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.

Белгород 2024

Оглавление

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:

а) генерацию ключа подписи

б) подпись данных

в) 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.

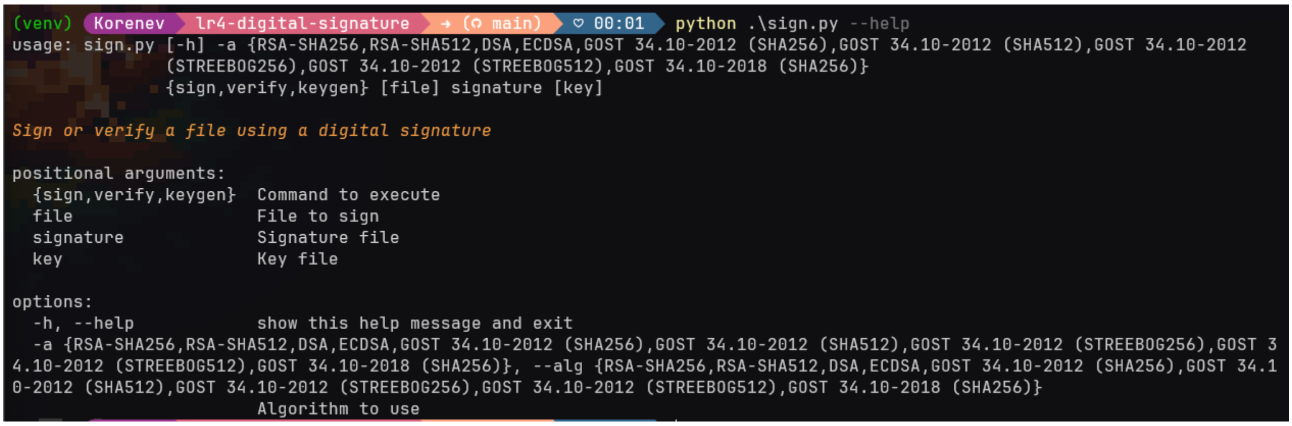


Рисунок 1. Помощь про программе.

Генерация ключа для подписи:

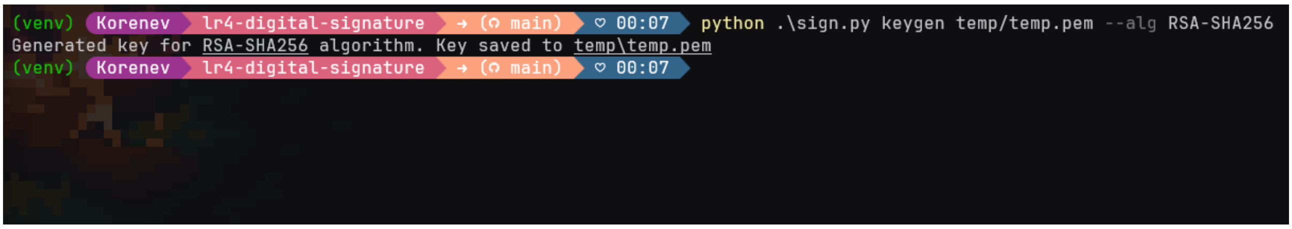


Рисунок 2. Результат генерации ключа.

Подпись файла:

Рисунок 3. Результат подписи файла.

Проверка подписи:

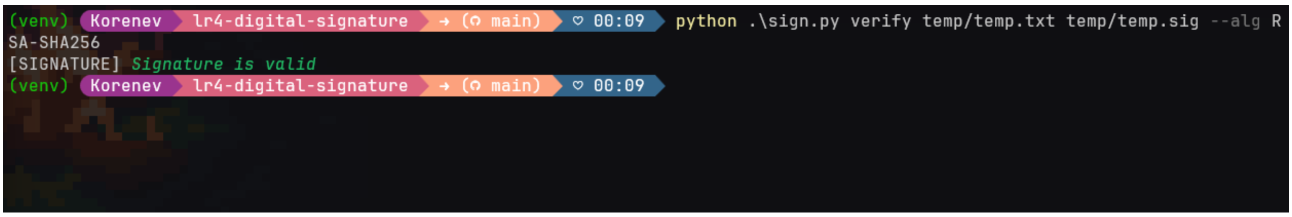


Рисунок 4. Результат проверки подписи на ориганале файла.

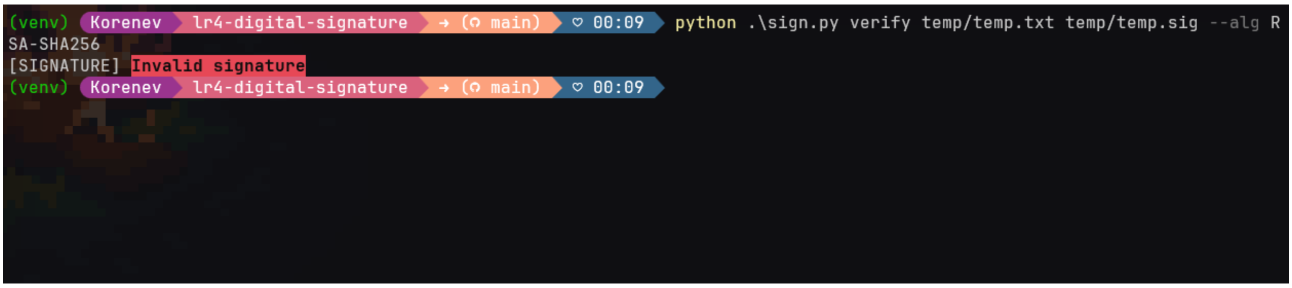


Рисунок 4. Результат проверки подписи на измененном файле.

Тестирование производительности

Для тестирования использовался скрипт, описанный в приложении 3.

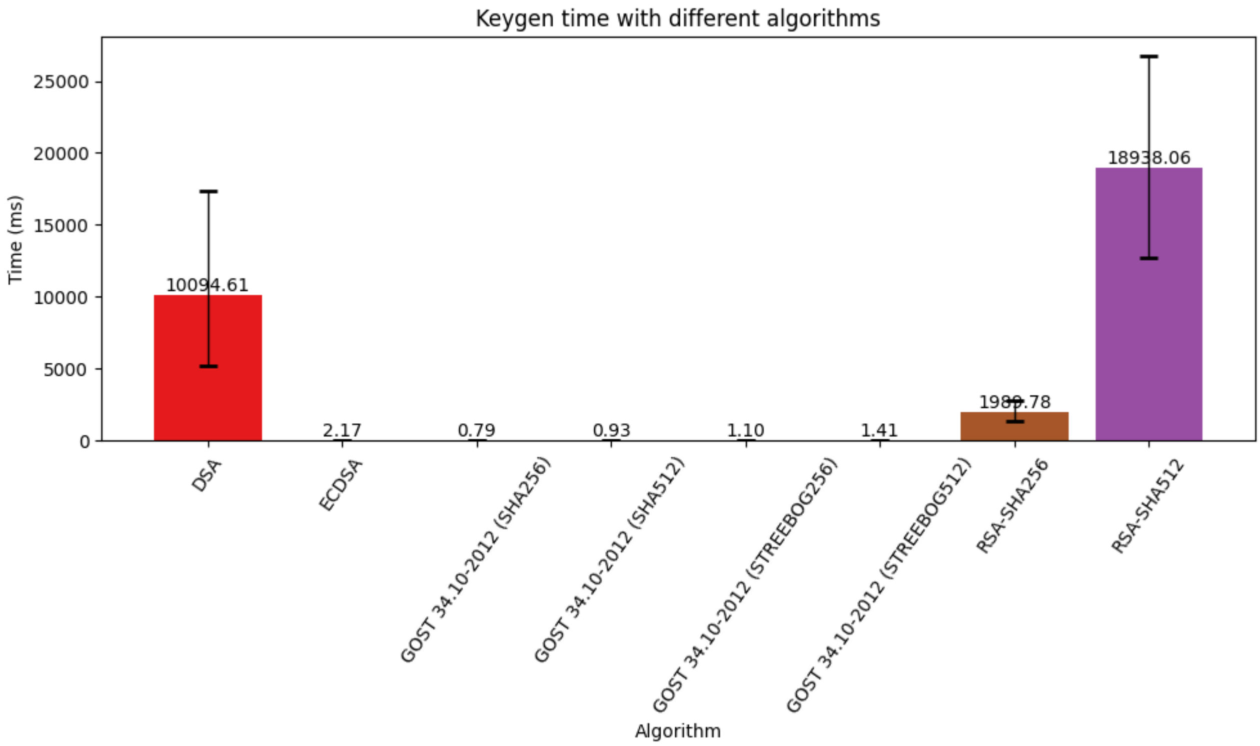


Рисунок 5. Время генерации ключа.

