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»

**«Асиметричне шифрування. Алгоритм RSA»**

Виконав: студент групи КН-М922б

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

(прізвище, ініціали)

Перевірив: доцент кафедри

\_Бульба С. С \_\_\_\_\_\_\_\_\_\_

(прізвище, ініціали)

Харків - 2022

**Мета:** Дослідити і реалізувати механізм асиметричного алгоритму шифрування RSA.

**Завдання:**

Розробити додаток обміну таємними посиланнями між двома клієнтами за допомогою алгоритму шифрування RSA

* Реалізувати алгоритм генерації ключів (public / private keys) для алгоритму RSA. Створити ключі заданої довжини (напр. 1024 біт)
* Реалізувати та продемонструвати роботу алгоритму шифрування та дешифрування повідомлення RSA
* Підтвердити роботу реалізованого алгоритму шляхом порівняння результату кодування з існуючим алгоритмом (наприклад, використовуючи утиліту openssl або вбудовані системи шифрування обраної мови програмування)

**Хід роботи**

**Ключові елементи програми:**

func (rsas \*RSASecurity) GetPrivatekey() (\*rsa.PrivateKey, error) {

return getPriKey([]byte(rsas.priStr))

}

func (rsas \*RSASecurity) GetPublickey() (\*rsa.PublicKey, error) {

return getPubKey([]byte(rsas.pubStr))

}

func (rsas \*RSASecurity) PubKeyENCTYPT(input []byte) ([]byte, error) {

if rsas.pubkey == nil {

return []byte(""), errors.New(`Please set the public key in advance`)

}

output := bytes.NewBuffer(nil)

err := pubKeyIO(rsas.pubkey, bytes.NewReader(input), output, true)

if err != nil {

return []byte(""), err

}

return ioutil.ReadAll(output)

}

func (rsas \*RSASecurity) PubKeyDECRYPT(input []byte) ([]byte, error) {

if rsas.pubkey == nil {

return []byte(""), errors.New(`Please set the public key in advance`)

}

output := bytes.NewBuffer(nil)

err := pubKeyIO(rsas.pubkey, bytes.NewReader(input), output, false)

if err != nil {

return []byte(""), err

}

return ioutil.ReadAll(output)

}

func (rsas \*RSASecurity) PriKeyENCTYPT(input []byte) ([]byte, error) {

if rsas.prikey == nil {

return []byte(""), errors.New(`Please set the private key in advance`)

}

output := bytes.NewBuffer(nil)

err := priKeyIO(rsas.prikey, bytes.NewReader(input), output, true)

if err != nil {

return []byte(""), err

}

return ioutil.ReadAll(output)

}

func (rsas \*RSASecurity) PriKeyDECRYPT(input []byte) ([]byte, error) {

if rsas.prikey == nil {

return []byte(""), errors.New(`Please set the private key in advance`)

}

output := bytes.NewBuffer(nil)

err := priKeyIO(rsas.prikey, bytes.NewReader(input), output, false)

if err != nil {

return []byte(""), err

}

return ioutil.ReadAll(output)

}

func PublicEncrypt(data, publicKey string) (string, error) {

grsa := RSASecurity{}

grsa.SetPublicKey(publicKey)

rsadata, err := grsa.PubKeyENCTYPT([]byte(data))

if err != nil {

return "", err

}

return base64.StdEncoding.EncodeToString(rsadata), nil

}

func PriKeyEncrypt(data, privateKey string) (string, error) {

grsa := RSASecurity{}

grsa.SetPrivateKey(privateKey)

rsadata, err := grsa.PriKeyENCTYPT([]byte(data))

if err != nil {

return "", err

}

return base64.StdEncoding.EncodeToString(rsadata), nil

}

func PublicDecrypt(data, publicKey string) (string, error) {

databs, \_ := base64.StdEncoding.DecodeString(data)

grsa := RSASecurity{}

if err := grsa.SetPublicKey(publicKey); err != nil {

return "", err

}

rsadata, err := grsa.PubKeyDECRYPT(databs)

if err != nil {

return "", err

}

return string(rsadata), nil

}

func PriKeyDecrypt(data, privateKey string) (string, error) {

databs, \_ := base64.StdEncoding.DecodeString(data)

grsa := RSASecurity{}

if err := grsa.SetPrivateKey(privateKey); err != nil {

return "", err

}

rsadata, err := grsa.PriKeyDECRYPT(databs)

