Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 7
по дисциплине «Защита информации»

Тема Электронная подпись

Студент Пермякова Е. Д.

Группа ИУ7-72Б

Преподаватели Руденкова Ю. С.

Москва, 2025

ВВЕДЕНИЕ

Целью данной работы является реализация программы, которая создает и проверяет электронную подпись для документа.

Для достижения поставленной цели требуется решить следующие задачи:

1) описать работу электронной подписи;

2) реализовать программу, которая создает и проверяет электронную подпись для документа;

1 Теоретическая часть

Электронная цифровая подпись (ЭЦП) — это цифровой аналог обычной рукописной подписи, который используется для идентификации человека и подписания электронных документов или сведений.

Электронный сертификат — это электронный или бумажный документ, выпущенный удостоверяющим центром и подтверждающий принадлежность открытого ключа ЭЦП определенному лицу или организации Сертификат содержит:

— Открытый ключ ЭЦП;

— Сведения о владельце сертификата;

— Сведения об удостоверяющем центре;

— Срок действия сертификата;

— Цифровую подпись удостоверяющего центра.

Электронный сертификат связывает открытый ключ с идентификационными данными владельца и обеспечивает доверие к ЭЦП.

Классификация видов электронной подписи

Выделяют три основных типа электронной подписи, которые различаются степенью защищённости и юридической значимостью:

1) **Простая электронная подпись (ПЭП)**

— Предназначена для подтверждения авторства документа;

— Не обеспечивает гарантий неизменности содержимого после момента подписания;

— Требует заключения дополнительного соглашения между сторонами для признания юридической силы;

2) **Неквалифицированная электронная подпись (НЭП)**

— Обеспечивает контроль целостности документа;

— Создаётся с использованием криптографических средств защиты;

— Требует заключения дополнительного соглашения между сторонами для признания юридической силы;

3) **Квалифицированная электронная подпись (КЭП)**

— Подтверждается квалифицированным сертификатом аккредитованного удостоверяющего центра;

— Производится с использованием сертифицированных ФСБ России средств криптографической защиты;

— Приравнивается к собственноручной подписи без необходимости дополнительных соглашений;

генерация ключевой пары

2 Описание алгоритма создания и проверки электронной подписи

На рисунках 2.1-2.2 приведена схема создания и проверки электронной подписи.