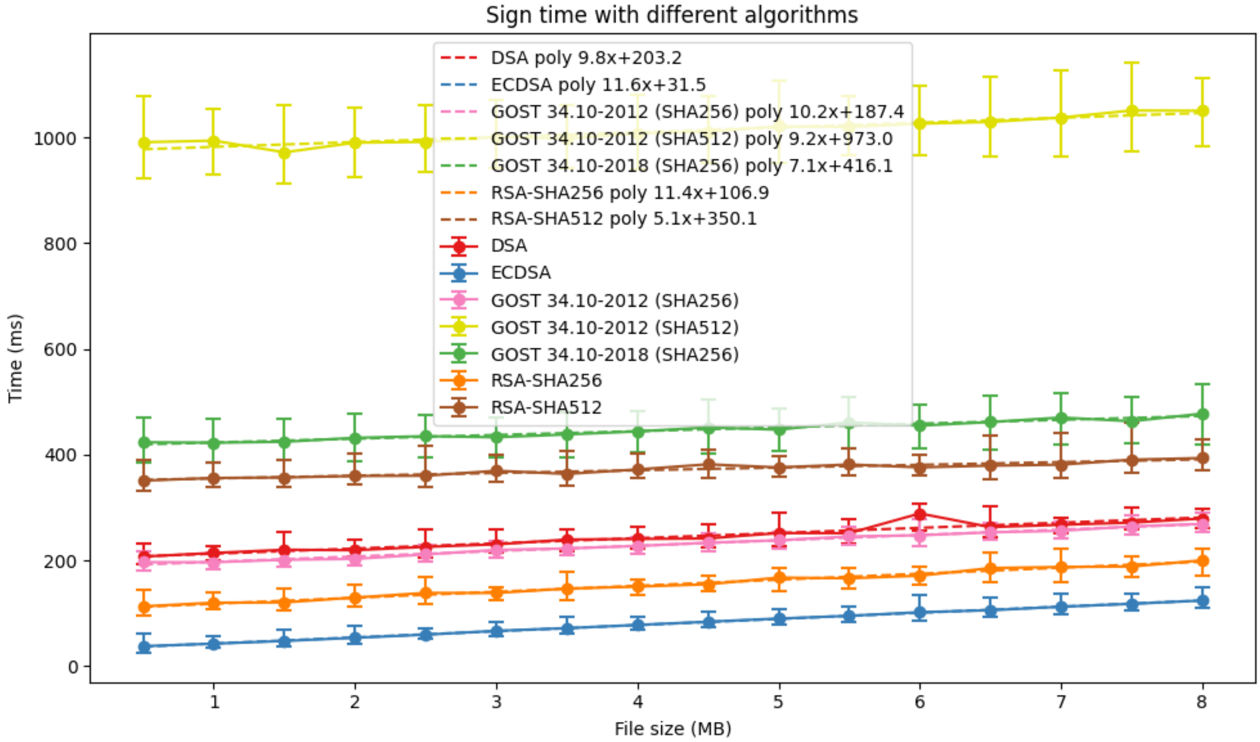


Рисунок 6. Время подписи.

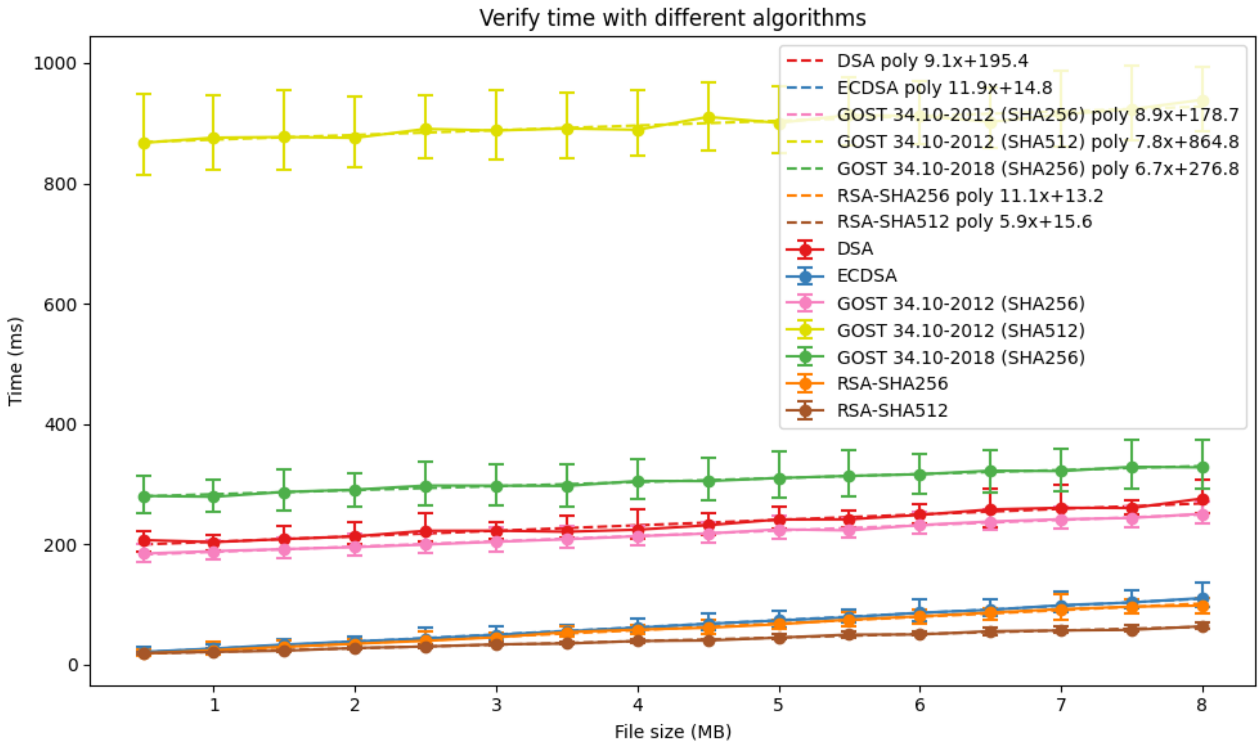


Рисунок 7. Время проверки подписи.

