**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное автономное 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 информация»

**Тема** Создание и проверка электронной подписи документа

**Студент** Шахнович Дмитрий Сергеевич

**Группа** ИУ7-72Б

**Преподаватель** Руденкова Ю.С.

Москва, 2025

# Содержание

# ВВЕДЕНИЕ

**Цель работы:** Создание и проверка простой электронной подписи.

**Задачи:**

1) проанализировать виды электронной подписи;
2) разработать алгоритм подписывания документов простой электронной подписью;
3) разработать алгоритм проверки простой электронной подписи;
4) обеспечить подпись произвольного текстового файла и его проверку.

# 1 Аналитическая часть

## 1.1 Определения

В соответствии с Федеральным законом № 63-ФЗ «Об электронной подписи» различают три вида электронной подписи: простую, неквалифицированную и квалифицированную.

### 1.1.1 Простая электронная подпись

**Простая электронная подпись** — электронная подпись, которая посредством использования кодов, паролей или иных средств подтверждает факт формирования электронной подписи определённым лицом (статья 5, пункт 2 Федерального закона № 63-ФЗ).

Данный вид подписи не требует применения средств криптографической защиты и обеспечивает только идентификацию подписанта, но не гарантирует целостность подписанного документа.

Примеры реализации простой электронной подписи:

— ввод логина и пароля в личном кабинете;

— подтверждение через SMS-код;

— отметка «согласен» на веб-форме;

— добавление в файл хэша пароля пользователя.

### 1.1.2 Неквалифицированная электронная подпись

**Неквалифицированная электронная подпись** — электронная подпись, которая:

1) получена в результате криптографического преобразования информации с использованием ключа электронной подписи;

2) позволяет определить лицо, подписавшее электронный документ;

3) позволяет обнаружить факт внесения изменений в электронный документ после момента его подписания;

4) создаётся с использованием средств электронной подписи.

(статья 5, пункт 3 Федерального закона № 63-ФЗ).

Таким образом, неквалифицированная подпись уже использует криптографические методы, обеспечивает контроль целостности и может применяться в большинстве внутренних информационных систем.

### 1.1.3 Квалифицированная электронная подпись

**Квалифицированная электронная подпись** — электронная подпись, которая соответствует всем признакам неквалифицированной электронной подписи и обладает следующими дополнительными признаками:

1) ключ проверки электронной подписи указан в квалифицированном сертификате;

2) для создания и проверки электронной подписи используются средства электронной подписи, имеющие подтверждение соответствия требованиям, установленным Федеральным законом № 63-ФЗ.

Квалифицированная подпись имеет полную юридическую силу и используется при взаимодействии с государственными органами и при оформлении юридически значимых документов.

## 1.2 Сравнение видов электронной подписи

| **Признак** | **ПЭП** | **НЭП** | **КЭП** |
|---|---|---|---|
| Использует криптографию | Нет | Да | Да |
| Позволяет определить подписанта | Да (по паролю, коду) | Да | Да |
| Позволяет проверить целостность документа | Нет | Да | Да |
| Использует сертифицированные средства | Нет | Нет | Да |
| Юридическая сила | По соглашению сторон | Средняя | Полная |

Таблица 1.1 — Сравнение видов электронной подписи согласно 63-ФЗ

## 1.3 Особенности простой электронной подписи

Простая электронная подпись (ПЭП) не обеспечивает защиту от изменения данных и не гарантирует неизменность документа после подписания. Её основная функция — подтверждение того, что подпись сформировал конкретный пользователь.

Для реализации ПЭП достаточно:

— зарегистрировать пользователя в системе;

— сформировать подпись на основе известного пользователю пароля или кода;

— хранить хэш пароля для последующей проверки подлинности подписи.

## 1.4 Алгоритм bcrypt

**bcrypt** — это криптографическая функция хеширования паролей, основанная на алгоритме блочного шифра `EksBlowfish`. Она предназначена для безопасного хранения паролей и защиты от атак перебора (brute-force).

### 1.4.1 Принцип работы bcrypt

Алгоритм bcrypt выполняет следующие шаги:

1) Генерируется случайная «соль» длиной 128 бит.

2) Пароль пользователя комбинируется с этой солью.

3) Выполняется многократное (десятки или сотни тысяч раз) применение алгоритма Blowfish с параметром `cost`, определяющим вычислительную сложность.

4) Результатом является хэш фиксированной длины (60 символов в кодировке Base64), который включает в себя:

— версию алгоритма;

— параметр `cost`;

— соль;

— итоговый хэш-код.

### 1.4.2 Свойства bcrypt

— Защищён от атак с использованием радужных таблиц (заранее заготовленных хеш-таблиц) благодаря индивидуальной соли.

— Позволяет регулировать сложность вычислений (`cost`), что делает атаки перебора крайне затратными.