Рисунок 2.1 – Схема создания электронной подписи

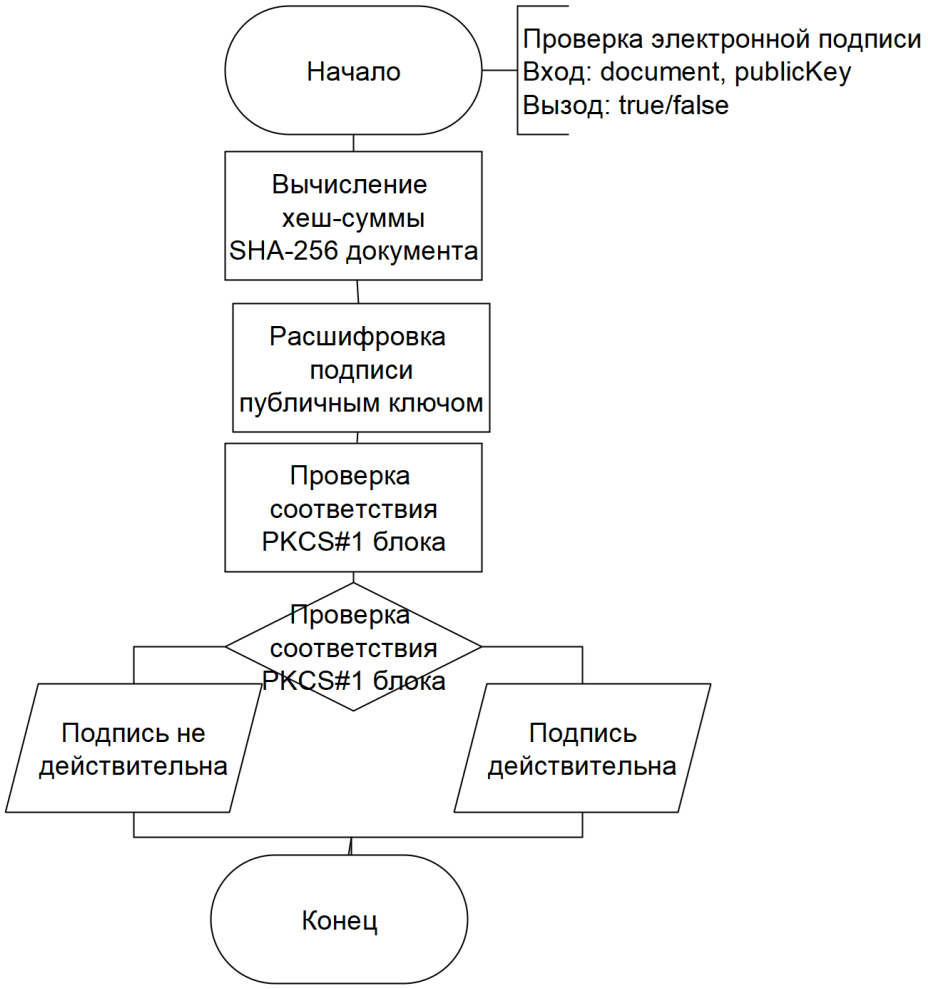

Рисунок 2.2 – Схема проверки электронной подписи

3 Пример создания и проверки электронной подписи

На рисунке 3.1 приведен пример создания и проверки электронной подписи.

```
kathrine@Viva:~/vuz/InfoSec$ cd is_sign/
kathrine@Viva:~/vuz/InfoSec/is_sign$ go run ./src/main.go
Использование:
  gen-keys [-priv private.pem] [-pub public.pem]
  sign <input_file> -priv <private_key> [-o signature.bin]
  verify <input_file> -pub <public_key> -s <signature_file>

Примеры:
  go run ./src/main.go gen-keys -priv private.pem -pub public.pem
  go run ./src/main.go sign ./data/text.txt -priv private.pem -o signature.bin
  go run ./src/main.go verify ./data/text.txt -pub public.pem -s signature.bin
exit status 1
kathrine@Viva:~/vuz/InfoSec/is_sign$ go run ./src/main.go gen-keys -priv private.pem -pub public.pem
Ключи сгенерированы:
 - Приватный: private.pem
 - Публичный: public.pem
kathrine@Viva:~/vuz/InfoSec/is_sign$ go run ./src/main.go sign ./data/text.txt -priv private.pem -o sig
nature.bin
Подпись сохранена в signature.bin
kathrine@Viva:~/vuz/InfoSec/is_sign$ go run ./src/main.go verify ./data/text.txt -pub public.pem -s sig
nature.bin
Подпись действительна.
```

Рисунок 3.1 – Пример создания и проверки электронной подписи

На рисунке 3.2 приведено содержимое исходного файла.

```
kathrine@Viva:~/vuz/InfoSec/is_sign$ cat ./data/text.txt
— Ну, здравствуйте, здравствуйте. Je vois que je vous fais peur 2, садитесь и рассказывайте.
Так говорила в июле 1805 года известная Анна Павловна Шерер, фрейлина и приближенная императрицы Марии
Феодоровны, встречая важного и чиновного князя Василия, первого приехавшего на ее вечер. Анна Павловна
кашляла несколько дней, у нее был грипп, как она говорила (грипп был тогда новое слово, употреблявшееся
 только редкими). В записочках, разосланных утром с красным лакеем, было написано без различия во всех:
«Si vous n'avez rien de mieux à faire, Monsieur le comte (или mon prince), et si la perspective de pass
er la soirée chez une pauvre malade ne vous effraye pas trop, je serai charmée de vous voir chez moi en
tre 7 et 10 heures. Annette Scherer» 3.
— Dieu, quelle virulente sortie! 4 — отвечал, нисколько не смутясь такою встречей, вошедший князь, в пр
идворном, шитом мундире, в чулках, башмаках и звездах, с светлым выражением плоского лица.
Он  говорил на том изысканном французском языке, на котором не только говорили, но и думали наши деды,
и с теми, тихими, покровительственными интонациями, которые свойственны состаревшемуся в свете и при дв
оре значительному человеку.
```

Рисунок 3.2 – Содержимое исходного файла

4 Реализация программы создания и проверки электронной подписи

В качестве средства реализации программы создания и проверки электронной подписи был выбран язык Go.

```
package main

import (
"crypto"
"crypto/rand"
"crypto/rsa"
"crypto/sha256"
"crypto/x509"
"encoding/pem"
"fmt"
"os"
)

func generateKeys(privateKeyPath, publicKeyPath string) error {
    privateKey, err := rsa.GenerateKey(rand.Reader, 2048)
    if err != nil {
        return fmt.Errorf("key generation error: %v", err)
    }

    privateKeyFile, err := os.Create(privateKeyPath)
    if err != nil {
        return fmt.Errorf("private key file creation error: %v"
            , err)
    }
    defer privateKeyFile.Close()

    privateKeyPEM := &pem.Block{
        Type:  "RSA PRIVATE KEY",
        Bytes: x509.MarshalPKCS1PrivateKey(privateKey),
    }
    if err := pem.Encode(privateKeyFile, privateKeyPEM); err !=
        nil {
        return fmt.Errorf("private key encoding error: %v", err
            )
    }
```

```
    publicKeyFile, err := os.Create(publicKeyPath)
    if err != nil {
        return fmt.Errorf("public key file creation error: %v",
            err)
    }
    defer publicKeyFile.Close()

    publicKeyBytes, err := x509.MarshalPKIXPublicKey(&
       privateKey.PublicKey)
    if err != nil {
        return fmt.Errorf("public key marshaling error: %v",
           err)
    }

    publicKeyPEM := &pem.Block{
        Type:  "RSA PUBLIC KEY",
        Bytes: publicKeyBytes,
    }
    if err := pem.Encode(publicKeyFile, publicKeyPEM); err !=
       nil {
        return fmt.Errorf("public key encoding error: %v", err)
    }

    fmt.Printf("Keys generated:\n - Private: %s\n - Public: %s\
       n", privateKeyPath, publicKeyPath)
    return nil
}

func signFile(inputFile, privateKeyPath, signatureFile string)
   error {
    privateKeyData, err := os.ReadFile(privateKeyPath)
    if err != nil {
        return fmt.Errorf("private key read error: %v", err)
    }

    block, _ := pem.Decode(privateKeyData)
    if block == nil {
        return fmt.Errorf("failed to decode private key PEM
           block")
    }
```

```
    privateKey, err := x509.ParsePKCS1PrivateKey(block.Bytes)
    if err != nil {
        return fmt.Errorf("private key parsing error: %v", err)
    }

    data, err := os.ReadFile(inputFile)
    if err != nil {
        return fmt.Errorf("file read error: %v", err)
    }

    hash := sha256.Sum256(data)

    signature, err := rsa.SignPKCS1v15(rand.Reader, privateKey,
        crypto.SHA256, hash[:])
    if err != nil {
        return fmt.Errorf("signature creation error: %v", err)
    }

    if err := os.WriteFile(signatureFile, signature, 0644); err
        != nil {
        return fmt.Errorf("signature save error: %v", err)
    }

    fmt.Printf("Signature saved to %s\n", signatureFile)
    return nil
}

func verifySignature(inputFile, publicKeyPath, signatureFile
   string) error {
    publicKeyData, err := os.ReadFile(publicKeyPath)
    if err != nil {
        return fmt.Errorf("public key read error: %v", err)
    }

    block, _ := pem.Decode(publicKeyData)
    if block == nil {
        return fmt.Errorf("failed to decode public key PEM
           block")
    }

    publicKeyInterface, err := x509.ParsePKIXPublicKey(block.
```

```
        Bytes)
    if err != nil {
        return fmt.Errorf("public key parsing error: %v", err)
    }

    publicKey, ok := publicKeyInterface.(*rsa.PublicKey)
    if !ok {
        return fmt.Errorf("invalid public key type")
    }

    data, err := os.ReadFile(inputFile)
    if err != nil {
        return fmt.Errorf("file read error: %v", err)
    }

    hash := sha256.Sum256(data)

    signature, err := os.ReadFile(signatureFile)
    if err != nil {
        return fmt.Errorf("signature read error: %v", err)
    }

    err = rsa.VerifyPKCS1v15(publicKey, crypto.SHA256, hash[:],
        signature)
    if err != nil {
        return fmt.Errorf("signature is invalid: %v", err)
    }

    fmt.Println("Signature is valid.")
    return nil
}

func main() {
    if len(os.Args) < 2 {
        printUsage()
        os.Exit(1)
    }

    command := os.Args[1]

    switch command {
```

```
        case "gen-keys":
        privateKey := "private.pem"
        publicKey := "public.pem"

        for i := 2; i < len(os.Args); i++ {
            arg := os.Args[i]
            switch arg {
                case "-priv", "--private":
                if i+1 < len(os.Args) {
                    privateKey = os.Args[i+1]
                    i++
                }
                case "-pub", "--public":
                if i+1 < len(os.Args) {
                    publicKey = os.Args[i+1]
                    i++
                }
            }
        }

        if err := generateKeys(privateKey, publicKey); err !=
           nil {
            fmt.Printf("Error: %v\n", err)
            os.Exit(1)
        }

        case "sign":
        if len(os.Args) < 5 {
            fmt.Println("Error: insufficient arguments for sign
               command")
            fmt.Println("Usage: sign <input_file> -priv <
               private_key> [-o signature.bin]")
            os.Exit(1)
        }

        inputFile := os.Args[2]
        privateKey := ""
        output := "signature.bin"

        for i := 3; i < len(os.Args); i++ {
            arg := os.Args[i]
```

```
            switch arg {
                case "-priv", "--private":
                if i+1 < len(os.Args) {
                    privateKey = os.Args[i+1]
                    i++
                }
                case "-o", "--output":
                if i+1 < len(os.Args) {
                    output = os.Args[i+1]
                    i++
                }
            }
        }

        if privateKey == "" {
            fmt.Println("Error: private key must be specified
               with -priv")
            os.Exit(1)
        }

        if err := signFile(inputFile, privateKey, output); err
           != nil {
            fmt.Printf("Error: %v\n", err)
            os.Exit(1)
        }

        case "verify":
        if len(os.Args) < 6 {
            fmt.Println("Error: insufficient arguments for
               verify command")
            fmt.Println("Usage: verify <input_file> -pub <
               public_key> -s <signature_file>")
            os.Exit(1)
        }

        inputFile := os.Args[2]
        publicKey := ""
        signature := ""

        for i := 3; i < len(os.Args); i++ {
            arg := os.Args[i]
```

```
            switch arg {
                case "-pub", "--public":
                if i+1 < len(os.Args) {
                    publicKey = os.Args[i+1]
                    i++
                }
                case "-s", "--signature":
                if i+1 < len(os.Args) {
                    signature = os.Args[i+1]
                    i++
                }
            }
        }

        if publicKey == "" {
            fmt.Println("Error: public key must be specified
               with -pub")
            os.Exit(1)
        }
        if signature == "" {
            fmt.Println("Error: signature file must be
               specified with -s")
            os.Exit(1)
        }

        if err := verifySignature(inputFile, publicKey,
           signature); err != nil {
            fmt.Printf("Error: %v\n", err)
            os.Exit(1)
        }

        default:
        fmt.Printf("Unknown command: %s\n", command)
        printUsage()
        os.Exit(1)
    }
}

func printUsage() {
    fmt.Println("Usage:")
    fmt.Println("  gen-keys [-priv private.pem] [-pub public.
```

```
        pem]")
    fmt.Println("  sign <input_file> -priv <private_key> [-o
        signature.bin]")
    fmt.Println("  verify <input_file> -pub <public_key> -s <
        signature_file>")
    fmt.Println("")
    fmt.Println("Examples:")
    fmt.Println("  go run ./src/main.go gen-keys -priv private.
        pem -pub public.pem")
    fmt.Println("  go run ./src/main.go sign ./data/text.txt -
        priv private.pem -o signature.bin")
    fmt.Println("  go run ./src/main.go verify ./data/text.txt
        -pub public.pem -s signature.bin")
}
```

Листинг 4.1 – Реализация программы создания и проверки электронной подписи

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы был реализован алгоритм симметричного шифрования (AES).

В процессе выполнения данной работы были выполнены все задачи:

1) описать алгоритм симметричного шифрования (AES);

2) реализовать виде программы алгоритм симметричного шифрования (AES);