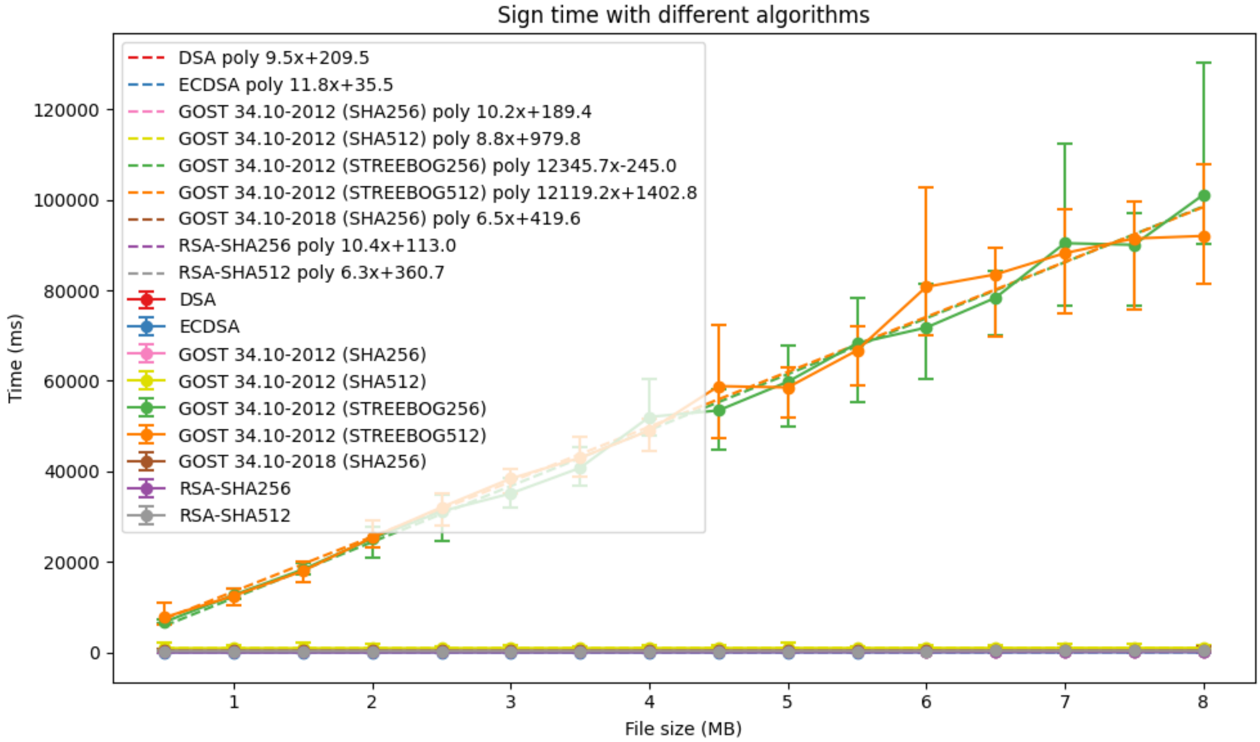


Рисунок 8. Время подписи (расширенный график).

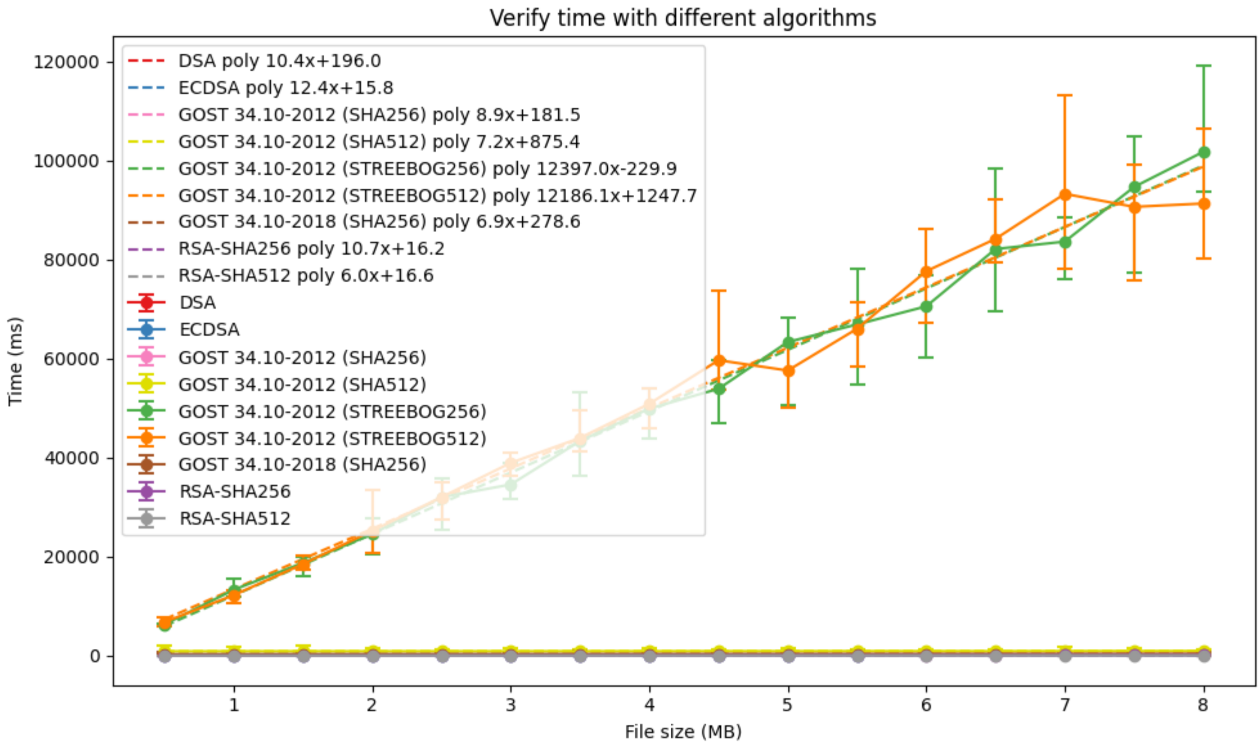


Рисунок 9. Время проверки подписи (расширенный график).