— При проверке пароля алгоритм повторно вычисляет хэш с теми же параметрами и сравнивает его с сохранённым.

### 1.4.3 Использование bcrypt в ПЭП

В данной работе bcrypt применяется для безопасного хранения хэша пароля пользователя. Так как простая электронная подпись не требует защиты целостности данных, хэш bcrypt служит средством подтверждения факта, что файл был подписан владельцем данного пароля.

# 2 Технологическая часть

## 2.1 Средства реализации

В качестве языка программирования выбран Go. Go обладает встроенными средствами работы с файлами, строками и хешированием паролей (пакет `golang.org/x/crypto/bcrypt`), что делает его удобным для реализации ПЭП.

## 2.2 Алгоритм формирования простой электронной подписи

Алгоритм можно описать следующими шагами:

1) Пользователь передаёт логин, пароль и исходный файл программе.
2) Пароль хэшируется с помощью алгоритма `bcrypt`.
3) Программа читает исходный файл и создаёт новый файл, содержащий:
   - оригинальные данные;
   - логин пользователя;
   - секцию подписи (хэш пароля);
   - отметку времени подписи.

## 2.3 Алгоритм проверки подписи

Для проверки:

1) Пользователь подаёт на вход логин, пароль и файл с подписью.
2) Из нового файла извлекается логин и хэш пароля.
3) Программа сравнивает введённый логин с сохранённым логином, пароль с сохранённым хэшем с помощью функции.
4) Совпадение означает, что документ подписан тем пользователем.

## 2.4 Код программы

Листинг 2.1 — Код создания и проверки подписи

```
package main

import (
  "bufio"
  "bytes"
  "encoding/base64"
  "fmt"
  "os"
  "strings"
  "time"
```

```
  "golang.org/x/crypto/bcrypt"
)

const signatureMarker = "\n---BEGIN SIMPLE SIGNATURE---\n"

func usage() {
  fmt.Println(`Usage:
  sign   <username> <password> <input> <output>    sign simple signature
     into file
  verify <username> <password> <signedfile>        Verify simple signature
     in file`)
}

// sign appends a signature block to the file
func sign(username, password, inputFile, outputFile string) error {
  data, err := os.ReadFile(inputFile)
  if err != nil {
    return err
  }

  // bcrypt hash of password
  hashed, err := bcrypt.GenerateFromPassword([]byte(password), bcrypt.
     DefaultCost)
  if err != nil {
    return err
  }

  timestamp := time.Now().Format(time.RFC3339)
  signatureBlock := fmt.Sprintf("%sUSER=%s\nTIME=%s\nPASSHASH=%s\n---END
     SIMPLE SIGNATURE---\n",
    signatureMarker, username, timestamp, base64.StdEncoding.
       EncodeToString(hashed))

  out := append(data, []byte(signatureBlock)...)
  return os.WriteFile(outputFile, out, 0644)
}

// verify compares provided password with stored hash
func verify(username, password, signedFile string) error {
  data, err := os.ReadFile(signedFile)
```

```
if err != nil {
  return err
}

// split content and signature
parts := bytes.SplitN(data, []byte(signatureMarker), 2)
if len(parts) != 2 {
  return fmt.Errorf("no signature block found")
}

sigPart := string(parts[1])
scanner := bufio.NewScanner(strings.NewReader(sigPart))

var sigUser, sigTime, sigHash string
for scanner.Scan() {
  line := scanner.Text()
  if strings.HasPrefix(line, "USER=") {
    sigUser = strings.TrimPrefix(line, "USER=")
  } else if strings.HasPrefix(line, "TIME=") {
    sigTime = strings.TrimPrefix(line, "TIME=")
  } else if strings.HasPrefix(line, "PASSHASH=") {
    sigHash = strings.TrimPrefix(line, "PASSHASH=")
  }
}

if sigUser == "" || sigHash == "" {
  return fmt.Errorf("invalid signature format")
}
if sigUser != username {
  return fmt.Errorf("username mismatch: expected %s, got %s", username,
     sigUser)
}

hashedBytes, err := base64.StdEncoding.DecodeString(sigHash)
if err != nil {
  return fmt.Errorf("bad encoding in signature: %v", err)
}

// bcrypt comparison
if err := bcrypt.CompareHashAndPassword(hashedBytes, []byte(password));
   err != nil {
```

```
    return fmt.Errorf("signature invalid (wrong password)")
  }

  fmt.Printf("Signature valid.\nSigned by: %s at %s\n", sigUser, sigTime)
  return nil
}

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

  switch os.Args[1] {
  case "sign":
    if len(os.Args) != 6 {
      usage()
      os.Exit(2)
    }
    if err := sign(os.Args[2], os.Args[3], os.Args[4], os.Args[5]); err !=
        nil {
      fmt.Println("Error:", err)
      os.Exit(1)
    }
    fmt.Println("File signed successfully (simple signature signded).")

  case "verify":
    if len(os.Args) != 5 {
      usage()
      os.Exit(2)
    }
    if err := verify(os.Args[2], os.Args[3], os.Args[4]); err != nil {
      fmt.Println("Error:", err)
      os.Exit(1)
    }

  default:
    usage()
    os.Exit(2)
  }
}
```

## 2.5 Описание работы программы

Программа состоит из двух режимов:

1) **sign** — создаёт новый файл, содержащий исходные данные и подпись.

