

## Лабораторная работа № 3.

### Создание электронной подписи в документе

Карасев Илья Алексеевич, М23-505

#### Цель работы, постановка задачи

Разработка процедур выработки и проверки электронной цифровой подписи (ЭЦП) сообщений на базе асимметричного криптографического алгоритма с применением функции хеширования

1. Реализовать программную реализацию алгоритма создания и проверки электронно-цифровой подписи.
2. Подписать текстовое сообщение
3. Проверить правильность ЭЦП.
4. Внести изменения в сделанную подпись. Убедится, что подпись не является подлинной.
5. Результаты работы оформить в виде отчета.

#### Описание исходных данных

$p = 23$

$q = 11$

$a = 6$

$x = 8$

Для контрольной суммы выбран 1й вариант: Количество 1 в битовом представлении символов исходного текста

Язык программирования: Python

#### Алгоритм работы программы

1. На вход программе дается строка для подписи и опционально секретный ключ (с помощью параметры -s). Если ключ не задан по умолчанию используется число 8
2. Инициализируется экземпляр класса DigitalSigGost
  - a. Проверяется соответствие секретного ключа  $1 < x < q-1$ . В случае неудовлетворения условия программа завершается с ошибкой
  - b. Рассчитывается открытый ключ  $y$
3. Создается электронная подпись
  - a. Рассчитывается контрольная сумма (Количество 1 в битовом представлении символов исходного текста)
  - b. Генерируется пара чисел  $(r1, s)$  которая и является электронной подписью
4. Электронная подпись выводится в консоль
5. Проверяется правильность ЭЦП
  - a. Проверяется значения электронной подписи:  $0 < r1 < q, 0 < s < q$ , если условие не соблюдено – подпись не верна
  - b. Вычисляется значение  $u$  и производится сравнение с  $r1$ , если  $u$  равно  $r1$ , то проверка успешна, иначе нет
6. Результат проверки выводится в консоль
7. Вносятся изменения в значения электронной подписи

8. Осуществляется повторная проверка подписи
9. Результат проверки выводится в консоль

## Текст программы

```
"""
Реализация ЭЦП в соответствии с ЭЦП ГОСТ Р34.10-94
"""

import argparse
import random
from typing import Tuple

class DigitalSigGost:
    """Предоставляет методы для генерации подписи и ее проверки
    по аналогии с ЭЦП ГОСТ Р34.10-94
    """

    def __init__(self, private=8) -> None:
        # Для ускорения работы воспользуемся небольшими p,q,a
        self.p = 23
        self.q = 11
        self.a = 6

        if private < 2 or private > self.q - 1:
            raise ValueError(
                f"Приватный ключ x={private} должен быть в диапазоне (0, {self.q})"
            )

        self.x = private
        self.y = pow(self.a, private, self.p)

    def get_hash(self, message: str) -> int:
        """Расчитывает хэш-код (контрольную сумму) для заданной строки
        Используется метод подсчета 1 в битовом
        представлении символов исходного текста
        """
        bites = "".join([f"{ord(c):b}" for c in message])
        ones = bites.count("1")
        h = 1 if ones % self.q == 0 else ones
        return h

    def create_sign(self, message: str) -> Tuple[int, int]:
        """Генерирует электронно-цифровую подпись
        для заданного текста
        """
        h = self.get_hash(message)
        k = 0
        r1 = 0
        s = 0
        while r1 == 0 or s == 0:
            k = random.randrange(1, self.q + 1)
            r1 = pow(self.a, k, self.p) % self.q
            s = (self.x * r1 + k * h) % self.q
        return (r1, s)

    def check_sign(self, message: str, sign: Tuple[int, int]) -> bool:
        """Проверка цифровой подписи"""
        r1, s = sign

        # Проверка параметров подписи
        if r1 <= 0 or r1 >= self.q or s <= 0 or s >= self.q:
            return False

        # Проверка подленности подписи
        h = self.get_hash(message)
        v = pow(h, self.q - 2, self.q)
        z1 = s * v % self.q
        z2 = (self.q - r1) * v % self.q
        u = (pow(self.a, z1) * pow(self.y, z2) % self.p) % self.q

        return u == r1
```

```

parser = argparse.ArgumentParser(
    prog="Создание подписи и ее проверка",
)

parser.add_argument(
    "text",
    type=str,
    help="Текст на подпись",
)

parser.add_argument(
    "-s",
    "--secret",
    type=int,
    action="store",
    default=8,
    help="Секретный ключ",
)

args = parser.parse_args()

d = DigitalSigGost(args.secret)
ds = d.create_sign(args.text)

print("Подпись: ", ds)

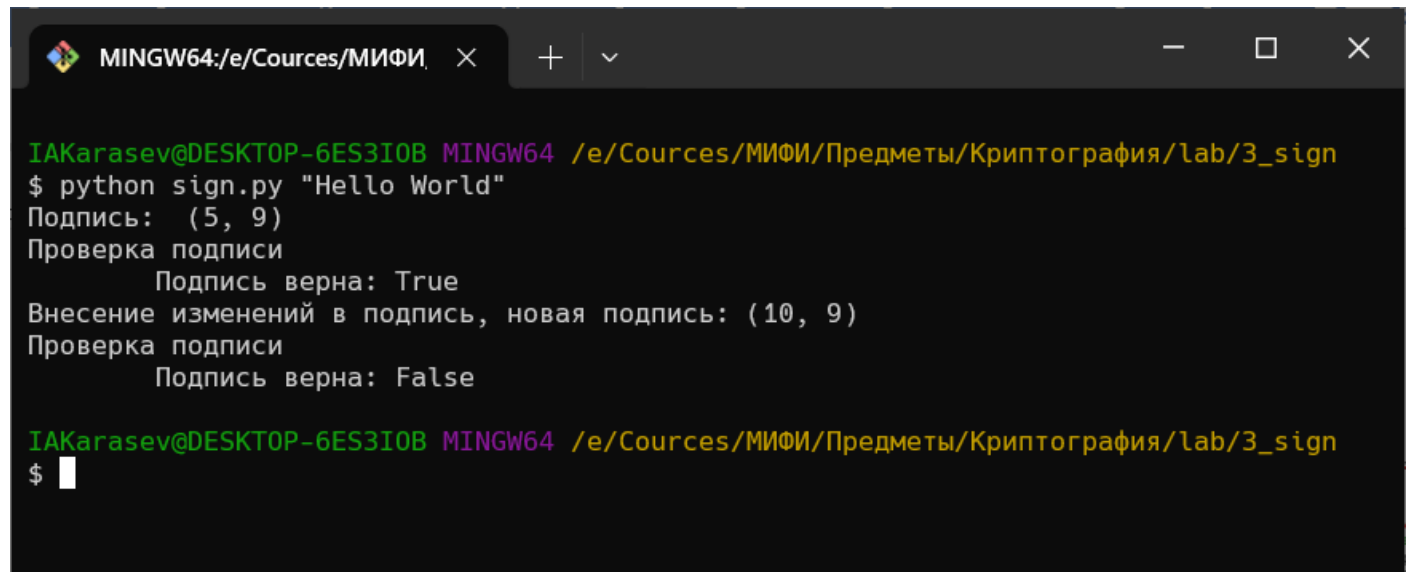
print("Проверка подписи")
print(f"\tПодпись верна: {d.check_sign(args.text, ds)}")

ds = (ds[0] * 2, ds[1])
print(f"Внесение изменений в подпись, новая подпись: {ds}")

print("Проверка подписи")
print(f"\tПодпись верна: {d.check_sign(args.text, ds)}")

