la libreria standard `crypto`, è possibile gestire queste operazioni in modo sicuro ed efficiente. La crittografia è una parte fondamentale della sicurezza delle informazioni, e Go offre le risorse necessarie per implementarla in modo efficace nei tuoi progetti.