Учреждение образования

«Белорусский государственный технологический университет»

**Кафедра информационных систем и технологий**

**«Отчёт по лабораторной работе №12**

“Исследование алгоритмов генерации и верификации электронной цифровой подписи”

**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 результатами выполнения предыдущих лабораторных работ, а также доступными библиотеками либо программными кодами.**

**Приложение должно реализовывать следующие операции:**

**• генерацию и верификацию ЭЦП на основе алгоритмов RSA, Эль-Гамаля и Шнорра;**

**• оценку времени выполнения указанных процедур при реальных (требуемых) ключевых параметрах**

Реализация приложения:

|  |
| --- |
| package main  import (  "crypto"  "crypto/ecdsa"  "crypto/elliptic"  "crypto/rand"  "crypto/rsa"  "crypto/sha256"  "fmt"  "math/big"  )  // Generates an RSA digital signature for the given message  func generateRSASignature(msg []byte, privateKey \*rsa.PrivateKey) ([]byte, error) {  hash := sha256.Sum256(msg)  signature, err := rsa.SignPKCS1v15(rand.Reader, privateKey, crypto.SHA256, hash[:])  if err != nil {  return nil, err  }  return signature, nil  }  // Verifies an RSA digital signature for the given message  func verifyRSASignature(msg []byte, signature []byte, publicKey \*rsa.PublicKey) (bool, error) {  hash := sha256.Sum256(msg)  err := rsa.VerifyPKCS1v15(publicKey, crypto.SHA256, hash[:], signature)  if err != nil {  return false, err  }  return true, nil  }  // Generates an ElGamal digital signature for the given message  func generateElGamalSignature(msg []byte, privateKey \*ecdsa.PrivateKey) ([]byte, []byte, error) {  hash := sha256.Sum256(msg)  r, s, err := ecdsa.Sign(rand.Reader, privateKey, hash[:])  if err != nil {  return nil, nil, err  }  rBytes, err := r.MarshalText()  if err != nil {  return nil, nil, err  }  sBytes, err := s.MarshalText()  if err != nil {  return nil, nil, err  }  return rBytes, sBytes, nil  }  // Verifies an ElGamal digital signature for the given message  func verifyElGamalSignature(msg []byte, rBytes []byte, sBytes []byte, publicKey \*ecdsa.PublicKey) (bool, error) {  hash := sha256.Sum256(msg)  var r, s big.Int  err := r.UnmarshalText(rBytes)  if err != nil {  return false, err  }  err = s.UnmarshalText(sBytes)  if err != nil {  return false, err  }  isValid := ecdsa.Verify(publicKey, hash[:], &r, &s)  return isValid, nil  }  // Generate a Schnorr digital signature for the given message  func generateSchnorrSignature(msg []byte, privateKey \*ecdsa.PrivateKey) ([]byte, []byte, error) {  hash := sha256.Sum256(msg)  r, s, err := ecdsa.Sign(rand.Reader, privateKey, hash[:])  if err != nil {  return nil, nil, err  }  rBytes, err := r.MarshalText()  if err != nil {  return nil, nil, err  }  sBytes, err := s.MarshalText()  if err != nil {  return nil, nil, err  }  return rBytes, sBytes, nil  }  // Verifies a Schnorr digital signature for the given message  func verifySchnorrSignature(msg []byte, rBytes []byte, sBytes []byte, publicKey \*ecdsa.PublicKey) (bool, error) {  hash := sha256.Sum256(msg)  var r, s big.Int  err := r.UnmarshalText(rBytes)  if err != nil {  return false, err  }  err = s.UnmarshalText(sBytes)  if err != nil {  return false, err  }  isValid := ecdsa.Verify(publicKey, hash[:], &r, &s)  return isValid, nil  }  func main() {  // Generate RSA keys  privateKey, err := rsa.GenerateKey(rand.Reader, 2048)  if err != nil {  panic(err)  }  publicKey := &privateKey.PublicKey  // Generate ElGamal keys  ecdsaPrivateKey, err := ecdsa.GenerateKey(elliptic.P256(), rand.Reader)  if err != nil {  panic(err)  }  // Generate Schnorr keys  secdsaPrivateKey, err := ecdsa.GenerateKey(elliptic.P256(), rand.Reader)  if err != nil {  panic(err)  }  // Generate message  msg := []byte("Hello, world!")  // Generate RSA signature and verify it  rsaSignature, err := generateRSASignature(msg, privateKey)  if err != nil {  panic(err)  }  isValid, err := verifyRSASignature(msg, rsaSignature, publicKey)  if err != nil {  panic(err)  }  fmt.Printf("RSA signature: %x\n", rsaSignature)  if isValid {  fmt.Println("RSA signature is valid")  } else {  fmt.Println("RSA signature is invalid")  }  // Generate ElGamal signature and verify it  rBytes, sBytes, err := generateElGamalSignature(msg, ecdsaPrivateKey)  if err != nil {  panic(err)  }  isValid, err = verifyElGamalSignature(msg, rBytes, sBytes, &ecdsaPrivateKey.PublicKey)  if err != nil {  panic(err)  }  if isValid {  fmt.Println("ElGamal signature is valid")  } else {  fmt.Println("ElGamal signature is invalid")  }  fmt.Printf("ElGamal signature: %x%x\n", rBytes, sBytes)  // Generate Schnorr signature and verify it  rBytes, sBytes, err = generateSchnorrSignature(msg, secdsaPrivateKey)  if err != nil {  panic(err)  }  isValid, err = verifySchnorrSignature(msg, rBytes, sBytes, &secdsaPrivateKey.PublicKey)  if err != nil {  panic(err)  }  if isValid {  fmt.Println("Schnorr signature is valid")  } else {  fmt.Println("Schnorr signature is invalid")  }  fmt.Printf("Schnorr signature: %x%x\n", rBytes, sBytes)  } |

Результат выполнения:

