Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

**Лабораторная работа № 12**

«ИССЛЕДОВАНИЕ АЛГОРИТМОВ ГЕНЕРАЦИИ И ВЕРИФИКАЦИИ ЭЛЕКТРОННОЙ ЦИФРОВОЙ ПОДПИСИ»

Выполнил:

Студент: Лысков К.Е.

ФИТ 3 курс 4 группа

Преподаватель: Сазонова Д.В.

Минск 2023

1. **Описание приложения**

Приложение написано на языке программирования Python и позволяет провести генерацию и верификацию ЭЦП на основе алгоритмов RSA, Эль-Гамаля и Шнорра.

1. **Методика выполнения расчетов**

В данной лабораторной работе была поставлена цель создания приложения, которое позволяет провести генерацию и верификацию ЭЦП на основе алгоритмов RSA, Эль-Гамаля и Шнорра. На листинге 2.1 представлены функции, реализующие данную функциональность.

def EDSBasedOnRsa(message: str, p: int, q: int):

*# Подготовка*

    verification\_time = time.process\_time()

    n = p \* q

    euler\_function = (p - 1) \* (q - 1)

    e = GetFirstMutuallyPrimeNumber(euler\_function)

    d = InverseNumber(e, euler\_function)[1] % euler\_function

    print('--- Подготовка ---\n'

          f'p = {p}\n'

          f'q = {q}\n'

          f'n = {n}\n'

          f'euler\_function = {euler\_function}\n'

          f'e = {e}\n'

          f'd = {d}')

*# Отправитель*

    hash\_message\_sender = hash(message) % n

    encrypt\_hash = FastPower(hash\_message\_sender, e, n)

    print(f'--- Отправитель ---\n'

          f'H(message) = {hash\_message\_sender}\n'

          f'S(H(message)) = {encrypt\_hash}\n'

          f'Отправлено {encrypt\_hash} и идентификатор отправителя')

*# Получатель*

    hash\_message\_recipient = hash(message) % n

    decrypt\_hash = FastPower(encrypt\_hash, d, n)

    comparison\_result = decrypt\_hash == hash\_message\_recipient

    print(f'--- Получатель ---\n'

          f'H(message) = {hash\_message\_recipient}\n'

          f'D(S(H(message)))) = {decrypt\_hash}\n'

          f'Подпись {"верифицирована!" if comparison\_result else "не верифицирована!"}')

    verification\_time = time.process\_time() - verification\_time

    print(f'--- Анализ ---\n'

          f'Время верификации: {verification\_time}')

def EDSBasedElGamal(message: str, p: int):

*# Подготовка*

    verification\_time = time.process\_time()

    g = GeneratePrimitiveRoot(p)

    x = int(random.random() \* p)

    y = FastPower(g, x, p)

    print(f'--- Подготовка ---\n'

          f'p = {p}\n'

          f'g = {g}\n'

          f'x = {x}\n'

          f'y = {y}')

*# Отправитель*

    k = GetFirstMutuallyPrimeNumber(p - 1)

    hash\_message\_sender = hash(message) % p

    a = FastPower(g, k, p)

    b = GetBByKAndP(k, p, x, a, hash\_message\_sender)

    print(f'--- Отправитель ---\n'

          f'k = {k}\n'

          f'a = {a}\n'

          f'b = {b}\n'

          f'H(message) = {hash\_message\_sender}\n'

          f'Отправлено S{a, b} и {message}')

*# Получатель*

    hash\_message\_recipient = hash(message) % p

    comparison\_result = (FastPower(y, a, p) \* FastPower(a, b, p)

                         ) % p == FastPower(g, hash\_message\_recipient, p)

    print(f'--- Получатель ---\n'

          f'H(message) = {hash\_message\_recipient}\n'

          f'Подпись {"верифицирована!" if comparison\_result else "не верифицирована!"}')

    verification\_time = time.process\_time() - verification\_time

    print(f'--- Анализ ---\n'

          f'Время верификации: {verification\_time}')

def EDSBasedSchnorr(message: str, p: int):

*# Подготовка*

    verification\_time = time.process\_time()

    q = GetDivisors(p - 1)

    x = int(random.random() \* q)

    g = FindOrder(q, p)

    y = GetYByGAndP(g, p, x)

    print(f'--- Подготовка ---\n'

          f'p = {p}\n'

          f'q = {q}\n'

          f'g = {g}\n'

          f'x = {x}\n'

          f'y = {y}')

*# Отправитель*

    k = int(random.random() \* q)

    a = pow(g, k, p)

    hash\_message\_sender = hash(message + str(a)) % p

    b = pow(k + x \* hash\_message\_sender, 1, q)

    print(f'--- Отправитель ---\n'

          f'k = {k}\n'

          f'a = {a}\n'

          f'H(message||a) = {hash\_message\_sender}\n'

          f'b = {b}\n'

          f'Отправлено {message}||S{hash\_message\_sender, b}')

