МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный университет имени М. В. Ломоносова»

ВОЕННЫЙ УЧЕБНЫЙ ЦЕНТР
Кафедра воздушно-космических сил

ОТЧЕТ ПО ПРАКТИЧЕСКОМУ ЗАДАНИЮ

Дисциплина: «Военно-специальная подготовка»
Тема: «Изучение механизмов функционирования систем электронной подписи»

Выполнил студент
взвода 2131, 2 г.о.
ВУС 751100
Пятов Владислав

апрель, 2021 г.

СОДЕРЖАНИЕ

1. Постановка задачи

В соответствии с *вариантом 18*, требуется разработать и программно реализовать учебную систему электронной подписи (ЭП) на основе следующих криптографических алгоритмов:

1. Алгоритмы формирования и проверки ЭП в соответствии с ГОСТ Р 34.10-2012.
2. Хэш-функция SHA-256 (из семейства SHA-2).

2. Описание используемых алгоритмов

2.1 ГОСТ Р 34.10-2012

Действующий отечественный стандарт электронной подписи ГОСТ Р 34.10-2012 был введен в действие в 2012 году. Действующий алгоритм электронной подписи строится на основе эллиптических кривых. Рассмотрим параметры подписи:

p – большое просто число длиной от 256 битов;

a, b – такие целые числа, меньшие p, что $4a^3 + 27b^2 \neq 0\ (mod\ p)$;

Эти числа определяют уравнение следующего вида

$$y^2 = x^3 + ax + b\ (mod\ p)$$

Множество пар $(x, y), 0 \leq x \leq p - 1, 0 \leq y \leq p - 1,$ удовлетворяющих этому уравнению, называется эллиптической кривой. К этому множеству обычно добавляется специальная бесконечно удаленная точка, которая обычно обозначается как O. Будем обозначать множество точек эллиптической кривой как $E(a, b)$.

Введем на множестве точек эллиптической кривой операцию сложения. Сумма любой точки с бесконечно удаленной равна самой этой точке. Сумма точек (x, y) и $(x, -y) = (x, p - y)$ равна бесконечно удаленной точке, т.е. $(x, -y)$ является обратной к точке (x, y). Сумма точек (x_P, y_P) и (x_Q, y_Q), вычисляется по формуле:

$$x_{P+Q} = \lambda - x_P - x_Q\ (mod\ p),$$

$$y_{P+Q} = -y_P + \lambda\left(x_P - x_{P+Q}\right)(mod\ p).$$

Здесь

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P}(mod\ p),$$

если $x_Q \neq x_P$ и

$$\lambda = \frac{3x_P^2 + a}{2y_P}(mod\ p),$$

если $x_Q = x_P$ (в этом случае операция сложения – операция удвоения точки, т.к. $y_Q = y_P$).

В силу того, что множество рассматриваемых пар точек конечно, можно говорить о конечной мощности множества точек эллиптической кривой $E(a, b)$. Обозначим число точек эллиптической кривой через $|E(a, b)|$.

q – простой делитель мощности $|E(a, b)|$ множества точек эллиптической кривой; требуется, чтобы размер делителя был 256 бит; также необходимо, чтобы во множестве точек была некоторая точка P порядка q, т.е. для этой точки выполняется следующее свойство: если сложить точку P саму с собой q раз, то получится O, при этом если сложить точку P саму с собой меньшее число раз, то точки O не получится. Для краткости сумму точки P саму с собой k раз будем обозначать как $k \cdot P$.

В качестве секретного ключа (или ключа подписи) выступает случайное целое число x, $1 < x < q$. Открытый ключ (ключ проверки подписи) Y вычисляется по формуле:

$$Y = x \cdot P.$$

Таким образом, ключ проверки подписи представляет собой точку эллиптической кривой $E(a, b)$, которая получается сложением точки P самой с собой x раз.

Формирование подписи под сообщение M происходит следующим образом:

1) вычисляется число $z = h(M)(mod\ q)$; если $z = 0$, то положить $z = 1$;
2) выбирается случайное число k, $0 < k < q$;
3) вычисляется $(x_R, y_R) = k \cdot P$ и $r = x_R(mod\ q)$;
4) вычисляется $s = (r \cdot x + k \cdot e)\ (mod\ q)$;
5) если $r = 0$ или $s = 0$, то повторно выполнить шаги 2-4.

Подписью под сообщением M является пара (r, s).

Проверка подписи под сообщением M происходит следующим образом:

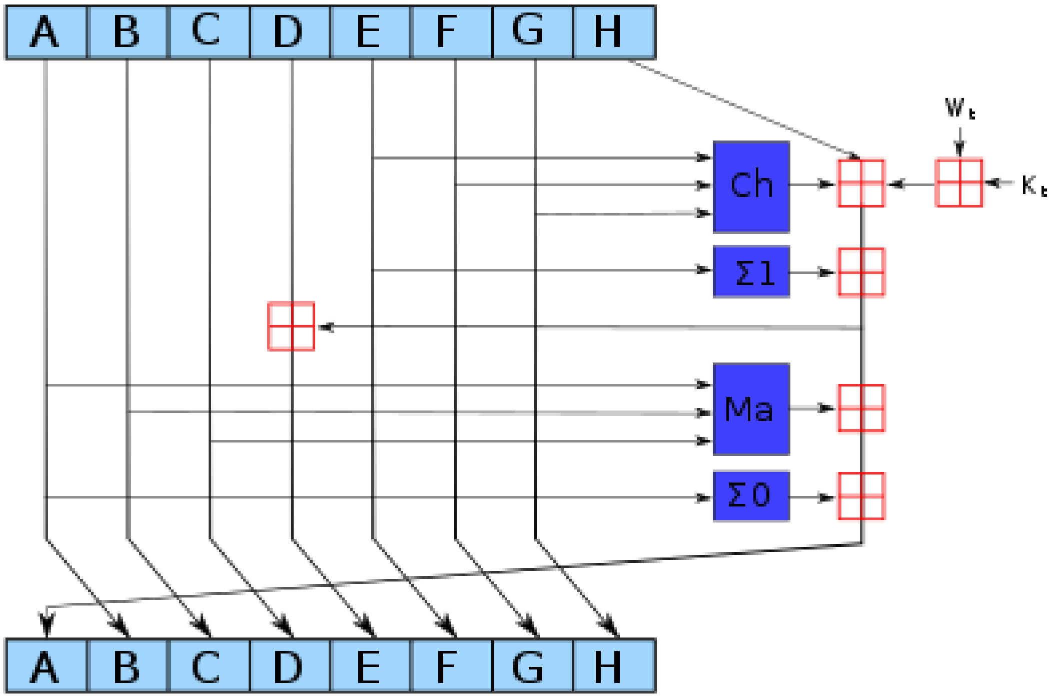
1) если не выполнено хотя бы одно из неравенств $0 < r < q$ и $0 < s < q$, то подпись отвергается;
2) вычисляется $z = h(M)(mod\ q)$; если $z = 0$, то положить $z = 1$;
3) вычисляется $v = z^{-1}(mod\ q)$;
4) вычисляется $u_1 = s \cdot v(mod\ q)$;
5) вычисляется $u_2 = -r \cdot v\ (mod\ q)$;
6) вычисляется $(x_{R'}, y_{R'}) = u_1 P + u_2 Y$ и $r' = x_{R'}(mod\ q)$;
7) подпись принимается, если выполняется равенство $r' = r$.

2.2 Хеш-функция SHA-256

SHA-256 представляет собой однонаправленную функцию для создания цифровых отпечатков фиксированной длины (256 бит, 32 байт) из входных данных размером до 2,31 эксабайт (2^{64} бит) и является частным случаем алгоритма из семейства криптографических алгоритмов SHA-2 (Secure Hash Algorithm Version 2) опубликованным АНБ США в 2002 году.

Хеш-функции семейства SHA-2 построены на основе структуры Меркла — Дамгарда.

Исходное сообщение после дополнения разбивается на блоки, каждый блок — на 16 слов. Алгоритм пропускает каждый блок сообщения через цикл с 64 итерациями. На каждой итерации 2 слова преобразуются, функцию преобразования задают остальные слова. Результаты обработки каждого блока складываются, сумма является значением хеш-функции. Так как инициализация внутреннего состояния производится результатом обработки предыдущего блока, то нет возможности обрабатывать блоки параллельно. Графическое представление одной итерации обработки блока данных:



Рассмотрим подробнее алгоритм работы SHA-256, разбив его на несколько этапов:

1 этап – предобработка входных данных:

- входная последовательность переводится в двоичную систему счисления;
- в конец последовательности добавляется единица (10 с.с.);
- последовательность дополняется нулями до тех пор, пока не станет кратной 448;
- в конец последовательности добавляется 64 бита, означающие длину входной последовательности в двоичной системе счисления;

2 этап – инициализация:

- производится инициализация 8 значений хеша $(h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7)$, которые представляют собой первые 32 бита дробных частей квадратных корней первых 8 простых чисел: 2, 3, 5, 7, 11, 13, 17, 19;
- производится инициализация 64 значений округленных констант, являющихся первыми 32 битами дробных частей кубических корней первых 64 простых чисел;

3 этап – обработка каждой 512-битной подпоследовательности последовательности, полученной на первом этапе:

- последовательность преобразуется в массив из 16 32-битных чисел;
- добавляется 48 32-битных чисел, инициализированных нулями для получения массива из 64 чисел;

- последние 48 элементов преобразуются с использованием специализированного алгоритма (доступен к ознакомлению в разделе программной реализации);
- инициализируются переменные $a = h_0, b = h_1, c = h_2, d = h_3, e = h_4,\ f = h_5, g = h_6, h = h_7$;
- применяется алгоритм сжатия (доступен к ознакомлению в разделе программной реализации), в результате которого происходит обновление переменных $h_0..h_7$ по правилу $h_0 = h_0 + a,\ h_1 = h_1 + b, \dots, h_7 = h_7 + h$;
- 3 этап повторяется для следующей подпоследовательности.

4 этап – консолидация:

- Значения переменных конкатенируются в одно результирующее значение хеша.

3. Особенности программной реализации

3.1 *ГОСТ Р 34.10-2012*

В основе программной реализации алгоритма формирования и проверки ЭП в соответствии с ГОСТ Р 34.10-2012 на языке python3 лежит класс Curve для работы с эллиптической кривой. При создании объекта класса инициализируются следующие параметры:

p – модуль эллиптической кривой;

q – порядок циклической подгруппы группы точек эллиптической кривой;

a, b – коэффициенты эллиптической кривой;

x, y – координаты точки эллиптической кривой.

Класс Curve реализует следующие возможности:

1) проверка принадлежности точки кривой:

```
def is_on_curve(self, point):
    x, y = point
    r1 = y * y % self.p
    r2 = ((x * x + self.a) * x + self.b) % self.p
    return r1 == self.pos(r2)
```

2) операция сложения двух точек:

```
def _sum(self, x1, y1, x2, y2):
    if x1 == x2 and y1 == y2:
        t = ((3 * x1 * x1 + self.a) * modinvert(2 * y1, self.p)) % self.p
    else:
        tx = self.pos(x2 - x1) % self.p
        ty = self.pos(y2 - y1) % self.p
        t = (ty * modinvert(tx, self.p)) % self.p
    tx = self.pos(t * t - x1 - x2) % self.p
    ty = self.pos(t * (x1 - tx) - y1) % self.p
    return tx, ty
```

3) операция многократного сложения точки самой с собой (для генерации публичного ключа, подписи и верификации):

```
def summator(self, k, x=None, y=None):
    x = x or self.x
    y = y or self.y
    tx = x
    ty = y
    if k == 0:
        raise ValueError("Bad k value")
    k -= 1
    while k != 0:
        if k & 1 == 1:
            tx, ty = self._sum(tx, ty, x, y)
        k = k >> 1
        x, y = self._sum(x, y, x, y)
    return tx, ty
```

Немаловажную роль играют отдельные методы генерации публичного ключа, подписи и верификации, приводимые далее:

```
def public_key(curve, prv):
    """Generate public key from the private one

    :param Curve curve: curve to use
    :param long prv: private key
    :returns: public key's parts, X and Y
    :rtype: (long, long)
    """
    return curve.summator(prv)
```

```python
def sign(curve, prv, digest, rand=None):
    """Calculate signature for provided digest

    :param Curve curve: curve to use
    :param long prv: private key
    :param digest: digest for signing
    :type digest: bytes, 32 or 64 bytes
    :param rand: optional predefined random data used for k/r generation
    :type rand: bytes, 32 or 64 bytes
    :returns: signature, BE(S) || BE(R)
    :rtype: bytes, 64 or 128 bytes
    """
    size = curve.point_size
    q = curve.q
    e = bytes2long(digest) % q
    if e == 0:
        e = 1
    while True:
        if rand is None:
            rand = urandom(size)
        elif len(rand) != size:
            raise ValueError("rand length != %d" % size)
        k = bytes2long(rand) % q
        if k == 0:
            continue
        r, _ = curve.summator(k)
        r %= q
        if r == 0:
            continue
        d = prv * r
        k *= e
        s = (d + k) % q
        if s == 0:
            continue
        break
    return long2bytes(s, size) + long2bytes(r, size)
```

```python
def verify(curve, pub, digest, signature):
    """Verify provided digest with the signature

    :param Curve curve: curve to use
    :type pub: (long, long)
    :param digest: digest needed to check
    :type digest: bytes, 32 or 64 bytes
    :param signature: signature to verify with
    :type signature: bytes, 64 or 128 bytes
    :rtype: bool
    """
    size = curve.point_size
    if len(signature) != size * 2:
        raise ValueError("Invalid signature length")
    q = curve.q
    p = curve.p
    s = bytes2long(signature[:size])
    r = bytes2long(signature[size:])
    if r <= 0 or r >= q or s <= 0 or s >= q:
        return False
    e = bytes2long(digest) % curve.q
    if e == 0:
        e = 1
    v = modinvert(e, q)
    z1 = s * v % q
    z2 = q - r * v % q
    p1x, p1y = curve.summator(z1)
    q1x, q1y = curve.summator(z2, pub[0], pub[1])
    lm = q1x - p1x
    if lm < 0:
        lm += p
    lm = modinvert(lm, p)
    z1 = q1y - p1y
    lm = lm * z1 % p
    lm = lm * lm % p
    lm = lm - p1x - q1x
    lm = lm % p
    if lm < 0:
        lm += p
    lm %= q

    return lm == r
```

3.2 Хеш-функция SHA-256

Программная реализация алгоритма хеширования SHA-256 на языке python3 полностью повторяет изложенный выше алгоритм. Методы, поля и переменные необходимые для реализации инкапсулированы в классе SHA256. Ключевыми методами являются метод _update, который обновляет хешируемые данные добавлением новых и метод_compress, которые выполняет хеширование:

```python
def update(self, m):
    if not m:
        return

    self._counter += len(m)
    m = self._cache + m

    for i in range(0, len(m) // 64):
        self._compress(m[64 * i:64 * (i + 1)])
    self._cache = m[-(len(m) % 64):]
```

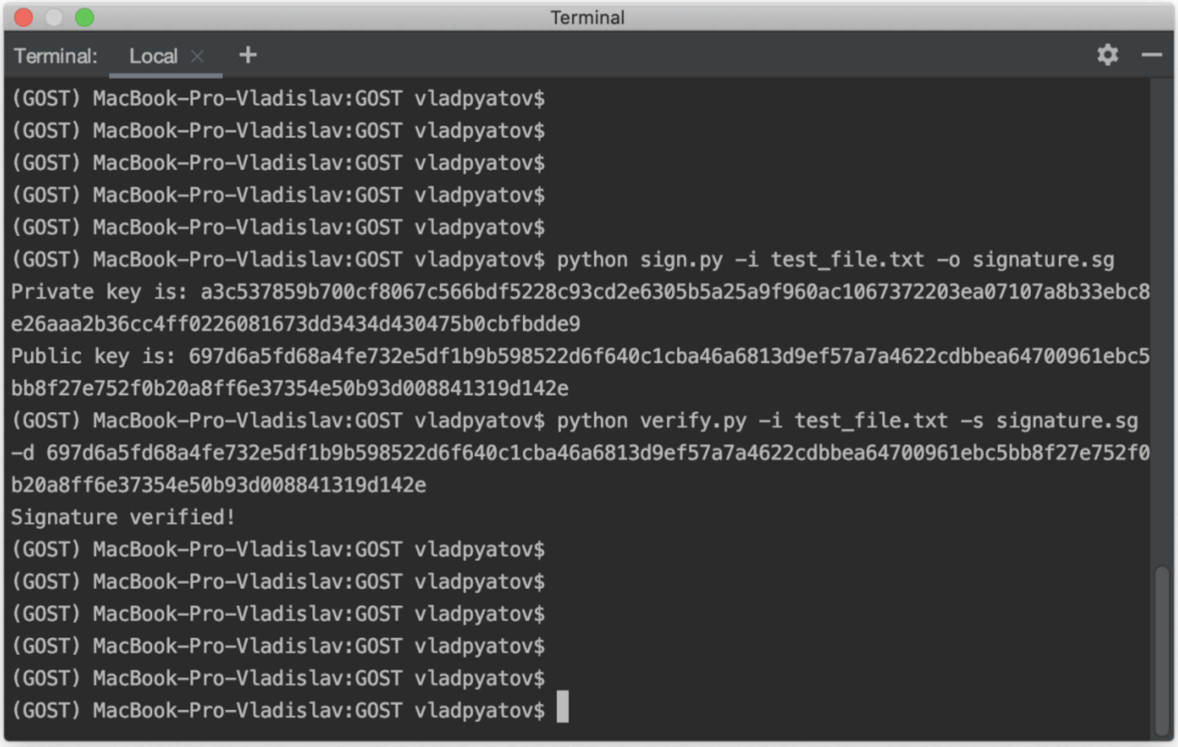
```python
def _compress(self, c):
    w = [0] * 64
    w[0:16] = struct.unpack('!16L', c)

    for i in range(16, 64):
        s0 = _rotr(w[i-15], 7) ^ _rotr(w[i-15], 18) ^ (w[i-15] >> 3)
        s1 = _rotr(w[i-2], 17) ^ _rotr(w[i-2], 19) ^ (w[i-2] >> 10)
        w[i] = (w[i-16] + s0 + w[i-7] + s1) & F32

    a, b, c, d, e, f, g, h = self._h

    for i in range(64):
        s0 = _rotr(a, 2) ^ _rotr(a, 13) ^ _rotr(a, 22)
        t2 = s0 + _maj(a, b, c)
        s1 = _rotr(e, 6) ^ _rotr(e, 11) ^ _rotr(e, 25)
        t1 = h + s1 + _ch(e, f, g) + self._k[i] + w[i]

        h = g
        g = f
        f = e
        e = (d + t1) & F32
        d = c
        c = b
        b = a
        a = (t1 + t2) & F32

    for i, (x, y) in enumerate(zip(self._h, [a, b, c, d, e, f, g,
h])):
        self._h[i] = (x + y) & F32
```

4. Результаты тестирования программы

Программа тестировалась со стандартным набором параметров:

$$p = 57896044618658097711785492504343953926634992332820282019728792003956564821041$$
$$q = 57896044618658097711785492504343953927082934583725450622380973592137631069619$$
$$a = 7$$
$$b = 43308876546767276905765904595650931995942111794451039583252968842033849580414$$
$$x = 2$$
$$y = 40189740565390375033354494229370597756357393899055450806909793652134315662 80$$

Результат тестирования:



Криптостойкость цифровой подписи опирается на две компоненты — на стойкость хеш-функции и на стойкость самого алгоритма шифрования.

