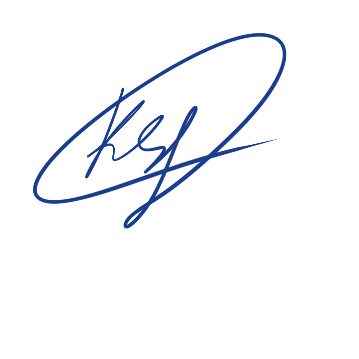
**Федеральное государственное бюджетное образовательное учреждение высшего образования "Белгородский государственный технологический университет им. В.Г. Шухова"**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем.

**Лабораторная работа №4**

Электронная подпись.

Выполнил:

Студент группы КБ-211

Коренев Д.Н.

Принял:

Смакаев А.В.

Оглавление

[Задание 3](#_Toc183129580)

[Разработанная программа 4](#_Toc183129581)

[Тестирование производительности 5](#_Toc183129582)

[Вывод 7](#_Toc183129583)

[Приложения 9](#_Toc183129584)

*Цель работы:* ознакомиться с принципами работы и алгоритмами, используемыми для создания электронной подписи. Разработать консольное приложение, позволяющее сгенерировать и проверить цифровую подпись для файла.

Задание

1. Разработать консольное приложение, осуществляющее основные этапы ЭП:

1. генерацию ключа подписи
2. подпись данных
3. проверку подписи

Приложение должно обладать функционалом выбора алгоритма ЭП из следующего списка:

* RSA-SHA256
* RSA-SHA512
* DSA
* ECDSA
* ГОСТ 34.10-2018 - опционально

2. Для каждого алгоритма измерить время, необходимое для формирования ключа, подписания и проверки подписи файла размером 2мб.

Результаты можно представить в виде таблицы или диаграммы.

Требования к консольному приложению:

Консольное приложение должно иметь три режима:

* генерация ключа подписи и ключа для проверки подписи
* подпись файла
* проверка подписи

Консольное приложение должно принимать на вход следующие аргументы:

* режим работы
* алгоритм для подписи
* имя/имена файлов для ключей
* имя файла для подписи/проверки подписи
* имя файла для результата

Разработанная программа

Код реализации программы в приложении 2.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Веб-сайт

Автоматически созданное описание  
Рисунок 1. Помощь про программе.

Генерация ключа для подписи:

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 2. Результат генерации ключа.

Подпись файла:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание  
Рисунок 3. Результат подписи файла.

Проверка подписи:

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание  
Рисунок 4. Результат проверки подписи на ориганале файла.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание  
Рисунок 4. Результат проверки подписи на измененном файле.

Тестирование производительности

Для тестирования использовался скрипт, описанный в приложении 3.

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание  
Рисунок 5. Время генерации ключа.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание  
Рисунок 6. Время подписи.

Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание  
Рисунок 7. Время проверки подписи.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание  
Рисунок 8. Время подписи (расширенный график).

Изображение выглядит как текст, линия, диаграмма, снимок экрана

Автоматически созданное описание  
Рисунок 9. Время проверки подписи (расширенный график).

Вывод

В ходе лабораторной работы мы ознакомились с основами создания и проверки электронной подписи (ЭП) и реализовали консольное приложение для выполнения этих задач. Программа поддерживает несколько алгоритмов, что позволяет сравнить их эффективность в зависимости от потребностей пользователя.

Приложения

Приложение 1. Ссылка на репозиторий реализцаии.

<https://github.com/Kseen715/crypto-io-lr/tree/main/lr4-digital-signature>

Приложение 2. Код реализации программы.

Python 3.12

import argparse

import sys

from pathlib import Path

from typing import Optional

from Crypto.Hash import SHA256, SHA512

from Crypto.PublicKey import RSA, DSA, ECC

from Crypto.Signature import pkcs1\_15, DSS

from Crypto.Random import get\_random\_bytes

from Crypto.IO import PEM

from gostcrypto import gostsignature, gosthash

import ksilorama

import GOST\_R\_34\_10\_2018

msg\_valid\_signature = \

    '[SIGNATURE] ' \

    + ksilorama.Fore.HEX('#22BB66') \

    + ksilorama.Style.ITALIC \

    + 'Signature is valid' \

    + ksilorama.Style.RESET\_ALL

msg\_invalid\_signature = \

    '[SIGNATURE] ' \

    + ksilorama.Fore.RED \

    + ksilorama.Style.BLINK \

    + ksilorama.Style.BRIGHT \

    + ksilorama.Style.INVERTED \

    + 'Invalid signature' \

    + ksilorama.Style.RESET\_ALL

def sign\_RSA\_SHA256(data: bytes, key: RSA.RsaKey) -> bytes:

    h = SHA256.new(data)

    return pkcs1\_15.new(key).sign(h)

def verify\_RSA\_SHA256(data: bytes, signature: bytes, key: RSA.RsaKey) -> bool:

    h = SHA256.new(data)

    try:

        pkcs1\_15.new(key).verify(h, signature)

        return True

    except (ValueError, TypeError):

        return False

def sign\_RSA\_SHA512(data: bytes, key: RSA.RsaKey) -> bytes:

    h = SHA512.new(data)

    return pkcs1\_15.new(key).sign(h)

def verify\_RSA\_SHA512(data: bytes, signature: bytes, key: RSA.RsaKey) -> bool:

    h = SHA512.new(data)

    try:

        pkcs1\_15.new(key).verify(h, signature)

        return True

    except (ValueError, TypeError):

        return False

def sign\_DSA(data: bytes, key: DSA.DsaKey) -> bytes:

    h = SHA256.new(data)

    return DSS.new(key, 'fips-186-3').sign(h)

def verify\_DSA(data: bytes, signature: bytes, key: DSA.DsaKey) -> bool:

    h = SHA256.new(data)

    try:

        DSS.new(key, 'fips-186-3').verify(h, signature)

        return True

    except (ValueError, TypeError):

        return False

def sign\_GOST\_34\_10\_2012\_SHA256(data: bytes, private\_key) -> bytes:

    sign\_obj = gostsignature.new(gostsignature.MODE\_256,

                                 gostsignature.CURVES\_R\_1323565\_1\_024\_2019['id-tc26-gost-3410-2012-256-paramSetB'])

    h = SHA256.new(data)

    return sign\_obj.sign(private\_key, h.digest())

def verify\_GOST\_34\_10\_2012\_SHA256(data: bytes, signature: bytes, public\_key) -> bool:

    sign\_obj = gostsignature.new(gostsignature.MODE\_256,

                                 gostsignature.CURVES\_R\_1323565\_1\_024\_2019['id-tc26-gost-3410-2012-256-paramSetB'])

    h = SHA256.new(data)

    try:

        return sign\_obj.verify(public\_key, h.digest(), signature)

    except (ValueError, TypeError):

        return False

def sign\_GOST\_34\_10\_2012\_SHA512(data: bytes, private\_key) -> bytes:

    sign\_obj = gostsignature.new(gostsignature.MODE\_512,

                                 gostsignature.CURVES\_R\_1323565\_1\_024\_2019['id-tc26-gost-3410-12-512-paramSetA'])

    h = SHA512.new(data)

    return sign\_obj.sign(private\_key, h.digest())

def verify\_GOST\_34\_10\_2012\_SHA512(data: bytes, signature: bytes, public\_key) -> bool:

    sign\_obj = gostsignature.new(gostsignature.MODE\_512,

                                 gostsignature.CURVES\_R\_1323565\_1\_024\_2019['id-tc26-gost-3410-12-512-paramSetA'])

    h = SHA512.new(data)

    try:

        return sign\_obj.verify(public\_key, h.digest(), signature)

    except (ValueError, TypeError):

        return False

def sign\_GOST\_34\_10\_2012\_STREEBOG256(data: bytes, private\_key) -> bytes:

    sign\_obj = gostsignature.new(gostsignature.MODE\_256,

                                 gostsignature.CURVES\_R\_1323565\_1\_024\_2019['id-tc26-gost-3410-2012-256-paramSetB'])

    h = gosthash.new('streebog256', data=data)

    return sign\_obj.sign(private\_key, h.digest())

def verify\_GOST\_34\_10\_2012\_STREEBOG256(data: bytes, signature: bytes, public\_key) -> bool:

    sign\_obj = gostsignature.new(gostsignature.MODE\_256,

                                 gostsignature.CURVES\_R\_1323565\_1\_024\_2019['id-tc26-gost-3410-2012-256-paramSetB'])

    h = gosthash.new('streebog256', data=data)

    try:

        return sign\_obj.verify(public\_key, h.digest(), signature)

    except (ValueError, TypeError):

        return False

def sign\_GOST\_34\_10\_2012\_STREEBOG512(data: bytes, private\_key) -> bytes:

    sign\_obj = gostsignature.new(gostsignature.MODE\_512,

                                 gostsignature.CURVES\_R\_1323565\_1\_024\_2019['id-tc26-gost-3410-12-512-paramSetA'])

    h = gosthash.new('streebog512', data=data)

    return sign\_obj.sign(private\_key, h.digest())

def verify\_GOST\_34\_10\_2012\_STREEBOG512(data: bytes, signature: bytes, public\_key) -> bool:

    sign\_obj = gostsignature.new(gostsignature.MODE\_512,

                                 gostsignature.CURVES\_R\_1323565\_1\_024\_2019['id-tc26-gost-3410-12-512-paramSetA'])

    h = gosthash.new('streebog512', data=data)

    try:

        return sign\_obj.verify(public\_key, h.digest(), signature)

    except (ValueError, TypeError):

        return False

def generate\_key(key\_path: Path, alg: str) -> None:

    if alg == 'RSA-SHA256':

        key = RSA.generate(2048)

        with Path(key\_path).open('wb') as f:

            f.write(key.export\_key())

    elif alg == 'RSA-SHA512':

        key = RSA.generate(4096)

        with Path(key\_path).open('wb') as f:

            f.write(key.export\_key())

    elif alg == 'DSA':

        key = DSA.generate(2048)

        with Path(key\_path).open('wb') as f:

            f.write(key.export\_key())

    elif alg == 'ECDSA':

        key = ECC.generate(curve='P-256')

        with Path(key\_path).open('wb') as f:

            f.write(key.export\_key(format='PEM').encode())

    elif alg == 'GOST 34.10-2012 (SHA256)':

        private\_key = get\_random\_bytes(32)

        # format keys in PEM and save to file

        private\_key\_pem = PEM.encode(private\_key, 'PRIVATE KEY')

        with Path(key\_path).open('wb') as f:

            f.write(private\_key\_pem.encode())

    elif alg == 'GOST 34.10-2012 (SHA512)':

        private\_key = get\_random\_bytes(64)

        # format keys in PEM and save to file

        private\_key\_pem = PEM.encode(private\_key, 'PRIVATE KEY')

        with Path(key\_path).open('wb') as f:

            f.write(private\_key\_pem.encode())

    elif alg == 'GOST 34.10-2012 (STREEBOG256)':

        private\_key = get\_random\_bytes(32)

        # format keys in PEM and save to file

        private\_key\_pem = PEM.encode(private\_key, 'PRIVATE KEY')

        with Path(key\_path).open('wb') as f:

            f.write(private\_key\_pem.encode())

    elif alg == 'GOST 34.10-2012 (STREEBOG512)':

        private\_key = get\_random\_bytes(64)

        # format keys in PEM and save to file

        private\_key\_pem = PEM.encode(private\_key, 'PRIVATE KEY')

        with Path(key\_path).open('wb') as f:

            f.write(private\_key\_pem.encode())

    elif alg == 'GOST 34.10-2018 (SHA256)':

        print('GOST 34.10-2018 (SHA256) key generation is not supported')

def sign\_file(file: Path, signature\_file: Path, key\_path: Path, alg: str) -> None:

    # keep key in the begining of sig file

    if alg == 'RSA-SHA256':

        key = RSA.import\_key(Path(key\_path).read\_bytes())

        with Path(file).open('rb') as f:

            data = f.read()

        signature = sign\_RSA\_SHA256(data, key)

        with Path(signature\_file).open('wb') as f:

            key\_data = key.publickey().export\_key()

            f.write(len(key\_data).to\_bytes(4, 'big'))

            f.write(key\_data)

            f.write(signature)

    elif alg == 'RSA-SHA512':

        key = RSA.import\_key(Path(key\_path).read\_bytes())

        with Path(file).open('rb') as f:

            data = f.read()

        signature = sign\_RSA\_SHA512(data, key)

        with Path(signature\_file).open('wb') as f:

            key\_data = key.publickey().export\_key()

            f.write(len(key\_data).to\_bytes(4, 'big'))

            f.write(key\_data)

            f.write(signature)

    elif alg == 'DSA':

        key = DSA.import\_key(Path(key\_path).read\_bytes())

        with Path(file).open('rb') as f:

            data = f.read()

        signature = sign\_DSA(data, key)

        with Path(signature\_file).open('wb') as f:

            key\_data = key.publickey().export\_key()

            f.write(len(key\_data).to\_bytes(4, 'big'))

            f.write(key\_data)

            f.write(signature)

    elif alg == 'ECDSA':

        key = ECC.import\_key(Path(key\_path).read\_bytes())

        with Path(file).open('rb') as f:

            data = f.read()

        h = SHA256.new(data)

        signer = DSS.new(key, 'fips-186-3')

        signature = signer.sign(h)

        with Path(signature\_file).open('wb') as f:

            key\_data = key.public\_key().export\_key(format='PEM')

            f.write(len(key\_data).to\_bytes(4, 'big'))

            f.write(key\_data.encode())

            f.write(signature)

    elif alg == 'GOST 34.10-2012 (SHA256)':

        private\_key = PEM.decode(Path(key\_path).read\_text())[0]

        with Path(file).open('rb') as f:

            data = f.read()

        signature = sign\_GOST\_34\_10\_2012\_SHA256(data, private\_key)

        sign\_obj = gostsignature.new(

            gostsignature.MODE\_256,

            gostsignature.CURVES\_R\_1323565\_1\_024\_2019['id-tc26-gost-3410-2012-256-paramSetB'])

        public\_key = sign\_obj.public\_key\_generate(private\_key)

        with Path(signature\_file).open('wb') as f:

            key\_data = public\_key

            f.write(len(key\_data).to\_bytes(4, 'big'))

            f.write(key\_data)

            f.write(signature)

    elif alg == 'GOST 34.10-2012 (SHA512)':

        private\_key = PEM.decode(Path(key\_path).read\_text())[0]

        with Path(file).open('rb') as f:

            data = f.read()

        signature = sign\_GOST\_34\_10\_2012\_SHA512(data, private\_key)

        sign\_obj = gostsignature.new(

            gostsignature.MODE\_512,

            gostsignature.CURVES\_R\_1323565\_1\_024\_2019['id-tc26-gost-3410-12-512-paramSetA'])

        public\_key = sign\_obj.public\_key\_generate(private\_key)

        with Path(signature\_file).open('wb') as f:

            key\_data = public\_key

            f.write(len(key\_data).to\_bytes(4, 'big'))

            f.write(key\_data)

            f.write(signature)

    elif alg == 'GOST 34.10-2012 (STREEBOG256)':

        private\_key = PEM.decode(Path(key\_path).read\_text())[0]

        with Path(file).open('rb') as f:

            data = f.read()

        signature = sign\_GOST\_34\_10\_2012\_STREEBOG256(data, private\_key)

        sign\_obj = gostsignature.new(

            gostsignature.MODE\_256,

            gostsignature.CURVES\_R\_1323565\_1\_024\_2019['id-tc26-gost-3410-2012-256-paramSetB'])

        public\_key = sign\_obj.public\_key\_generate(private\_key)

        with Path(signature\_file).open('wb') as f:

            key\_data = public\_key

            f.write(len(key\_data).to\_bytes(4, 'big'))

            f.write(key\_data)

            f.write(signature)

    elif alg == 'GOST 34.10-2012 (STREEBOG512)':

        private\_key = PEM.decode(Path(key\_path).read\_text())[0]

        with Path(file).open('rb') as f:

            data = f.read()

        signature = sign\_GOST\_34\_10\_2012\_STREEBOG512(data, private\_key)

        sign\_obj = gostsignature.new(

            gostsignature.MODE\_512,

            gostsignature.CURVES\_R\_1323565\_1\_024\_2019['id-tc26-gost-3410-12-512-paramSetA'])

        public\_key = sign\_obj.public\_key\_generate(private\_key)

        with Path(signature\_file).open('wb') as f:

            key\_data = public\_key

            f.write(len(key\_data).to\_bytes(4, 'big'))

            f.write(key\_data)

            f.write(signature)

    elif alg == 'GOST 34.10-2018 (SHA256)':

        GOST\_R\_34\_10\_2018.elgamal\_ecc\_sign(file, signature\_file)

def verify\_file(file: Path, signature\_file: Path, alg: str) -> bool:

    if alg == 'RSA-SHA256':

        with Path(signature\_file).open('rb') as f:

            key\_len = int.from\_bytes(f.read(4), 'big')

            public\_key = RSA.import\_key(f.read(key\_len))

            signature = f.read()

        with Path(file).open('rb') as f:

            data = f.read()

        if verify\_RSA\_SHA256(data, signature, public\_key):

            return True

        else:

            return False

    elif alg == 'RSA-SHA512':

        with Path(signature\_file).open('rb') as f:

            key\_len = int.from\_bytes(f.read(4), 'big')

            key = RSA.import\_key(f.read(key\_len))

            signature = f.read()

        with Path(file).open('rb') as f:

            data = f.read()

        if verify\_RSA\_SHA512(data, signature, key):

            return True

        else:

            return False

    elif alg == 'DSA':

        with Path(signature\_file).open('rb') as f:

            key\_len = int.from\_bytes(f.read(4), 'big')

            key = DSA.import\_key(f.read(key\_len))

            signature = f.read()

        with Path(file).open('rb') as f:

            data = f.read()

        if verify\_DSA(data, signature, key):

            return True

        else:

            return False

    elif alg == 'ECDSA':

        with Path(signature\_file).open('rb') as f:

            key\_len = int.from\_bytes(f.read(4), 'big')

            key = ECC.import\_key(f.read(key\_len))

            signature = f.read()

        with Path(file).open('rb') as f:

            data = f.read()

        h = SHA256.new(data)

        verifier = DSS.new(key, 'fips-186-3')

        try:

            verifier.verify(h, signature)

            return True

        except (ValueError, TypeError):

            return False

    elif alg == 'GOST 34.10-2012 (SHA256)':

        with Path(signature\_file).open('rb') as f:

            key\_len = int.from\_bytes(f.read(4), 'big')

            key = f.read(key\_len)

            signature = f.read()

        with Path(file).open('rb') as f:

            data = f.read()

        if verify\_GOST\_34\_10\_2012\_SHA256(data, signature, key):

            return True

        else:

            return False

    elif alg == 'GOST 34.10-2012 (SHA512)':

        with Path(signature\_file).open('rb') as f:

            key\_len = int.from\_bytes(f.read(4), 'big')

            key = f.read(key\_len)

            signature = f.read()

        with Path(file).open('rb') as f:

            data = f.read()

        if verify\_GOST\_34\_10\_2012\_SHA512(data, signature, key):

            return True

        else:

            return False

    elif alg == 'GOST 34.10-2012 (STREEBOG256)':

        with Path(signature\_file).open('rb') as f:

            key\_len = int.from\_bytes(f.read(4), 'big')

            key = f.read(key\_len)

            signature = f.read()

        with Path(file).open('rb') as f:

            data = f.read()

        if verify\_GOST\_34\_10\_2012\_STREEBOG256(data, signature, key):

            return True

        else:

            return False

    elif alg == 'GOST 34.10-2012 (STREEBOG512)':

        with Path(signature\_file).open('rb') as f:

            key\_len = int.from\_bytes(f.read(4), 'big')

            key = f.read(key\_len)

            signature = f.read()

        with Path(file).open('rb') as f:

            data = f.read()

        if verify\_GOST\_34\_10\_2012\_STREEBOG512(data, signature, key):

            return True

        else:

            return False

    elif alg == 'GOST 34.10-2018 (SHA256)':

        return GOST\_R\_34\_10\_2018.elgamal\_ecc\_verify(file, signature\_file)

algs = [

    'RSA-SHA256',

    'RSA-SHA512',

    'DSA',

    'ECDSA',

    'GOST 34.10-2012 (SHA256)',

    'GOST 34.10-2012 (SHA512)',

    'GOST 34.10-2012 (STREEBOG256)',

    'GOST 34.10-2012 (STREEBOG512)',

    'GOST 34.10-2018 (SHA256)',

]

if \_\_name\_\_ == '\_\_main\_\_':

    description = \

        ksilorama.Fore.HEX('#EE9944') \

        + ksilorama.Style.ITALIC \

        + f'Sign or verify a file using a digital signature' \

        + ksilorama.Style.RESET\_ALL

    parser = argparse.ArgumentParser(description=description)

    parser.add\_argument(

        'command', choices=['sign', 'verify', 'keygen'],

        help='Command to execute')

    parser.add\_argument('file', type=Path, help='File to sign', nargs='?')

    parser.add\_argument('signature', type=Path, help='Signature file')

    # key is optional for verification

    parser.add\_argument('key', type=Path, help='Key file', nargs='?')

    parser.add\_argument(

        '-a', '--alg', type=str,

        choices=algs, required=True,

        help='Algorithm to use')

    args = parser.parse\_args()

    try:

        if args.command == 'verify':

            res = verify\_file(args.file, args.signature, args.alg)

            if res:

                print(msg\_valid\_signature)

            else:

                print(msg\_invalid\_signature)

        elif args.command == 'sign':

            sign\_file(args.file, args.signature, args.key, args.alg)

            print(f'Signed {ksilorama.Style.UNDERLINE}{args.file}'

                  + f'{ksilorama.Style.RESET\_ALL} with '

                  + f'{ksilorama.Style.UNDERLINE}{args.alg}'

                  + f'{ksilorama.Style.RESET\_ALL} algorithm. Signature saved '

                  + f'to {ksilorama.Style.UNDERLINE}{args.signature}'

                  + f'{ksilorama.Style.RESET\_ALL}')

        elif args.command == 'keygen':

            generate\_key(args.signature, args.alg)

            print(f'Generated key for {ksilorama.Style.UNDERLINE}{args.alg}'

                  + f'{ksilorama.Style.RESET\_ALL} algorithm. Key saved to '

                  + f'{ksilorama.Style.UNDERLINE}{args.signature}'

                  + f'{ksilorama.Style.RESET\_ALL}')

    except Exception as e:

        print(e, file=sys.stderr)

        sys.exit()

Приложение 3. Код тестирования программы.

Python 3.12

import random

import os

import io

import sys

from sign import \*

def test\_file\_sign\_RSA\_SHA256():

    try:

        file\_path = 'temp/test\_file\_sign\_RSA\_SHA256.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_RSA\_SHA256.sig'

        key\_file\_path = 'temp/test\_file\_sign\_RSA\_SHA256.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'RSA-SHA256')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path, key\_file\_path, 'RSA-SHA256')

        # Verify the file

        assert verify\_file(file\_path, signature\_file\_path, 'RSA-SHA256')

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_RSA\_SHA256\_changed\_data():

    try:

        file\_path = 'temp/test\_file\_sign\_RSA\_SHA256\_changed\_data.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_RSA\_SHA256\_changed\_data.sig'

        key\_file\_path = 'temp/test\_file\_sign\_RSA\_SHA256\_changed\_data.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'RSA-SHA256')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path, key\_file\_path, 'RSA-SHA256')

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Verify the file

        assert (verify\_file(file\_path, signature\_file\_path, 'RSA-SHA256') == False)

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_RSA\_SHA512():

    try:

        file\_path = 'temp/test\_file\_sign\_RSA\_SHA512.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_RSA\_SHA512.sig'

        key\_file\_path = 'temp/test\_file\_sign\_RSA\_SHA512.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'RSA-SHA512')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path, key\_file\_path, 'RSA-SHA512')

        # Verify the file

        assert verify\_file(file\_path, signature\_file\_path, 'RSA-SHA512')

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_RSA\_SHA512\_changed\_data():

    try:

        file\_path = 'temp/test\_file\_sign\_RSA\_SHA512\_changed\_data.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_RSA\_SHA512\_changed\_data.sig'

        key\_file\_path = 'temp/test\_file\_sign\_RSA\_SHA512\_changed\_data.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'RSA-SHA512')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path, key\_file\_path, 'RSA-SHA512')

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Verify the file

        assert (verify\_file(file\_path, signature\_file\_path, 'RSA-SHA512') == False)

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_DSA():

    try:

        file\_path = 'temp/test\_file\_sign\_DSA.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_DSA.sig'

        key\_file\_path = 'temp/test\_file\_sign\_DSA.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'DSA')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path, key\_file\_path, 'DSA')

        # Verify the file

        assert verify\_file(file\_path, signature\_file\_path, 'DSA')

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_DSA\_changed\_data():

    try:

        file\_path = 'temp/test\_file\_sign\_DSA\_changed\_data.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_DSA\_changed\_data.sig'

        key\_file\_path = 'temp/test\_file\_sign\_DSA\_changed\_data.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'DSA')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path, key\_file\_path, 'DSA')

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Verify the file

        assert (verify\_file(file\_path, signature\_file\_path, 'DSA') == False)

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_ECDSA():

    try:

        file\_path = 'temp/test\_file\_sign\_ECDSA.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_ECDSA.sig'

        key\_file\_path = 'temp/test\_file\_sign\_ECDSA.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'ECDSA')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path, key\_file\_path, 'ECDSA')

        # Verify the file

        assert verify\_file(file\_path, signature\_file\_path, 'ECDSA')

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_ECDSA\_changed\_data():

    try:

        file\_path = 'temp/test\_file\_sign\_ECDSA\_changed\_data.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_ECDSA\_changed\_data.sig'

        key\_file\_path = 'temp/test\_file\_sign\_ECDSA\_changed\_data.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'ECDSA')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path, key\_file\_path, 'ECDSA')

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Verify the file

        assert (verify\_file(file\_path, signature\_file\_path, 'ECDSA') == False)

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_GOST\_34\_10\_2018\_SHA256():

    try:

        file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2018.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2018.sig'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Sign the file

        sign\_file(file\_path, signature\_file\_path, None, 'GOST 34.10-2018 (SHA256)')

        # Verify the file

        assert verify\_file(file\_path, signature\_file\_path, 'GOST 34.10-2018 (SHA256)')

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

def test\_file\_sign\_GOST\_34\_10\_2018\_SHA256\_changed\_data():

    try:

        file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2018\_SHA256\_changed\_data.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2018\_SHA256\_changed\_data.sig'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Sign the file

        sign\_file(file\_path, signature\_file\_path, None, 'GOST 34.10-2018 (SHA256)')

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Verify the file

        assert (verify\_file(file\_path, signature\_file\_path,

                'GOST 34.10-2018 (SHA256)') == False)

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

def test\_file\_sign\_GOST\_34\_10\_2018\_SHA256\_key\_not\_supported\_msg():

    try:

        file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2018\_SHA256\_key\_not\_supported\_msg.txt'

        key\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2018\_SHA256\_key\_not\_supported\_msg.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Create a StringIO object to capture the output

        captured\_output = io.StringIO()

        # Redirect stdout to the StringIO object

        sys.stdout = captured\_output

        # Generate a key

        generate\_key(key\_file\_path, 'GOST 34.10-2018 (SHA256)')

        # Reset stdout to its original value

        sys.stdout = sys.\_\_stdout\_\_

        # Get the captured output

        output = captured\_output.getvalue()

        # Verify the output

        assert output == 'GOST 34.10-2018 (SHA256) key generation is not supported\n'

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

def test\_file\_sign\_GOST\_34\_10\_2012\_SHA256():

    try:

        file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_SHA256.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_SHA256.sig'

        key\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_SHA256.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'GOST 34.10-2012 (SHA256)')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path,

                  key\_file\_path, 'GOST 34.10-2012 (SHA256)')

        # Verify the file

        assert verify\_file(file\_path, signature\_file\_path,

                           'GOST 34.10-2012 (SHA256)')

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_GOST\_34\_10\_2012\_SHA256\_changed\_data():

    try:

        file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_SHA256\_changed\_data.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_SHA256\_changed\_data.sig'

        key\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_SHA256\_changed\_data.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'GOST 34.10-2012 (SHA256)')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path,

                  key\_file\_path, 'GOST 34.10-2012 (SHA256)')

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Verify the file

        assert (verify\_file(file\_path, signature\_file\_path,

                'GOST 34.10-2012 (SHA256)') == False)

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG256():

    try:

        file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG256.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG256.sig'

        key\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG256.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'GOST 34.10-2012 (STREEBOG256)')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path,

                  key\_file\_path, 'GOST 34.10-2012 (STREEBOG256)')

        # Verify the file

        assert verify\_file(file\_path, signature\_file\_path,

                           'GOST 34.10-2012 (STREEBOG256)')

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG256\_changed\_data():

    try:

        file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG256\_changed\_data.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG256\_changed\_data.sig'

        key\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG256\_changed\_data.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'GOST 34.10-2012 (STREEBOG256)')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path,

                  key\_file\_path, 'GOST 34.10-2012 (STREEBOG256)')

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Verify the file

        assert (verify\_file(file\_path, signature\_file\_path,

                'GOST 34.10-2012 (STREEBOG256)') == False)

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG512():

    try:

        file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG512.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG512.sig'

        key\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG512.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'GOST 34.10-2012 (STREEBOG512)')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path,

                  key\_file\_path, 'GOST 34.10-2012 (STREEBOG512)')

        # Verify the file

        assert verify\_file(file\_path, signature\_file\_path,

                           'GOST 34.10-2012 (STREEBOG512)')

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG512\_changed\_data():

    try:

        file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG512\_changed\_data.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG512\_changed\_data.sig'

        key\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_STREEBOG512\_changed\_data.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'GOST 34.10-2012 (STREEBOG512)')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path,

                  key\_file\_path, 'GOST 34.10-2012 (STREEBOG512)')

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Verify the file

        assert (verify\_file(file\_path, signature\_file\_path,

                'GOST 34.10-2012 (STREEBOG512)') == False)

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

def test\_file\_sign\_GOST\_34\_10\_2012\_SHA512():

    try:

        file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_SHA512.txt'

        signature\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_SHA512.sig'

        key\_file\_path = 'temp/test\_file\_sign\_GOST\_34\_10\_2012\_SHA512.pem'

        # Ensure the temp directory exists

        os.makedirs('temp', exist\_ok=True)

        # Create a file with random bytes

        with open(file\_path, 'wb') as f:

            f.write(random.randbytes(1024))

        # Generate a key

        generate\_key(key\_file\_path, 'GOST 34.10-2012 (SHA512)')

        # Sign the file

        sign\_file(file\_path, signature\_file\_path,

                  key\_file\_path, 'GOST 34.10-2012 (SHA512)')

        # Verify the file

        assert verify\_file(file\_path, signature\_file\_path,

                           'GOST 34.10-2012 (SHA512)')

    finally:

        # Cleanup

        if os.path.exists(file\_path):

            os.remove(file\_path)

        if os.path.exists(signature\_file\_path):

            os.remove(signature\_file\_path)

        if os.path.exists(key\_file\_path):

            os.remove(key\_file\_path)

Приложение 4. Результат тестирования программы.

Изображение выглядит как текст, электроника, снимок экрана, компьютер

Автоматически созданное описание

Приложение 5. Код анализа статистики работы алгоритмов.

Python 3.12

from sign import \*

import csv

import argparse

import os

import timeit

from hashlib import sha256

import time

import multiprocessing

import random

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

CYN = ksilorama.Fore.CYAN

CLR = ksilorama.Fore.HEX('#FF6677') \

    + ksilorama.Style.BRIGHT \

RST = ksilorama.Style.RESET\_ALL

wo\_keygen = ['GOST 34.10-2018']

times = 1000

algs = [

    # 'RSA-SHA256',

    # 'RSA-SHA512',

    # 'DSA',

    # 'ECDSA',

    # 'GOST 34.10-2012 (SHA256)',

    # 'GOST 34.10-2012 (SHA512)',

    'GOST 34.10-2012 (STREEBOG256)',

    'GOST 34.10-2012 (STREEBOG512)',

    # 'GOST 34.10-2018 (SHA256)',

]

def test\_case(alg, in\_sizes):

    try:

        for i in range(times):

            sizes = list(in\_sizes)

            # grab PID

            pid = os.getpid()

            # get time in ns

            time\_ns = time.time\_ns()

            # get hash of PID and time

            hash = sha256(f'{pid}{time\_ns}'.encode()).hexdigest()

            while sizes:

                size = random.choice(sizes)

                sizes.remove(size)

                try:

                    folder\_txt = Path('temp/txt')

                    folder\_sig = Path('temp/sig')

                    folder\_key = Path('temp/key')

                    file = Path(f'{folder\_txt}/file\_{size}\_{hash}.txt')

                    signature\_file = Path(

                        f'{folder\_sig}/signature\_{size}\_{hash}.sig')

                    key\_file = Path(f'{folder\_key}/key\_{size}\_{hash}.key')

                    if not os.path.exists(folder\_txt):

                        os.makedirs(folder\_txt)

                    if not os.path.exists(folder\_sig):

                        os.makedirs(folder\_sig)

                    if not os.path.exists(folder\_key):

                        os.makedirs(folder\_key)

                    with file.open('wb') as f:

                        f.write(os.urandom(int(size \* 1024 \* 1024)))

                    if alg not in wo\_keygen:

                        keygen\_time = timeit.timeit(

                            lambda: generate\_key(key\_file, alg), number=1)

                    else:

                        keygen\_time = 'inf'

                    sign\_time = timeit.timeit(

                        lambda: sign\_file(file, signature\_file, key\_file, alg),

                        number=1)

                    verify\_time = timeit.timeit(

                        lambda: verify\_file(file, signature\_file, alg),

                        number=1)

                    with open('temp/benchmark.csv', 'a') as res\_file:

                        res\_file.write(f'"{alg}",{size},{keygen\_time},{

                            sign\_time},{verify\_time}\n')

                    print(f'[{CYN}BENCH{RST}] '

                          + f'{CLR}Alg{RST}: {alg}, '

                          + f'{size} MB, '

                          + f'{CLR}Key{RST}: {float(keygen\_time):.6f}s, '

                          + f'{CLR}Sign{RST}: {sign\_time:.6f}s, '

                          + f'{CLR}Verify{RST}: {verify\_time:.6f}s', end='\n')

                finally:

                    if os.path.exists(file):

                        os.remove(file)

                    if os.path.exists(signature\_file):

                        os.remove(signature\_file)

                    if os.path.exists(key\_file) and alg not in wo\_keygen:

                        os.remove(key\_file)

    except KeyboardInterrupt:

        print(f'\n{CYN}Benchmarking interrupted{RST}')

        exit(1)

    except Exception as e:

        print(f'[{CYN}BENCH{RST}] {CLR}Error{RST}: {e}', end='\n')

def benchmark():

    sizes = np.arange(0.5, 8 + 0.1, 0.5)

    # csv format:

    # alg, size, keygen\_time, sign\_time, verify\_time

    if not os.path.exists('temp/benchmark.csv'):

        with open('temp/benchmark.csv', 'w') as f:

            f.write('alg,size,keygen\_time,sign\_time,verify\_time\n')

    try:

        # use multiprocessing to run tests in parallel

        with multiprocessing.Pool(processes=24) as pool:

            pool.starmap(test\_case, [(alg, sizes)

                         for alg in algs])

    except KeyboardInterrupt:

        print(f'\n{CYN}Benchmarking interrupted{RST}')

        exit(1)

class StatPlotter():

    color = [

        '#e41a1c',

        '#377eb8',

        '#f781bf',

        '#dede00',

        '#4daf4a',

        '#ff7f00',

        '#a65628',

        '#984ea3',

        '#999999',

    ]

    @staticmethod

    def \_remove\_outliers(data: pd.DataFrame, m=2):

        """

        Remove outliers from data

        :param data: data to remove outliers from

        :param m: number of standard deviations to consider as outlier

        :return: data without outliers

        """

        return data[abs(data - data.mean()) < m \* data.std()]

    @staticmethod

    def \_dot(v, w):

        """

        Скалярное произведение векторов

        Parameters

        ----------

            v (list): Вектор

            w (list): Вектор

        Returns

        -------

            float: Скалярное произведение векторов

        """

        if type(v) != list:

            raise TypeError("v should be a list, not " + str(type(v)) + ".")

        if type(w) != list:

            raise TypeError("w should be a list, not " + str(type(w)) + ".")

        if len(v) != len(w):

            raise ValueError("vectors should be the same length "

                             "(v: " + str(len(v)) + ", w: " + str(len(w)) + ").")

        if len(v) == 0 or len(w) == 0:

            raise ValueError("vectors should be non-empty.")

        if type(v[0]) != int and type(v[0]) != float:

            raise TypeError("v should contain numbers, "

                            "not " + str(type(v[0])) + ".")

        if type(w[0]) != int and type(w[0]) != float:

            raise TypeError("w should contain numbers, "

                            "not " + str(type(w[0])) + ".")

        return sum(v\_i \* w\_i for v\_i, w\_i in zip(v, w))

    @staticmethod

    def \_gauss\_slae(A, b):

        """

        Метод Гаусса решения СЛАУ

        Parameters

        ----------

            A (list of list): Матрица коэффициентов

            b (list): Свободные члены

        Returns

        -------

            list: Решение

        """

        n = len(b)  # вычисляем порядок системы

        # строим расширенную матрицу системы

        G = [ai+[bi] for ai, bi in zip(A, b)]

        # Прямой проход

        for i in range(n):

            for j in range(i, n):

                G[j] = list(map(lambda x: x/G[j][i], G[j]))

                if j > i:

                    G[j] = [g - u for g, u in zip(G[j], G[i])]

        # Обратный проход

        x = [0]\*n      # инициируем список, который потом станет решением

        for i in range(n-1, -1, -1):

            x[i] = G[i][-1]-StatPlotter.\_dot(x, G[i][:-1])

        return x

    @staticmethod

    def \_approx\_poly(x, t, r):

        """

        Аппроксимация полиномом

        Parameters

        ----------

            x (list): Список чисел

            t (list): Список чисел, range(1, len(x)+1)

            r (int): Степень полинома

        Returns

        -------

            list: Параметры полинома

        """

        M = [[] for \_ in range(r+1)]

        b = []

        for l in range(r+1):

            for q in range(r+1):

                M[l].append(sum(list(map(lambda z: z\*\*(l+q), t))))

            b.append(sum(xi\*ti\*\*l for xi, ti in zip(x, t)))

        a = StatPlotter.\_gauss\_slae(M, b)

        return a

    @staticmethod

    def plot\_lines(

            df: pd.DataFrame,

            xcolumn: str,

            ycolumn: str,

            column\_with\_line\_name: str,

            groupby: list,

            title: str = 'Title',

            xlabel: str = 'X axis',

            ylabel: str = 'Y asix',

            exclude\_line\_name: list = [],

            m: float = 1e9999,

            output\_folder: str = 'temp',

            file\_postfix: str = '',

    ):

        """

        Plot line plot with error bars

        :param df: data to plot

        :param xcolumn: x axis column

        :param ycolumn: y axis column

        :param column\_with\_line\_name: column with line name (e.g. 'alg')

        :param groupby: columns to group by (e.g. ['alg', 'size'])

        :param title: plot title

        :param xlabel: x axis label

        :param ylabel: y axis label

        :param m: number of standard deviations to consider as outlier

        (default: 1e9999)

        :param output\_folder: output folder for plot (default: 'temp')

        """

        # exclude alg from exclude\_line\_name

        df = df[~df[column\_with\_line\_name].isin(exclude\_line\_name)]

        # remove outliers

        df = df.groupby(groupby).apply(StatPlotter.\_remove\_outliers, m)

        # calculate average time for every alg for every size by iteration

        means = df.groupby(groupby).mean()

        # calculate yerr for every alg

        mins = df.groupby(groupby).min()

        maxs = df.groupby(groupby).max()

        # combine data

        means = means.reset\_index()

        mins = mins.reset\_index()

        maxs = maxs.reset\_index()

        column\_max = ycolumn + '\_max'

        column\_min = ycolumn + '\_min'

        column\_mean = ycolumn + '\_mean'

        data = pd.merge(means, mins, on=groupby, suffixes=('\_mean', '\_min'))

        data = pd.merge(data, maxs, on=groupby)

        data = data.rename(columns={ycolumn: column\_max})

        # calculate linear regression for every alg

        def gen\_poly\_data(x, P):

            return [sum([P[i] \* x \*\* i for i in range(len(P))]) for x in x]

        def gen\_poly\_str(P):

            terms = []

            for i in range(len(P)-1, 0, -1):

                if i == len(P)-1:

                    if i == 1:

                        terms.append(f'{P[i]:.1f}x' if P[i] != 0 else '')

                    else:

                        terms.append(f'{P[i]:.1f}x^{i}' if P[i] != 0 else '')

                else:

                    if i == 1:

                        terms.append(f'{P[i]:+.1f}x' if P[i] != 0 else '')

                    else:

                        terms.append(f'{P[i]:+.1f}x^{i}' if P[i] != 0 else '')

            return ''.join(filter(None, terms)) \

                + (f'{P[0]:+.1f}' if P[0] != 0 else '')

        for alg in data[column\_with\_line\_name].unique():

            alg\_data = data[data[column\_with\_line\_name] == alg]

            x = alg\_data['size']

            y = alg\_data[column\_mean]

            m = StatPlotter.\_approx\_poly(y.tolist(), x.tolist(), 1)

            data.loc[data[column\_with\_line\_name] == alg, 'poly'] \

                = gen\_poly\_str(m)

            data.loc[data[column\_with\_line\_name] == alg, 'poly\_data'] \

                = gen\_poly\_data(x, m)

        print(data)

        percent\_to\_plot = 100

        plot\_lim = int(100 / percent\_to\_plot)

        fig, ax = plt.subplots(figsize=(10, 6))

        for alg in data[column\_with\_line\_name].unique():

            alg\_data = data[data[column\_with\_line\_name] == alg]

            ax.errorbar(

                alg\_data[xcolumn][::plot\_lim],

                alg\_data[column\_mean][::plot\_lim],

                yerr=[(alg\_data[column\_mean]

                       - alg\_data[column\_min])[::plot\_lim],

                      (alg\_data[column\_max]

                       - alg\_data[column\_mean])[::plot\_lim]],

                label=alg,

                fmt='-o',

                color=StatPlotter.color[data[column\_with\_line\_name].unique(

                ).tolist().index(alg)],

                capsize=4,

                capthick=1.5,

            )

            alg\_data = data[data[column\_with\_line\_name] == alg]

            ax.plot(

                alg\_data[xcolumn][::plot\_lim],

                alg\_data['poly\_data'][::plot\_lim],

                label=f'{alg} poly {alg\_data["poly"].iloc[0]}',

                linestyle='--',

                color=StatPlotter.color[data[column\_with\_line\_name].unique(

                ).tolist().index(alg)],

            )

        ax.set\_xlabel(xlabel)

        ax.set\_ylabel(ylabel)

        ax.legend()

        plt.title(title)

        plt.tight\_layout()

        plt.savefig(f'{output\_folder}/plt\_{ycolumn}{file\_postfix}.png')

    @staticmethod

    def plot\_bars(

            df: pd.DataFrame,

            xcolumn: str,

            ycolumn: str,

            column\_with\_line\_name: str,

            groupby: list,

            title: str = 'Title',

            xlabel: str = 'X axis',

            ylabel: str = 'Y asix',

            m: float = 1e9999,

            exclude\_bar\_name: list = [],

            xlabel\_rotation: int = 0,

            output\_folder: str = 'temp',

            file\_postfix: str = '',

    ):

        """

        Plot bar plot with error bars

        :param df: data to plot

        :param xcolumn: x axis column

        :param ycolumn: y axis column

        :param column\_with\_line\_name: column with line name (e.g. 'alg')

        :param groupby: columns to group by (e.g. ['alg', 'size'])

        :param title: plot title

        :param xlabel: x axis label

        :param ylabel: y axis label

        :param m: number of standard deviations to consider as outlier

        (default: 1e9999)

        :param output\_folder: output folder for plot (default: 'temp')

        """

        # exclude alg from exclude\_line\_name

        df = df[~df[column\_with\_line\_name].isin(exclude\_bar\_name)]

        # remove outliers

        df = df.groupby(groupby).apply(StatPlotter.\_remove\_outliers, m)

        # calculate average time for every alg for every size by iteration

        means = df.groupby(groupby).mean()

        # calculate yerr for every alg

        mins = df.groupby(groupby).min()

        maxs = df.groupby(groupby).max()

        # combine data

        means = means.reset\_index()

        mins = mins.reset\_index()

        maxs = maxs.reset\_index()

        time\_name\_max = ycolumn + '\_max'

        time\_name\_min = ycolumn + '\_min'

        time\_name\_mean = ycolumn + '\_mean'

        data = pd.merge(means, mins, on=groupby, suffixes=('\_mean', '\_min'))

        data = pd.merge(data, maxs, on=groupby)

        data = data.rename(columns={ycolumn: time\_name\_max})

        print(data)

        # plot bar plot with error bars

        fig, ax = plt.subplots(figsize=(10, 6))

        alg\_data = data

        ax.bar(

            alg\_data[xcolumn],

            alg\_data[time\_name\_mean],

            yerr=[(alg\_data[time\_name\_mean]

                   - alg\_data[time\_name\_min]),

                  (alg\_data[time\_name\_max]

                   - alg\_data[time\_name\_mean])],

            color=StatPlotter.color,

            error\_kw=dict(lw=1, capsize=5, capthick=2),

        )

        # add number on bottom of bars

        for i, v in enumerate(alg\_data[time\_name\_mean]):

            ax.text(i, v, f'{v:.2f}', ha='center', va='bottom')

        ax.set\_xlabel(xlabel)

        ax.set\_ylabel(ylabel)

        # add title

        plt.title(title)

        plt.xticks(rotation=xlabel\_rotation)

        # expand fig to fit labels

        plt.tight\_layout()

        plt.savefig(f'{output\_folder}/plt\_{ycolumn}{file\_postfix}.png')

def plot():

    #     alg, size, keygen\_time, sign\_time, verify\_time

    time\_names\_line\_plot = ['sign\_time', 'verify\_time']

    time\_names\_bar\_plot = ['keygen\_time']

    for time\_name in time\_names\_line\_plot:

        df = pd.read\_csv('temp/benchmark.csv')

        other\_time = (time\_names\_line\_plot.copy() + time\_names\_bar\_plot.copy())

        other\_time.remove(time\_name)

        # Drop other time columns

        df = df.drop(columns=other\_time)

        # convert time from s to ms

        df[time\_name] = df[time\_name] \* 1000

        StatPlotter.plot\_lines(

            df,

            'size',

            time\_name,

            'alg',

            ['alg', 'size'],

            f'{str(time\_name).capitalize().replace(

                '\_', ' ')} with different algorithms',

            'File size (MB)', 'Time (ms)',

            output\_folder='temp',

            m=0.5,

            exclude\_line\_name=[

                # 'GOST 34.10-2012 (SHA512)',

                'GOST 34.10-2012 (STREEBOG256)',

                'GOST 34.10-2012 (STREEBOG512)',

            ],

            file\_postfix='\_smaller',

        )

    for time\_name in time\_names\_line\_plot:

        df = pd.read\_csv('temp/benchmark.csv')

        other\_time = (time\_names\_line\_plot.copy() + time\_names\_bar\_plot.copy())

        other\_time.remove(time\_name)

        # Drop other time columns

        df = df.drop(columns=other\_time)

        # convert time from s to ms

        df[time\_name] = df[time\_name] \* 1000

        StatPlotter.plot\_lines(

            df,

            'size',

            time\_name,

            'alg',

            ['alg', 'size'],

            f'{str(time\_name).capitalize().replace(

                '\_', ' ')} with different algorithms',

            'File size (MB)', 'Time (ms)',

            output\_folder='temp',

            m=1e9999,

            file\_postfix='\_full',

        )

    for time\_name in time\_names\_bar\_plot:

        df = pd.read\_csv('temp/benchmark.csv')

        # Drop size column

        df = df.drop(columns=['size'])

        other\_time = (time\_names\_line\_plot.copy() + time\_names\_bar\_plot.copy())

        other\_time.remove(time\_name)

        # Drop other time columns

        df = df.drop(columns=other\_time)

        # convert time from s to ms

        df[time\_name] = df[time\_name] \* 1000

        StatPlotter.plot\_bars(

            df,

            'alg',

            time\_name,

            'alg',

            ['alg'],

            f'{str(time\_name).capitalize().replace(

                '\_', ' ')} with different algorithms',

            'Algorithm', 'Time (ms)',

            output\_folder='temp',

            m=0.5,

            xlabel\_rotation=55,

            file\_postfix='',

            exclude\_bar\_name=[

                'GOST 34.10-2018 (SHA256)',

            ],

        )

if \_\_name\_\_ == '\_\_main\_\_':

    description = \

        ksilorama.Fore.HEX('#EE9944') \

        + ksilorama.Style.ITALIC \

        + 'Benchmark and plot the results' \

        + ksilorama.Style.RESET\_ALL

    parser = argparse.ArgumentParser(

        description=description)

    parser.add\_argument('command', choices=['bench', 'plot'])

    args = parser.parse\_args()

    if args.command == 'bench':

        try:

            benchmark()

        except KeyboardInterrupt:

            print(f'\n{CYN}Benchmarking interrupted{RST}')

            exit(1)

    elif args.command == 'plot':

        try:

            plot()

        except KeyboardInterrupt:

            print(f'\n{CYN}Plotting interrupted{RST}')

            exit(1)

    else:

        print('Invalid command')

        exit(1)