*# Получатель*

    X = pow(FastPower(g, b, p) \* FastPower(y, hash\_message\_sender, p), 1, p)

    hash\_message\_recipient = hash(message + str(X)) % p

    comparison\_result = hash\_message\_recipient == hash\_message\_sender

    print(f'--- Получатель ---\n'

          f'X = {X}\n'

          f'H(message||X) = {hash\_message\_recipient}\n'

          f'Подпись {"верифицирована!" if comparison\_result else "не верифицирована!"}')

    verification\_time = time.process\_time() - verification\_time

    print(f'--- Анализ ---\n'

          f'Время верификации: {verification\_time}')

def InverseNumber(a: int, N: int) -> int:

    if N == 0:

        return a, 1, 0

    else:

        d, x, y = InverseNumber(N, a % N)

        return d, y, x - y \* (a // N)

def GetFirstMutuallyPrimeNumber(number: int) -> int:

    def gcd(a, b):

        while b != 0:

            a, b = b, a % b

        return a

    for i in range(2, int(number \*\* 0.5) + 1):

        if gcd(number, i) == 1:

            return i

    return None

def IsPrimitiveRoot(number: int, p: int) -> bool:

    remains = [False] \* (p - 1)

    for i in range(1, p):

        power = FastPower(number, i, p)

        if remains[power - 1]:

            return False

        remains[power - 1] = True

    return True

def GeneratePrimitiveRoot(p: int) -> int:

    def is\_prime(n: int) -> bool:

        if n <= 1:

            return False

        if n <= 3:

            return True

        if n % 2 == 0 or n % 3 == 0:

            return False

        i = 5

        while i \* i <= n:

            if n % i == 0 or n % (i + 2) == 0:

                return False

            i += 6

        return True

    def euler\_totient(n: int) -> int:

        result = n

        i = 2

        while i \* i <= n:

            if n % i == 0:

                while n % i == 0:

                    n //= i

                result -= result // i

            i += 1

        if n > 1:

            result -= result // n

        return result

    if not is\_prime(p):

        return None

    for g in range(2, p):

        if FastPower(g, euler\_totient(p), p) == 1:

            return g

    return None

def GetBByKAndP(k: int, p: int, x: int, a: int, hash\_message: int) -> int:

    m = p - 1

    gcd, c, d = InverseNumber(k, m)

    if gcd == 1:

        inverse\_a = c % m

        return (((hash\_message - x \* a) % m) \* inverse\_a) % m

def GetDivisors(number: int) -> int:

    for i in range(2, number):

        if number % i == 0:

            return i

def FindOrder(q: int, mod: int) -> int:

    result = 2

    power = 0

    while power != 1:

        power = pow(result, q, mod)  *# Используем встроенную функцию pow*

        if power == 1:

            break

        result += 1

    return result - 1

def GetYByGAndP(g: int, p: int, x: int) -> int:

    gcd, c, d = InverseNumber(g, p)

    c = (c % p + p) % p

    return pow(c, x, p)

def FastPower(base: int, exponent: int, modulus: int) -> int:

    result = 1

    while exponent > 0:

        if exponent % 2 == 1:

            result = (result \* base) % modulus

        base = (base \* base) % modulus

        exponent //= 2

    return result

Листинг 2.1 –код программы, реализующие заданную ранее функциональность

**3. Результаты работы приложения**

Для выполнения расчетов достаточно необходимо запустить приложение. Рисунок 3.1 и 3.2 показывают требуемые в данной лабораторной работе результаты.

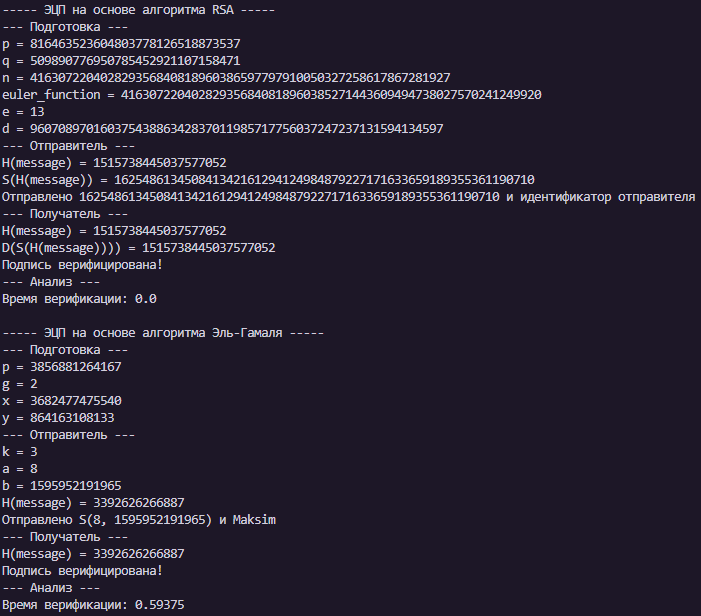


Рисунок 3.1 – Результат работы

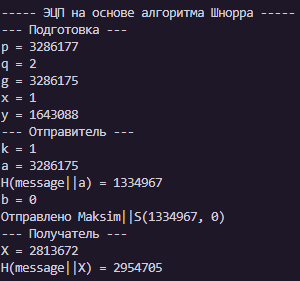


Рисунок 3.2 – Результат работы

**Вывод**

В ходе лабораторной работы было разработано приложение для генерации и верификации ЭЦП на основе алгоритмов RSA, Эль-Гамаля и Шнорра.