if err != nil {

return "", err

}

return string(rsadata), nil

}

**Фрагмент основної програми:**

var PubKey = `-----BEGIN Public key-----

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAk+89V7vpOj1rG6bTAKYM

56qmFLwNCBVDJ3MltVVtxVUUByqc5b6u909MmmrLBqS//PWC6zc3wZzU1+ayh8xb

UAEZuA3EjlPHIaFIVIz04RaW10+1xnby/RQE23tDqsv9a2jv/axjE/27b62nzvCW

eItu1kNQ3MGdcuqKjke+LKhQ7nWPRCOd/ffVqSuRvG0YfUEkOz/6UpsPr6vrI331

hWRB4DlYy8qFUmDsyvvExe4NjZWblXCqkEXRRAhi2SQRCl3teGuIHtDUxCskRIDi

aMD+Qt2Yp+Vvbz6hUiqIWSIH1BoHJer/JOq2/O6X3cmuppU4AdVNgy8Bq236iXvr

MQIDAQAB

-----END Public key-----

`

var PriKey = `-----BEGIN Private key-----

MIIEpAIBAAKCAQEAk+89V7vpOj1rG6bTAKYM56qmFLwNCBVDJ3MltVVtxVUUByqc

5b6u909MmmrLBqS//PWC6zc3wZzU1+ayh8xbUAEZuA3EjlPHIaFIVIz04RaW10+1

xnby/RQE23tDqsv9a2jv/axjE/27b62nzvCWeItu1kNQ3MGdcuqKjke+LKhQ7nWP

RCOd/ffVqSuRvG0YfUEkOz/6UpsPr6vrI331hWRB4DlYy8qFUmDsyvvExe4NjZWb

lXCqkEXRRAhi2SQRCl3teGuIHtDUxCskRIDiaMD+Qt2Yp+Vvbz6hUiqIWSIH1BoH

Jer/JOq2/O6X3cmuppU4AdVNgy8Bq236iXvrMQIDAQABAoIBAQCCbxZvHMfvCeg+

YUD5+W63dMcq0QPMdLLZPbWpxMEclH8sMm5UQ2SRueGY5UBNg0WkC/R64BzRIS6p

jkcrZQu95rp+heUgeM3C4SmdIwtmyzwEa8uiSY7Fhbkiq/Rly6aN5eB0kmJpZfa1

6S9kTszdTFNVp9TMUAo7IIE6IheT1x0WcX7aOWVqp9MDXBHV5T0Tvt8vFrPTldFg

IuK45t3tr83tDcx53uC8cL5Ui8leWQjPh4BgdhJ3/MGTDWg+LW2vlAb4x+aLcDJM

CH6Rcb1b8hs9iLTDkdVw9KirYQH5mbACXZyDEaqj1I2KamJIU2qDuTnKxNoc96HY

2XMuSndhAoGBAMPwJuPuZqioJfNyS99x++ZTcVVwGRAbEvTvh6jPSGA0k3cYKgWR

NnssMkHBzZa0p3/NmSwWc7LiL8whEFUDAp2ntvfPVJ19Xvm71gNUyCQ/hojqIAXy

tsNT1gBUTCMtFZmAkUsjqdM/hUnJMM9zH+w4lt5QM2y/YkCThoI65BVbAoGBAMFI

GsIbnJDNhVap7HfWcYmGOlWgEEEchG6Uq6Lbai9T8c7xMSFc6DQiNMmQUAlgDaMV

b6izPK4KGQaXMFt5h7hekZgkbxCKBd9xsLM72bWhM/nd/HkZdHQqrNAPFhY6/S8C

IjRnRfdhsjBIA8K73yiUCsQlHAauGfPzdHET8ktjAoGAQdxeZi1DapuirhMUN9Zr

kr8nkE1uz0AafiRpmC+cp2Hk05pWvapTAtIXTo0jWu38g3QLcYtWdqGa6WWPxNOP

NIkkcmXJjmqO2yjtRg9gevazdSAlhXpRPpTWkSPEt+o2oXNa40PomK54UhYDhyeu

akuXQsD4mCw4jXZJN0suUZMCgYAgzpBcKjulCH19fFI69RdIdJQqPIUFyEViT7Hi

bsPTTLham+3u78oqLzQukmRDcx5ddCIDzIicMfKVf8whertivAqSfHytnf/pMW8A

vUPy5G3iF5/nHj76CNRUbHsfQtv+wqnzoyPpHZgVQeQBhcoXJSm+qV3cdGjLU6OM

HgqeaQKBgQCnmL5SX7GSAeB0rSNugPp2GezAQj0H4OCc8kNrHK8RUvXIU9B2zKA2

z/QUKFb1gIGcKxYr+LqQ25/+TGvINjuf6P3fVkHL0U8jOG0IqpPJXO3Vl9B8ewWL

cFQVB/nQfmaMa4ChK0QEUe+Mqi++MwgYbRHx1lIOXEfUJO+PXrMekw==

-----END Private key-----

`

const msg = `hello world`

func main() {

if err := applyPubEPriD(); err != nil {

log.Println(err)

}

if err := applyPriEPubD(); err != nil {

log.Println(err)

}

block, \_ := pem.Decode([]byte(PriKey))

priKey, err := x509.ParsePKCS1PrivateKey(block.Bytes)

if err != nil {

log.Fatal(err)

}

pubKey := priKey.PublicKey

ciphertext, err := rsapkg.EncryptOAEP(sha256.New(), rand.Reader, &pubKey, []byte(msg), nil)

if err != nil {

log.Fatal(err)

}

plaintext, err := rsapkg.DecryptOAEP(sha256.New(), rand.Reader, priKey, ciphertext, nil)

if err != nil {

log.Fatal(err)

}

fmt.Println("RSA standard package plaintext:", string(plaintext))

}

func applyPubEPriD() error {

pubenctypt, err := rsa.PublicEncrypt(msg, PubKey)

if err != nil {

return err

}

pridecrypt, err := rsa.PriKeyDecrypt(pubenctypt, PriKey)

if err != nil {

return err

}

if pridecrypt != msg {

return errors.New(`Decryption failed`)

}

fmt.Println("RSA custom package plaintext:", pridecrypt)

return nil

}

func applyPriEPubD() error {

prienctypt, err := rsa.PriKeyEncrypt(msg, PriKey)

if err != nil {

return err

}

pubdecrypt, err := rsa.PublicDecrypt(prienctypt, PubKey)

if err != nil {

return err

}

if pubdecrypt != msg {

return errors.New(`Decryption failed`)

}

return nil

}

**Результати роботи програми**

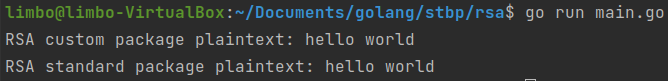
****

Рисунок 1 – Результат роботи програми

**Висновки:** досліджено і реалізувано механізм асиметричного алгоритму шифрування RSA. Розроблено програму роботи даного алгоритму.