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)':
```

```python
        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
```

```python
        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'
```

```python
        # 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'
```

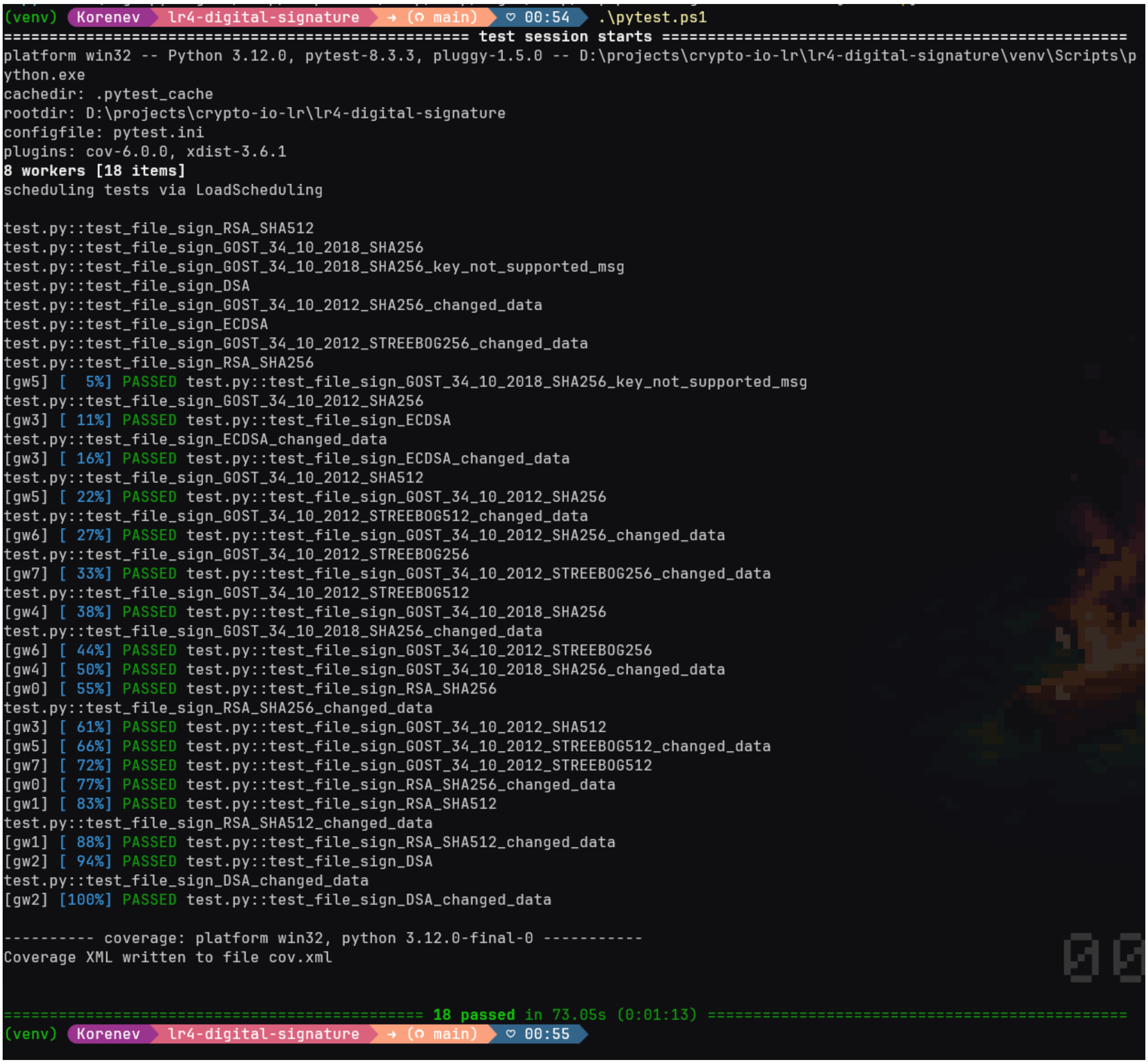
```
        signature_file_path =
'temp/test_file_sign_GOST_34_10_2012_SHA256_changed_data.sig'
        key_file_path = 'temp/test_file_sign_GOST_34_10_2012_SHA256_changed_data.pem'

