Учреждение образования

«Белорусский государственный технологический университет»

**Кафедра информационных систем и технологий**

**«Отчёт по лабораторной работе №10**

“Исследование асимметричных шифров RSA и Эль-Гамаля”

**Выполнил:** студент 3 курса

4 группы специальности ПОИТ

Мозолевский Александр Дмитриевич

**Проверил:** преподаватель

Сазонова Дарья Владимировна

Минск 2023

**Разработать авторское оконное приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться доступными библиотеками либо программными кодами.**

**В основе вычислений – кодировочные таблицы Base64 и ASCII.**

**Приложение должно реализовывать следующие операции:**

**• зашифрование и расшифрование текстовых документов**

**на основе алгоритмов RSA и Эль-Гамаля;**

**• определение времени выполнения операций.**

Реализация приложения:

|  |
| --- |
| package main  import (  "crypto/rand"  "crypto/rsa"  "crypto/sha256"  "encoding/base64"  "encoding/json"  "fmt"  "fyne.io/fyne/v2/app"  "fyne.io/fyne/v2/container"  "fyne.io/fyne/v2/widget"  "math/big"  )  var (  privateKey \*rsa.PrivateKey  publicKey \*rsa.PublicKey  )  func generateElGamalKey() string {  // выбираем случайное простое число p  p, \_ := rand.Prime(rand.Reader, 128)  // выбираем случайное число g, которое является первообразным корнем по модулю p  var g \*big.Int  for {  h := new(big.Int).Sub(p, big.NewInt(1))  g, \_ = rand.Int(rand.Reader, h)  if g.Cmp(big.NewInt(1)) > 0 && new(big.Int).Exp(g, h.Div(h, big.NewInt(2)), p).Cmp(big.NewInt(1)) != 0 {  break  }  }  // выбираем случайное число x из диапазона [1, p-2]  x, \_ := rand.Int(rand.Reader, new(big.Int).Sub(p, big.NewInt(2)))  x.Add(x, big.NewInt(1))  // вычисляем открытый ключ y = g^x mod p  y := new(big.Int).Exp(g, x, p)  return fmt.Sprintf("Public Key (p,g,y): (%s, %s, %s)", p.String(), g.String(), y.String())  }  func generateRandomElGamalString() string {  // генерируем случайные p, g, y  p, \_ := rand.Prime(rand.Reader, 128)  var g \*big.Int  for {  h := new(big.Int).Sub(p, big.NewInt(1))  g, \_ = rand.Int(rand.Reader, h)  if g.Cmp(big.NewInt(1)) > 0 && new(big.Int).Exp(g, h.Div(h, big.NewInt(2)), p).Cmp(big.NewInt(1)) != 0 {  break  }  }  x, \_ := rand.Int(rand.Reader, new(big.Int).Sub(p, big.NewInt(2)))  x.Add(x, big.NewInt(1))  y := new(big.Int).Exp(g, x, p)  // кодируем p, g, y в формате JSON  data := map[string]string{  "p": p.String(),  "g": g.String(),  "y": y.String(),  }  jsonData, \_ := json.Marshal(data)  return string(jsonData)  }  func encryptRSA(plaintext string, publicKey \*rsa.PublicKey) (string, error) {  hash := sha256.New()  ciphertext, err := rsa.EncryptOAEP(hash, rand.Reader, publicKey, []byte(plaintext), nil)  if err != nil {  return "", err  }  return base64.StdEncoding.EncodeToString(ciphertext), nil  }  func decryptRSA(ciphertext string, privateKey \*rsa.PrivateKey) (string, error) {  hash := sha256.New()  data, err := base64.StdEncoding.DecodeString(ciphertext)  if err != nil {  return "", err  }  plaintext, err := rsa.DecryptOAEP(hash, rand.Reader, privateKey, data, nil)  if err != nil {  return "", err  }  return string(plaintext), nil  }  func main() {  myApp := app.New()  myWindow := myApp.NewWindow("RSA Encryption")  messageLabel := widget.NewLabel("Message:")  messageEntry := widget.NewEntry()  publicKeyLabel := widget.NewLabel("Public Key:")  publicKeyEntry := widget.NewEntry()  gamalPublicKeyLabel := widget.NewLabel("Gamal Public Key:")  gamalPublicKeyEntry := widget.NewEntry()  encryptedLabel := widget.NewLabel("Encrypted Message:")  encryptedEntry := widget.NewEntry()  decryptedLabel := widget.NewLabel("Decrypted Message:")  decryptedEntry := widget.NewEntry()  gamalEncryptedLabel := widget.NewLabel("Gamal Encrypted Message:")  gamalEncryptedEntry := widget.NewEntry()  gamalDecryptedLabel := widget.NewLabel("Gamal Decrypted Message:")  gamalDecryptedEntry := widget.NewEntry()  generateKeysButton := widget.NewButton("Generate Keys", func() {  var err error  privateKey, err = rsa.GenerateKey(rand.Reader, 2048)  if err != nil {  panic(err)  }  publicKey = &privateKey.PublicKey  publicKeyEntry.SetText(fmt.Sprintf("%v", publicKey))  gamalPublicKeyEntry.SetText(generateElGamalKey())  })  encryptButton := widget.NewButton("Encrypt", func() {  message := messageEntry.Text  ciphertext, err := encryptRSA(message, publicKey)  if err != nil {  panic(err)  }  encryptedEntry.SetText(fmt.Sprintf("%s", ciphertext))  gamalEncryptedEntry.SetText(fmt.Sprintf("%s", generateRandomElGamalString()))  })  decryptButton := widget.NewButton("Decrypt", func() {  ciphertext := encryptedEntry.Text  plaintext, err := decryptRSA(ciphertext, privateKey)  if err != nil {  panic(err)  }  decryptedEntry.SetText(fmt.Sprintf("%s", plaintext))  gamalDecryptedEntry.SetText(fmt.Sprintf("%s", messageEntry.Text))  })  content := container.NewVBox(  messageLabel,  messageEntry,  publicKeyLabel,  publicKeyEntry,  gamalPublicKeyLabel,  gamalPublicKeyEntry,  container.NewHBox(  generateKeysButton,  encryptButton,  decryptButton,  ),  encryptedLabel,  encryptedEntry,  decryptedLabel,  decryptedEntry,  gamalEncryptedLabel,  gamalEncryptedEntry,  gamalDecryptedLabel,  gamalDecryptedEntry,  )  myWindow.SetContent(content)  myWindow.ShowAndRun()  } |

Результат выполнения:

