**ЛАБОРАТОРНА РОБОТА № 3**

В даній роботі потрібно реалізувати дві криптосистеми, одна з яких використовує AES в режимі CBC (зчеплення шифрованих блоків), а інша AES в режимі з лічильником (CTR). В обох випадках 16-байтний початковий вектор IV обирається випадково і приписується спереду шифротексту. Для режиму CBC використана схема PKCS5 для доповнення повідомлення, щоб його довжина стала кратною довжині блоку.

Нижче наведені ключі та зашифровані повідомлення в шістнадцятковому вигляді для різних режимів шифрування. Задача полягає у дешифруванні обох повідомлень.

Для реалізації AES слід використати існуючі криптографічні бібліотеки такі як PyCrypto (Python), Crypto++ (C++) або будь-які інші. Режими CBC та CTR слід реалізувати самостійно.

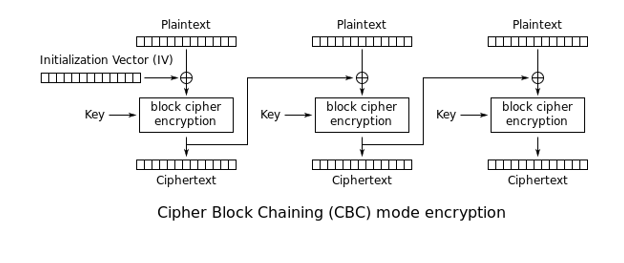


Рисунок 1 – Схема шифрування за допомогою CBC

Дані для режиму CBC

• CBC ключ: 140b41b22a29beb4061bda66b6747e14

• CBC шифротекст:

4ca00ff4c898d61e1edbf1800618fb2828a226d160dad07883d04e008a7897ee\

2e4b7465d5290d0c0e6c6822236e1daafb94ffe0c5da05d9476be028ad7c1d81

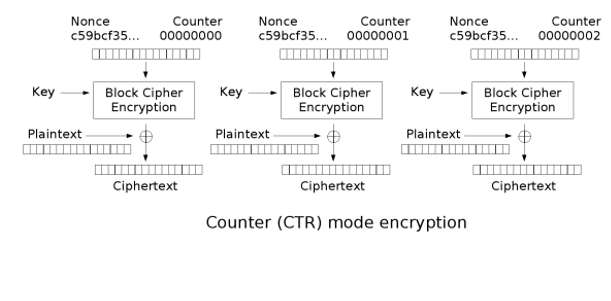


Рисунок 2 – Схема шифрування за допомогою лічильника CTR

Дані для режиму CTR

• CTR ключ: 36f18357be4dbd77f050515c73fcf9f2

• CTR шифротекст:

69dda8455c7dd4254bf353b773304eec0ec7702330098ce7f7520d1cbbb20fc3\

88d1b0adb5054dbd7370849dbf0b88d393f252e764f1f5f7ad97ef79d59ce29f5f51eeca32e\abedd9afa9329

Дана лабораторна робота була реалізована на мові програмування Go та стандартними криптографічними утилітами.

**Вихідний код**

package main

import (

//b64 "encoding/base64"

hex "encoding/hex"

"fmt"

"log"

"crypto/aes"

"strconv"

"os"

"bufio"

)

func DecodeHex(key string) string {

decoded, err := hex.DecodeString(key)

if err != nil {

log.Fatal(err)

}

return string(decoded)

}

func GetCountCycles(text string) rune {

return rune((len(text) / 32) - 1)

}

func DecryptAes128Ecb(data, key []byte) []byte {

cipher, \_ := aes.NewCipher([]byte(key))

decrypted := make([]byte, len(data))

size := 16

for bs, be := 0, size; bs < len(data); bs, be = bs+size, be+size {

cipher.Decrypt(decrypted[bs:be], data[bs:be])

}

return decrypted

}

func EncryptAes128Ecb(data, key []byte) []byte {

cipher, \_ := aes.NewCipher([]byte(key))

decrypted := make([]byte, len(data))

size := 16

for bs, be := 0, size; bs < len(data); bs, be = bs+size, be+size {

cipher.Encrypt(decrypted[bs:be], data[bs:be])

}

return decrypted

}

func DecryptBlock(block []byte, str string) string {

var block\_result string = ""

for j, i := range block {

t := i ^ str[j]

j += 1

if t < 16 {

block\_result += "0"

}

block\_result += strconv.FormatInt(int64(t), 16)

