Міністерство освіти і науки України

Національний технічний університет «ХПІ»

Навчально-науковий інститут комп’ютерних наук та інформаційних технологій

Кафедра комп’ютерної інженерії та програмування

**ЗВІТ**

з лабораторної роботи

з дисципліни «СУЧАСНІ ТЕХНОЛОГІЇ БЕЗПЕЧНОГО ПРОГРАМУВАННЯ»

**«Використання** **мнемонічних фраз для формування ключів шифрування»**

Виконав: студент групи КН-М922б

\_Кліщов Б. Р.\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ініціали)

Перевірив: доцент кафедри

\_Бульба С. С \_\_\_\_\_\_\_\_\_\_

(прізвище, ініціали)

Харків - 2022

**Мета:** Дослідити принципи роботи шифрування за допомогою мнемонічних фраз для формування ключів шифрування на прикладі алгоритму bip39.

**Завдання:**

* Використовуючи алгоритм bip39, створити seed генератора псевдовипадкових чисел за допомогою мнемонічної фрази та стосовні ключі шифрування.
* Зашифрувати текст
* Використовуючи раніше створену мнемонічну фразу, відновити ключі шифрування на дешифрувати текст. Вдосконалитись, що оригінальний та дешифрований тексти однакові.

**Хід роботи**

**Ключові елементи програми:**

func NewEntropy(bitSize int) ([]byte, error) {

if err := validateEntropyBitSize(bitSize); err != nil {

return nil, err

}

entropy := make([]byte, bitSize/8)

\_, \_ = rand.Read(entropy) // err is always nil

return entropy, nil

}

func EntropyFromMnemonic(mnemonic string) ([]byte, error) {

mnemonicSlice, isValid := splitMnemonicWords(mnemonic)

if !isValid {

return nil, ErrInvalidMnemonic

}

var (

wordBytes [2]byte

b = big.NewInt(0)

)

for \_, v := range mnemonicSlice {

index, found := wordMap[v]

if !found {

return nil, fmt.Errorf("word `%v` not found in reverse map", v)

}

binary.BigEndian.PutUint16(wordBytes[:], uint16(index))

b.Mul(b, shift11BitsMask)

b.Or(b, big.NewInt(0).SetBytes(wordBytes[:]))

}

// Build and add the checksum to the big.Int.

checksum := big.NewInt(0)

checksumMask := wordLengthChecksumMasksMapping[len(mnemonicSlice)]

checksum = checksum.And(b, checksumMask)

b.Div(b, big.NewInt(0).Add(checksumMask, bigOne))

entropy := b.Bytes()

entropy = padByteSlice(entropy, len(mnemonicSlice)/3\*4)

entropyChecksumBytes := computeChecksum(entropy)

entropyChecksum := big.NewInt(int64(entropyChecksumBytes[0]))

if l := len(mnemonicSlice); l != 24 {

checksumShift := wordLengthChecksumShiftMapping[l]

entropyChecksum.Div(entropyChecksum, checksumShift)

}

if checksum.Cmp(entropyChecksum) != 0 {

return nil, ErrChecksumIncorrect

}

return entropy, nil

}

func NewMnemonic(entropy []byte) (string, error) {

entropyBitLength := len(entropy) \* 8

checksumBitLength := entropyBitLength / 32

sentenceLength := (entropyBitLength + checksumBitLength) / 11

err := validateEntropyBitSize(entropyBitLength)

if err != nil {

return "", err

}

entropy = addChecksum(entropy)

entropyInt := new(big.Int).SetBytes(entropy)

words := make([]string, sentenceLength)

word := big.NewInt(0)

for i := sentenceLength - 1; i >= 0; i-- {

word.And(entropyInt, last11BitsMask)

entropyInt.Div(entropyInt, shift11BitsMask)

wordBytes := padByteSlice(word.Bytes(), 2)

words[i] = wordList[binary.BigEndian.Uint16(wordBytes)]

}

return strings.Join(words, " "), nil

}

func MnemonicToByteArray(mnemonic string, raw ...bool) ([]byte, error) {

var (

mnemonicSlice = strings.Split(mnemonic, " ")

entropyBitSize = len(mnemonicSlice) \* 11

checksumBitSize = entropyBitSize % 32

fullByteSize = (entropyBitSize-checksumBitSize)/8 + 1

)

rawEntropyBytes, err := EntropyFromMnemonic(mnemonic)

if err != nil {

return nil, err

}

if len(raw) > 0 && raw[0] {

return rawEntropyBytes, nil

}

return padByteSlice(addChecksum(rawEntropyBytes), fullByteSize), nil

}

func NewSeedWithErrorChecking(mnemonic string, password string) ([]byte, error) {

\_, err := MnemonicToByteArray(mnemonic)

if err != nil {

return nil, err

}

return NewSeed(mnemonic, password), nil

}

**Фрагмент основної програми:**

func main() {

entropy, \_ := bip39.NewEntropy(256)

mnemonic, \_ := bip39.NewMnemonic(entropy)

seed := bip39.NewSeed(mnemonic, "Secret Passphrase")

masterKey, \_ := bip32.NewMasterKey(seed)

publicKey := masterKey.PublicKey()

fmt.Println("Mnemonic: ", mnemonic)

fmt.Println("Master private key: ", masterKey)

fmt.Println("Master public key: ", publicKey)

departmentKeys := map[string]\*bip32.Key{}

departmentKeys["Sales"], \_ = masterKey.NewChildKey(0)

departmentKeys["Marketing"], \_ = masterKey.NewChildKey(1)

departmentKeys["Engineering"], \_ = masterKey.NewChildKey(2)

departmentKeys["Customer Support"], \_ = masterKey.NewChildKey(3)

departmentAuditKeys := map[string]\*bip32.Key{}

departmentAuditKeys["Sales"] = departmentKeys["Sales"].PublicKey()

departmentAuditKeys["Marketing"] = departmentKeys["Marketing"].PublicKey()

departmentAuditKeys["Engineering"] = departmentKeys["Engineering"].PublicKey()

departmentAuditKeys["Customer Support"] = departmentKeys["Customer Support"].PublicKey()

for department, pubKey := range departmentAuditKeys {

fmt.Println(department, pubKey)

}

}

**Результати роботи програми**

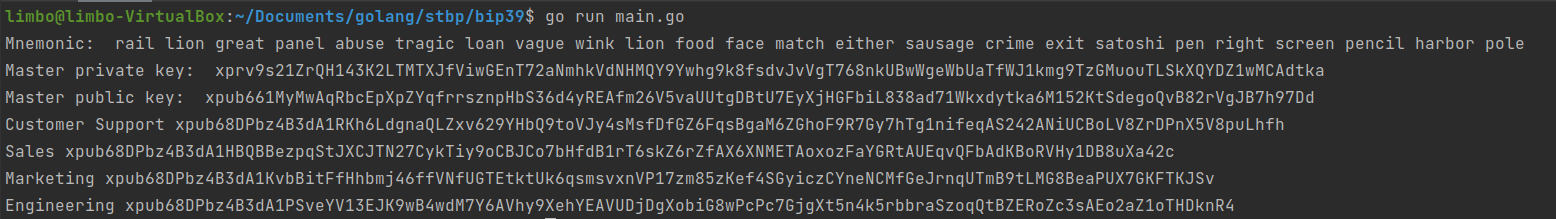
****

Рисунок 1 – Результат роботи програми (генерація ключів)

**Висновки:** досліджено принципи роботи шифрування за допомогою мнемонічних фраз для формування ключів шифрування на прикладі алгоритму bip39. Розроблено програму роботи даного алгоритму.