2) **verify** — проверяет подпись по введённому паролю.

Пример запуска:

```
go run main.go sign impi impi test.txt test.sig
go run main.go verify impi impi test.sig
```

При создании подписи в конец файла добавляется блок следующего вида:

```
---BEGIN SIMPLE SIGNATURE---
USER=impi
TIME=2025-11-10T00:46:25+03:00
PASSHASH=JDJhJDEwJDNNTUp0SmZQLjZ1V2xzZ1AwaFR2dmVaWEt5eExVQ0ZJeTFOZXZTUmU2dWRTSFpjV
---END SIMPLE SIGNATURE---
```

При проверке программа извлекает хэш и сверяет введённый пароль.

## 2.6 Пример работы программы

В качестве примера программы был взят текст из книги «Принципы юнит-тестирования» автора Владимир Хориков, длиной примерно 13000 символов.

На рисунке 2.1 представлен процесс подписи файла, проверки подписи с корректным паролем, и не корректным.

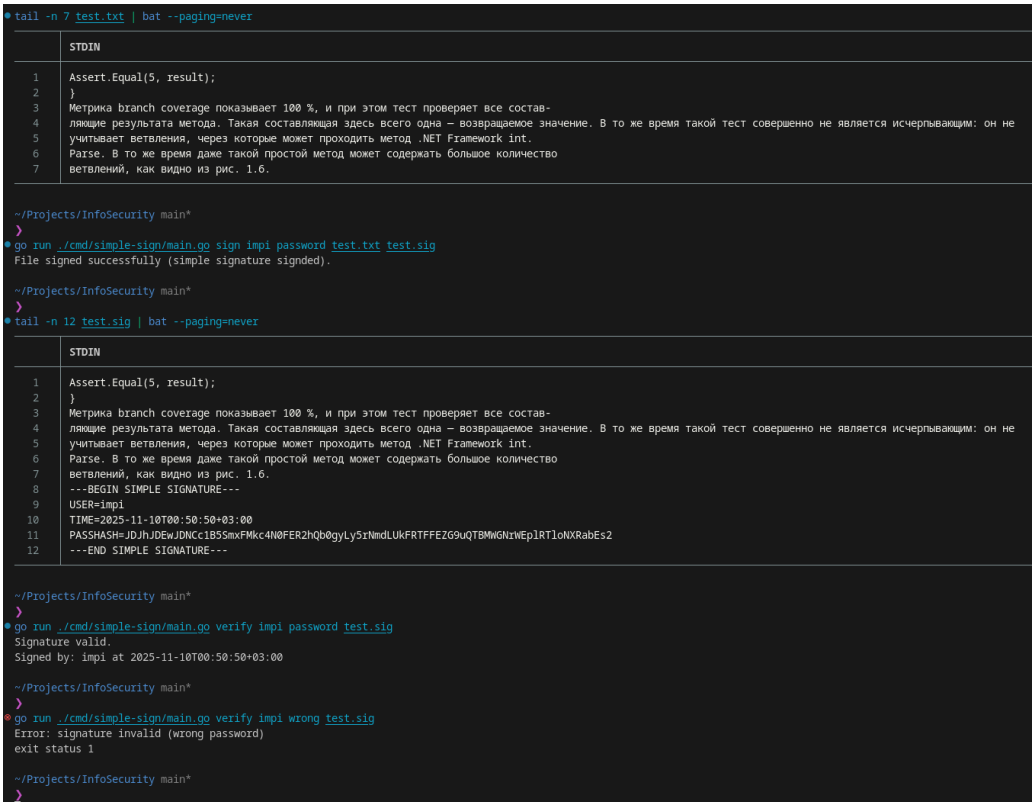


Рисунок 2.1 — Подпись и проверка текстового файла

# ЗАКЛЮЧЕНИЕ

В результате лабораторной работы была разработана программа для простой электронной подписи.

Были выполнены следующие задачи:

1) проанализированы виды электронной подписи;
2) разработаны алгоритмы подписания и проверки подписи;
3) реализован алгоритм формирования простой электронной подписи на языке Go;
4) реализована проверка подписи.

Все поставленные цели и задачи были выполнены.