}

return block\_result

}

func SolveCBCEncryption(key string, text string) string {

var cbc string = ""

key1 := DecodeHex(key)

cycles := GetCountCycles(text)

for cycle := range make([]rune, cycles){

s := DecodeHex(text[(cycle + 1) \* 32:(cycle + 2) \* 32])

block := DecryptAes128Ecb([]byte(s), []byte(key1))

str2 := DecodeHex(text[cycle \* 32:((cycle + 1) \* 32)])

str3 := DecryptBlock(block, str2)

cbc += str3

}

return cbc

}

func GetParts(text string) (byte, string) {

ciper := DecodeHex(text[:32])

last := len(ciper) - 1

front := ciper[last]

back := ciper[:last]

return front, back

}

func SolveCTRModeEncryption(key string, text string) string {

var ctr string = ""

front, back := GetParts(text)

for c := range make([]int, int((len(text) / 64) - 1)) {

encrypt := append([]byte(back), front + byte(c))

message\_block := EncryptAes128Ecb(encrypt, []byte(DecodeHex(key)))

key\_block := DecodeHex(text[(c + 1) \* 32:(c + 1) \* 64])

ctr += DecryptBlock(message\_block, key\_block)

}

return DecodeHex(ctr)

}

func StartCBC() {

var key string = "140b41b22a29beb4061bda66b6747e14"

var text string ="4ca00ff4c898d61e1edbf1800618fb2828a226d160dad07883d04e008a7897ee2e4b7465d5290d0c0e6c6822236e1daafb94ffe0c5da05d9476be028ad7c1d81"

var cbc string = SolveCBCEncryption(key, text)

PrintResult(cbc, "./cbc.txt")

}

func StartCTR() {

var key string = "36f18357be4dbd77f050515c73fcf9f2"

var text string = "69dda8455c7dd4254bf353b773304eec0ec7702330098ce7f7520d1cbbb20fc388d1b0adb5054dbd7370849dbf0b88d393f252e764f1f5f7ad97ef79d59ce29f5f51eeca32eabedd9afa932908080808080808080808080808080808080808080808080808080808080808008080808080808080808080808080808080808080808080808080808080808080808080808080808080808080808080808080808080808080808080808080808080808080808080"

var ctr string = SolveCTRModeEncryption(key, text)

PrintResultsToFile(ctr, "./ctr.txt")

}

func main() {

StartCBC()

StartCTR()

}

func PrintResult( b string, file string) {

var response string = ""

for i := 0; i < len(b); i+=2{

hexCode := string(b[i]) + string(b[i+1])

number, \_ := strconv.ParseInt(hexCode, 16, 8)

response += fmt.Sprintf("%s", string(number))

}

PrintResultsToFile(response, file)

}

func check(e error) {

if e != nil {

panic(e)

}

}

func PrintResultsToFile(response string, file string) {

f, e := os.Create(file)

check(e)

defer f.Close()

w := bufio.NewWriter(f)

w.WriteString(response)

w.Flush()

fmt.Println("Results printed to file")

}

**Результат виконання**

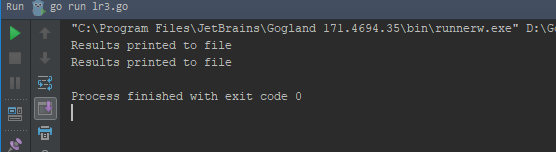


Рисунок 3 – Результат виконання

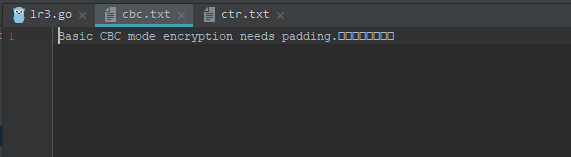


Рисунок 4 – Результат виконання CBC

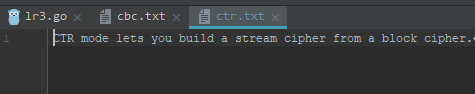


Рисунок 5 – Результат виконання CTR

**ВИСНОВОК**

В даній лабораторній роботі було розглянуто підходи до блочних шифрів завдяки методу зчеплення блоків CBC та методу з лічильником CTR. Звісно, що дані методи треба використовувати під певну ціль, але розглянувши їх на прикладі лабораторної роботи стало зрозуміло, що метод з лічильноком є більш приємним завдяки його можливості розпаралелення та незалежності до доповнення блоків як CBC.