        # 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.



```
(venv) Korenev  lr4-digital-signature  → (main)  ♡ 00:54  .\pytest.ps1
==================================================== test session starts ====================================================
platform win32 -- Python 3.12.0, pytest-8.3.3, pluggy-1.5.0 -- D:\projects\crypto-io-lr\lr4-digital-signature\venv\Scripts\p
ython.exe
cachedir: .pytest_cache
rootdir: D:\projects\crypto-io-lr\lr4-digital-signature
configfile: pytest.ini
plugins: cov-6.0.0, xdist-3.6.1
8 workers [18 items]
scheduling tests via LoadScheduling

test.py::test_file_sign_RSA_SHA512
test.py::test_file_sign_GOST_34_10_2018_SHA256
test.py::test_file_sign_GOST_34_10_2018_SHA256_key_not_supported_msg
test.py::test_file_sign_DSA
test.py::test_file_sign_GOST_34_10_2012_SHA256_changed_data
test.py::test_file_sign_ECDSA
test.py::test_file_sign_GOST_34_10_2012_STREEBOG256_changed_data
test.py::test_file_sign_RSA_SHA256
[gw5] [  5%] PASSED test.py::test_file_sign_GOST_34_10_2018_SHA256_key_not_supported_msg
test.py::test_file_sign_GOST_34_10_2012_SHA256
[gw3] [ 11%] PASSED test.py::test_file_sign_ECDSA
test.py::test_file_sign_ECDSA_changed_data
[gw3] [ 16%] PASSED test.py::test_file_sign_ECDSA_changed_data
test.py::test_file_sign_GOST_34_10_2012_SHA512
[gw5] [ 22%] PASSED test.py::test_file_sign_GOST_34_10_2012_SHA256
test.py::test_file_sign_GOST_34_10_2012_STREEBOG512_changed_data
[gw6] [ 27%] PASSED test.py::test_file_sign_GOST_34_10_2012_SHA256_changed_data
test.py::test_file_sign_GOST_34_10_2012_STREEBOG256
[gw7] [ 33%] PASSED test.py::test_file_sign_GOST_34_10_2012_STREEBOG256_changed_data
test.py::test_file_sign_GOST_34_10_2012_STREEBOG512
[gw4] [ 38%] PASSED test.py::test_file_sign_GOST_34_10_2018_SHA256
test.py::test_file_sign_GOST_34_10_2018_SHA256_changed_data
[gw6] [ 44%] PASSED test.py::test_file_sign_GOST_34_10_2012_STREEBOG256
[gw4] [ 50%] PASSED test.py::test_file_sign_GOST_34_10_2018_SHA256_changed_data
[gw0] [ 55%] PASSED test.py::test_file_sign_RSA_SHA256
test.py::test_file_sign_RSA_SHA256_changed_data
[gw3] [ 61%] PASSED test.py::test_file_sign_GOST_34_10_2012_SHA512
[gw5] [ 66%] PASSED test.py::test_file_sign_GOST_34_10_2012_STREEBOG512_changed_data
[gw7] [ 72%] PASSED test.py::test_file_sign_GOST_34_10_2012_STREEBOG512
[gw0] [ 77%] PASSED test.py::test_file_sign_RSA_SHA256_changed_data
[gw1] [ 83%] PASSED test.py::test_file_sign_RSA_SHA512
test.py::test_file_sign_RSA_SHA512_changed_data
[gw1] [ 88%] PASSED test.py::test_file_sign_RSA_SHA512_changed_data
[gw2] [ 94%] PASSED test.py::test_file_sign_DSA
test.py::test_file_sign_DSA_changed_data
[gw2] [100%] PASSED test.py::test_file_sign_DSA_changed_data

---------- coverage: platform win32, python 3.12.0-final-0 -----------
Coverage XML written to file cov.xml


=========================================== 18 passed in 73.05s (0:01:13) ============================================
(venv) Korenev  lr4-digital-signature  → (main)  ♡ 00:55
```

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
```

```python
    @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
```

```python
        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)
```