```

## Результаты работы программы



The screenshot shows a terminal window with the following content:

```

MINGW64:/e/Cources/МИФИ
IAKarasev@DESKTOP-6ES3I0B MINGW64 /e/Cources/МИФИ/Предметы/Криптография/lab/3_sign
$ python sign.py "Hello World"
Подпись: (5, 9)
Проверка подписи
    Подпись верна: True
Внесение изменений в подпись, новая подпись: (10, 9)
Проверка подписи
    Подпись верна: False

IAKarasev@DESKTOP-6ES3I0B MINGW64 /e/Cources/МИФИ/Предметы/Криптография/lab/3_sign
$

```

## Выводы

На практике ознакомились с алгоритмом составления и проверки электронной подписи на примере ЭЦП ГОСТ 34.10-94

## Контрольные вопросы

1. Какие криптоалгоритмы используются для создания электронной цифровой подписи?  
В основном используются асимметричные криптографические алгоритмы, такие как RSA, DSA (Digital Signature Algorithm), ЭЦП ГОСТ34.10-2018

**2. Что такое криптографическая хэш-функция, какими свойствами она должна обладать?**

математическое преобразование, которое принимает входные данные (сообщение) произвольной длины и преобразует их в фиксированную длину

Основные свойства хэш-функции:

- a. Однонаправленность (необратимость): практически невозможно восстановить исходные данные из результата хэш-функции
- b. Устойчивость к коллизиям: минимизация возможности получения одинаковых хэшей для отличных исходных данных
- c. Быстродействие
- d. Равномерность распределения: равномерное распределение хэш-значений по всем возможным входным данным, чтобы предотвратить атаки, основанные на неоднородности распределения
- e. Неизменяемость: минимальные изменения в исходных данных должны приводить к значительным изменениям в хэш-значении

**3. Как содержание сообщения влияет на электронную цифровую подпись?**

Содержание сообщения оказывает существенное влияние на сам процесс создания и проверки электронной цифровой подписи. Небольшие изменения в сообщении должны приводить к существенным изменениям в ЦП. При изменении сообщения после подписи приводит к недействительности подписи. В большинстве случаев, длина сообщения не влияет на длину подписи.

**4. Где используется ЭЦП?**

ЭЦП нашли применение во многих областях. Например: электронный документооборот, авторизация пользователей в информационных системах, финансовых транзакциях, интернет-банкинге, электронной почте, электронное удостоверение личности и т.д.

**5. В каком случае электронная цифровая подпись при проверке отвергается?**

ЭЦП отвергается при внесении изменений в сообщение после подписания, при внесении изменения в саму ЭЦП. Если у ЭЦП имеется срок действия, то по его истечению она так же отвергается

**6. От каких угроз информации защищает ЭЦП?**

- a. Неавторизованный доступ и подделка данных – ЭЦП позволяет удостовериться в подлинности отправителя
- b. Целостность данных – гарантируется неизменность исходных данных после их подписания
- c. Отказ от подписи – владелец подписи не может от нее отказаться, т.к. владелец закрытого ключа создает и удостоверяет ее подлинность и действительность
- d. Подделка идентификации - только владелец закрытого ключа может успешно подписать данные