IV-initialization vector

5 under Initialization vector (IV) A vector used in defining the starting point of a cryptographic process (e.g. encryption and key wrapping).

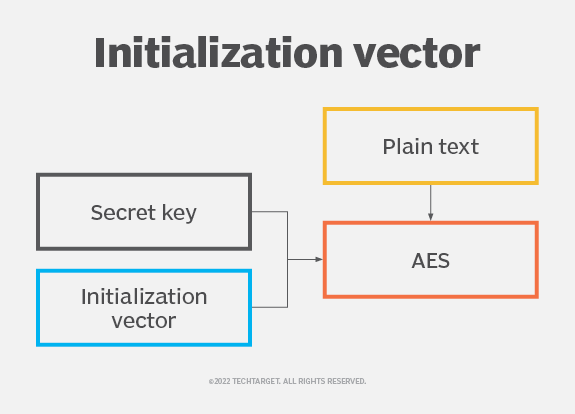
What is an initialization vector (IV)?

An initialization vector (IV) is an arbitrary number that can be used with a secret key for data encryption to foil cyber attacks. This number, also called a nonce (number used once), is employed only one time in any session to prevent unauthorized decryption of the message by a suspicious or malicious actor.

Importance of an initialization vector

The use of an IV prevents the repetition of a sequence of text in data encryption. Specifically, during encryption, an IV prevents a sequence of plaintext that’s identical to a previous plaintext sequence from producing the same ciphertext. If an attacker can view the same encrypted data multiple times, they get clues to decrypt and interpret the original values. That’s why encrypted ciphertext data is vulnerable to theft or compromise.

An IV is meant to prevent this from happening. A random unique nonce removes the need for repetition during encryption. The hacker cannot view the same encrypted information over and over, which makes it more difficult for them to decrypt the message via, say, a dictionary attack.



Properties of an ideal initialization vector

In a cryptographic algorithm, an IV is used as a “starting state”. Adding the IV to the cipher hides patterns in the encrypted data that may allow a hacker to decrypt it by guesswork or trial and error.

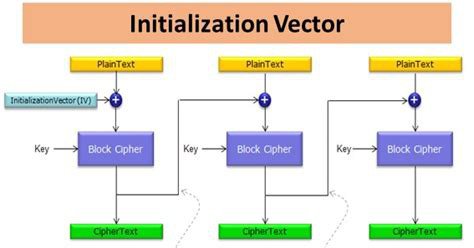
The ideal IV is a random or pseudorandom number. It must also be nonrepeating. Both randomness and nonrepetitiveness are crucial to prevent attackers from finding patterns in similar part of the encrypted message and then using this information to decrypt the message.

The IV need not be secret. In fact, the destination computer usually knows the IV so it can decrypt the encrypted data when it reveives it, Thus, the IV would be agreed on in advance by both the sender and the receipt. In addition, the IV can be transmitted independently or included as part of the session setup prior to message exchange.

Ways to make the IV available to a recipient

For a recipient to decrypt the encrypt the encrypted message, they must know the IV. There are many ways to make the IV available to the recipient to facilitate decryption. One way is to transmit it along with the ciphertext. Another way is for both the sender and recipient to agree on the IV during the key handshake or exchange.

The recipient can also discover the IV by calculating it incrementally or by measuring parameters like current time, using the sender’s or recipient’s address, or by using the packet or cluster number.



AES

The **Advanced Encryption Standard (AES)** is a specification for the encryption of electronic data established by the U.S National Institute of Standards and Technology (NIST) in 2001. AES is widely used today as it is a much stronger that DES and triple DES despite being harder to implement.

Using on Golang

In addition, all of source of code is on github.com. File: Web-Authentication-Encryption-JWT\Research-tutorial\aes-explanation.go

Required imports.

We will need the crypto/aes package for it to work.

import (

    "crypto/aes"

    "encoding/hex"

)

Encrypting a message using AES

    // cipher key

    key := "thisis32bitlongpassphraseimusing"

    // plaintext

    pt := "This is a secret"

    c := EncryptAES([]byte(key), pt)

Here the EncrypAES function is as follows:

func EncryptAES(key []byte, plaintext string) string {

    c, err := aes.NewCipher(key)

    CheckError(err)

    out := make([]byte, len(plaintext))

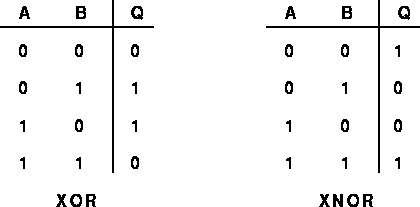
    c.Encrypt(out, []byte(plaintext))

    return hex.EncodeToString(out)

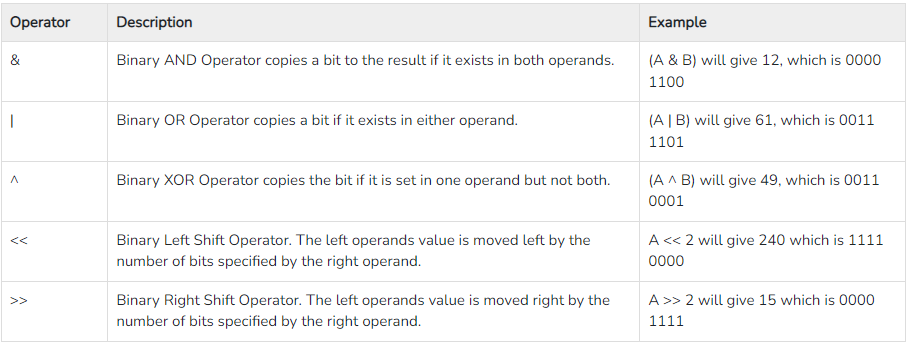
}

XOR bitwise operation

XOR gate (sometimes EOR, or EXOR and pronounced as Exclusive OR) is a digital logic gate that gives a true (1 or HIGH) output when the number of true inputs is odd. An XOR gate implements an exclusive or ( ) from mathematical logic; that is, a true output results if one, and only one, of the inputs to the gate is true.



The Bitwise operators supported by Go language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then –



Try the following example to understand all the bitwise operators available in Go programming language-

package main

import "fmt"

func main() {

   var a uint = 60  /\* 60 = 0011 1100 \*/

   var b uint = 13  /\* 13 = 0000 1101 \*/

   var c uint = 0

   c = a & b       /\* 12 = 0000 1100 \*/ AND

   fmt.Printf("Line 1 - Value of c is %d\n", c )

   c = a | b       /\* 61 = 0011 1101 \*/ OR

   fmt.Printf("Line 2 - Value of c is %d\n", c )

   c = a ^ b       /\* 49 = 0011 0001 \*/ XOR

   fmt.Printf("Line 3 - Value of c is %d\n", c )

   c = a << 2     /\* 240 = 1111 0000 \*/

   fmt.Printf("Line 4 - Value of c is %d\n", c )

   c = a >> 2     /\* 15 = 0000 1111 \*/

   fmt.Printf("Line 5 - Value of c is %d\n", c )

}

When you compile and execute the above program it produces the following result—

Line 1 - Value of c is 12

Line 2 - Value of c is 61

Line 3 - Value of c is 49

Line 4 - Value of c is 240

Line 5 - Value of c is 15

Binary code:

