Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ

к лабораторной работе №6

на тему

**ЦИФРОВАЯ ПОДПИСЬ**

БГУИР КП 1-40 04 01 025 ПЗ

Выполнил: студент гр.253504

Фроленко К.Ю.

Проверил: ассистент кафедры информатики Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

[1 Формулировка задачи 3](#_Toc209095978)

[2 Теоретеческие сведения 4](#_Toc209095979)

[3 Ход работы 6](#_Toc209095980)

[Заключение 7](#_Toc209095981)

[Приложение А (обязательное) Листинг программного кода 8](#_Toc209095982)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Современное развитие информационных технологий и повсеместное использование компьютерных сетей привели к увеличению объёмов передаваемой и хранимой информации, что требует обеспечения её целостности, подлинности и неотказуемости. Электронная цифровая подпись (ЭЦП) является важным инструментом, обеспечивающим данные свойства.  
 В данной лабораторной работе ставится задача изучить один из отечественных стандартов формирования и проверки ЭЦП – ГОСТ 34.102012, и на его основе разработать программное средство, позволяющее выполнять генерацию пары ключей, формирование подписи и её проверку.  
Особенность лабораторной работы заключается в том, что реализация должна быть выполнена на языке Python без использования внешних библиотек, с возможностью чтения и записи данных из/в файлы.  
Таким образом, цель работы: изучить теоретические основы работы ГОСТ 34.10-2012, реализовать программный алгоритм формирования и проверки ЭЦП, протестировать его на примерах и убедиться в корректности работы.  
 Для достижения поставленной цели необходимо:

1. Рассмотреть структуру алгоритма цифровой подписи, основанного на проблеме дискретного логарифма.

2. Описать процесс генерации ключей (закрытого и открытого).

3. Изучить этапы формирования и проверки подписи.

4. Реализовать все стадии алгоритма (хеширование, подпись, проверка).

5. Организовать процедуру чтения и записи данных в файлы.

6. Разработать интерфейс командной строки, позволяющий запускать программу в режимах keygen, sign, verify.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

ГОСТ 34.10-2012 – это стандарт Российской Федерации, описывающий алгоритмы формирования и проверки электронной цифровой подписи. Алгоритм основан на вычислительной сложности решения задачи дискретного логарифмирования в конечном поле. В упрощённой версии, реализуемой в данной работе, используется модульное арифметическое представление, а не эллиптические кривые. Тем не менее, структура остаётся схожей с оригинальным стандартом.

Основные этапы работы:

1. Генерация параметров и ключей:

– Выбираются параметры p, q, a, где q – простой делитель p - 1, a –первообразный корень.

– Закрытый ключ x – случайное число в диапазоне [1, q - 1].

– Открытый ключ y = a^x mod p.

1. Формирование подписи:

– Сообщение M хешируется: h = SHA-256(M) mod q.

– Если h = 0, то h = 1.

– Случайное k выбирается в диапазоне [1, q - 1].

– Вычисляем первую компоненту подписи: r = (a^k mod p) mod q.

– Вычисляем вторую компоненту подписи: s = (k \* h + x \* r) mod q.

– Подпись – пара (r, s).

1. Проверка подписи:

– Проверяется, что 0 < r < q и 0 < s < q.

– Вычисляем обратный элемент от хеша: e = h^(-1) mod q.

– Вычисляем вспомогательные значения: z1 = (s \* e) mod q, z2 = ((q - r) \* e) mod q.

– Вычисляем результат проверки: u = ((a^z1 \* y^z2) mod p) mod q.

– Подпись считается валидной, если u == r.

Этапы обеспечивают неотказуемость, целостность и аутентичность подписываемого сообщения.

Изображение выглядит как текст, диаграмма, чек, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 1 – Схема процесса формирования цифровой подписи

3 ХОД РАБОТЫ

Для выполнения лабораторной работы был реализован программный комплекс на языке Python, воспроизводящий работу ГОСТ 34.10-2012 в упрощённой версии.  
Алгоритм был реализован строго в соответствии с описанием стандарта:

– Генерация ключей: x – случайное, y = a^x mod p.

– Формирование подписи: хеширование, вычисление r и s.

– Проверка подписи: вычисление u и сравнение с r.

В качестве входных данных использовался текстовый файл.  
Результаты сохранялись в файлы в формате, удобном для проверки.  
Отладочный режим программы был реализован так, чтобы пользователь мог наблюдать промежуточные значения:

– Подписываемое сообщение.

– Сгенерированная подпись (r, s).

– Результат проверки (VALID / INVALID).

Для начала работы необходимо сгенерировать пару ключей – закрытый и открытый.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 2 – Генерация пары ключей с заданными параметрами

Для формирования подписи на сообщение test, которое содержится в файле in.txt, используется команда:

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3 – Формирование цифровой подписи

Для проверки подписи используется команда:



Рисунок 4 – Проверка подписи на действительность

Таким образом, при выполнении всех этапов программного обеспечения демонстрируется реализация алгоритма ГОСТ 34.10. Подпись формируется и проверяется.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была достигнута основная цель – реализовать на практике один из отечественных стандартов формирования и проверки электронной цифровой подписи – ГОСТ 34.10-2012.  
 В процессе изучения были рассмотрены основные элементы алгоритма:

– Генерация ключей.

– Формирование подписи.

– Проверка подписи.

Особое внимание уделялось пониманию того, как модульная арифметика, хеширование и обратные элементы в совокупности обеспечивают криптографическую стойкость. Программа была реализована без использования внешних библиотек, что позволило глубже понять внутренние механизмы. Практическая часть продемонстрировала корректность реализации:

– Подпись формируется и проверяется.

– Все промежуточные данные соответствуют описанию ГОСТ 34.10.

Таким образом, поставленные в работе задачи были выполнены полностью. Полученное программное средство можно использовать как учебный инструмент для понимания принципов формирования и проверки ЭЦП. Лабораторная работа позволила закрепить знания о криптографических методах защиты информации, а также на практике убедиться в том, что теоретические принципы, изложенные в стандарте, корректно реализуются средствами современного программирования.

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг программного кода**

import sys

import os

import hashlib

import secrets

from typing import Tuple

PRIVATE\_KEY\_FILE = "private.key"

PUBLIC\_KEY\_FILE = "public.key"

DEFAULT\_PARAMS\_FILE = "params.txt"

DEFAULT\_P = 33563

DEFAULT\_Q = 173

DEFAULT\_A = 10918

def read\_params(params\_path: str = DEFAULT\_PARAMS\_FILE) -> Tuple[int,int,int]:

    if not os.path.exists(params\_path):

        return DEFAULT\_P, DEFAULT\_Q, DEFAULT\_A

    p=q=a=None

    with open(params\_path, "r", encoding="utf-8") as f:

        for ln in f:

            ln = ln.split("#",1)[0].strip()

            if not ln:

                continue

            if "=" in ln:

                k,v = ln.split("=",1)

                k=k.strip().lower(); v=v.strip()

                if k=="p": p=int(v)

                elif k=="q": q=int(v)

                elif k=="a": a=int(v)

    if p is None or q is None or a is None:

        raise ValueError("params.txt должен содержать p=, q= и a=")

    return p,q,a

def modinv(a: int, m: int) -> int:

    a %= m

    if a == 0:

        raise ZeroDivisionError("Inverse does not exist")

    lm, hm = 1, 0

    low, high = a, m

    while low > 1:

        r = high // low

        nm = hm - lm \* r

        new = high - low \* r

        hm, lm = lm, nm

        high, low = low, new

    return lm % m

def hash\_to\_int(data: bytes, q: int) -> int:

    h = hashlib.sha256(data).digest()

    hv = int.from\_bytes(h, "big")

    return hv % q

def keygen(params\_path: str = DEFAULT\_PARAMS\_FILE):

    p,q,a = read\_params(params\_path)

    x = secrets.randbelow(q - 1) + 1

    y = pow(a, x, p)

    with open(PRIVATE\_KEY\_FILE, "w", encoding="utf-8") as f:

        f.write(str(x) + "\n")

    with open(PUBLIC\_KEY\_FILE, "w", encoding="utf-8") as f:

        f.write(f"{p}\n{q}\n{a}\n{y}\n")

    print("Ключи сгенерированы.")

    print(f"private -> {PRIVATE\_KEY\_FILE}")

    print(f"public  -> {PUBLIC\_KEY\_FILE}")

    print(f"Параметры: p={p}, q={q}, a={a}")

def sign\_file(in\_path: str, out\_path: str, params\_path: str = DEFAULT\_PARAMS\_FILE):

    if not os.path.exists(PRIVATE\_KEY\_FILE):

        print("Ошибка: private.key не найден. Выполните keygen.")

        return

    with open(PRIVATE\_KEY\_FILE, "r", encoding="utf-8") as f:

        x = int(f.read().strip().splitlines()[0])

    p,q,a = read\_params(params\_path)

    msg = open(in\_path, "rb").read()

    e = hash\_to\_int(msg, q)

    if e == 0:

        e = 1

    while True:

        k = secrets.randbelow(q - 1) + 1

        r = pow(a, k, p) % q

        if r == 0:

            continue

        s = (k \* e + x \* r) % q

        if s == 0:

            continue

        break

    with open(out\_path, "w", encoding="utf-8") as f:

        f.write(f"{r} {s}\n")

    print(f"Подпись записана в {out\_path}")

    print(f"r={r}, s={s}")

    verify\_input = "verify\_input.txt"

    with open(in\_path, "r", encoding="utf-8") as fin:

        message\_text = fin.read()

    with open(verify\_input, "w", encoding="utf-8") as f:

        f.write(f"{r} {s}\n{message\_text}")

    print(f"Файл для проверки создан: {verify\_input}")

def verify\_file(in\_path: str, out\_path: str, params\_path: str = DEFAULT\_PARAMS\_FILE):

    if not os.path.exists(PUBLIC\_KEY\_FILE):

        print("Ошибка: public.key не найден. Выполните keygen.")

        return

    with open(PUBLIC\_KEY\_FILE, "r", encoding="utf-8") as f:

        lines = [ln.strip() for ln in f if ln.strip()]

        if len(lines) < 4:

            print("public.key должен содержать p, q, a, y (по одной строке).")

            return

        p = int(lines[0]); q = int(lines[1]); a = int(lines[2]); y = int(lines[3])

    txt = open(in\_path, "r", encoding="utf-8").read().splitlines()

    if not txt:

        print("in.txt пустой.")

        return

    sig\_line = txt[0].strip()

    try:

        r\_str, s\_str = sig\_line.split()

        r = int(r\_str); s = int(s\_str)

    except Exception:

        print("Неверный формат подписи в первой строке in.txt. Ожидается 'r s'.")

        return

    message = ""

    if len(txt) > 1:

        message = "\n".join(txt[1:])

    msg\_bytes = message.encode("utf-8")

    e = hash\_to\_int(msg\_bytes, q)

    if e == 0:

        e = 1

    if not (0 < r < q and 0 < s < q):

        result = "INVALID"

    else:

        try:

            v = modinv(e, q)

        except Exception:

            result = "INVALID"

        else:

            z1 = (s \* v) % q

            z2 = ((q - r) \* v) % q

            u = (pow(a, z1, p) \* pow(y, z2, p)) % p

            u = u % q

            result = "VALID" if u == r else "INVALID"

    with open(out\_path, "w", encoding="utf-8") as f:

        f.write(result + "\n")

    print(f"Проверка: {result} (результат записан в {out\_path})")

def usage():

    print("Использование:")

    print("  python main.py keygen params.txt dummy")

    print("  python main.py sign in.txt out.txt")

    print("  python main.py verify verify\_input.txt result.txt")

    print("")

    print("Формат params.txt (опционально): p=<число> q=<число> a=<число>")

    print("Формат in.txt для verify: первая строка 'r s', начиная со 2-й — сообщение.")

def main():

    if len(sys.argv) != 4 or sys.argv[1] not in ("keygen","sign","verify"):

        usage()

        return

    cmd = sys.argv[1]

    in\_path = sys.argv[2]

    out\_path = sys.argv[3]

    if cmd == "keygen":

        params\_path = in\_path if os.path.exists(in\_path) else DEFAULT\_PARAMS\_FILE

        keygen(params\_path)

    elif cmd == "sign":

        sign\_file(in\_path, out\_path)

    else:

        verify\_file(in\_path, out\_path)

if \_\_name\_\_ == "\_\_main\_\_":

    main()