В 2003 году Гилберт и Хандшух провели исследование *SHA-2*, но не нашли каких-либо уязвимостей. Однако в марте 2008 года индийские исследователи Сомитра Кумар Санадия и Палаш Саркар опубликовали найденные ими коллизии для 22 итераций *SHA-256* и *SHA-512*. В сентябре того же года они представили метод конструирования коллизий для усечённых вариантов *SHA-2* (21 итерация). Позднее были найдены методы конструирования коллизий для 31 итерации *SHA-256*.

Стойкость алгоритма шифрования основывается на проблеме дискретного логарифмирования в группе точек эллиптической кривой. На данный момент нет метода решения данной проблемы хотя бы с субэкспоненциальной сложностью.

Один из самых быстрых алгоритмов, на данный момент, при правильном выборе параметров — ρ-метод и I-метод Полларда.

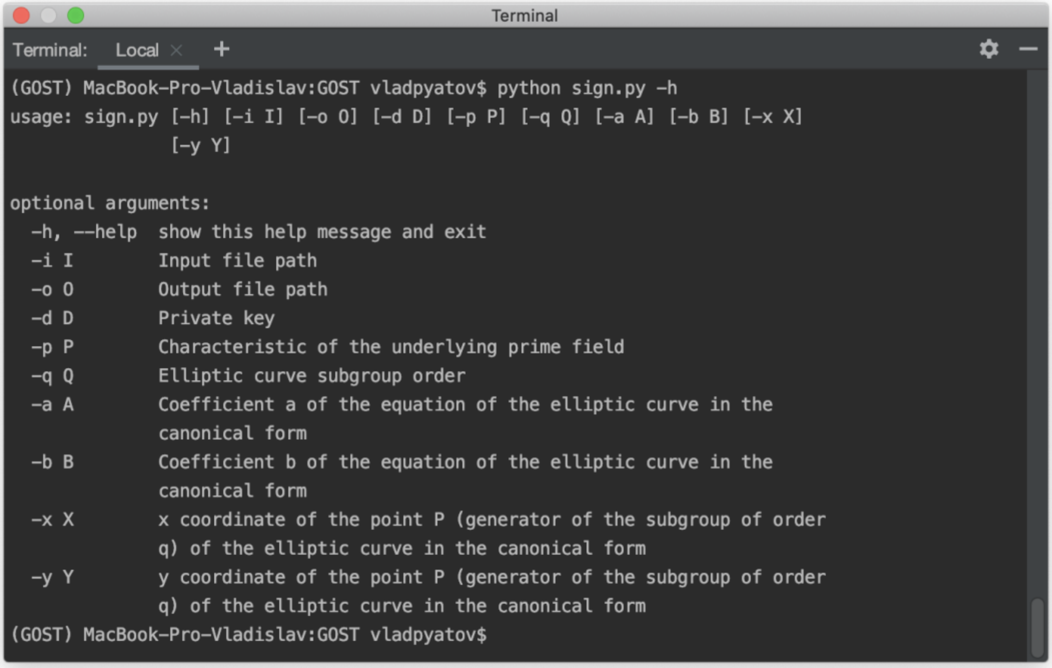
Для оптимизированного ρ-метода Полларда вычислительная сложность оценивается как $O\left(q^{\frac{1}{2}}\right)$. Таким образом для обеспечения криптостойкости 10^{30} операций необходимо использовать 256-разрядное q

5. Руководство пользователя

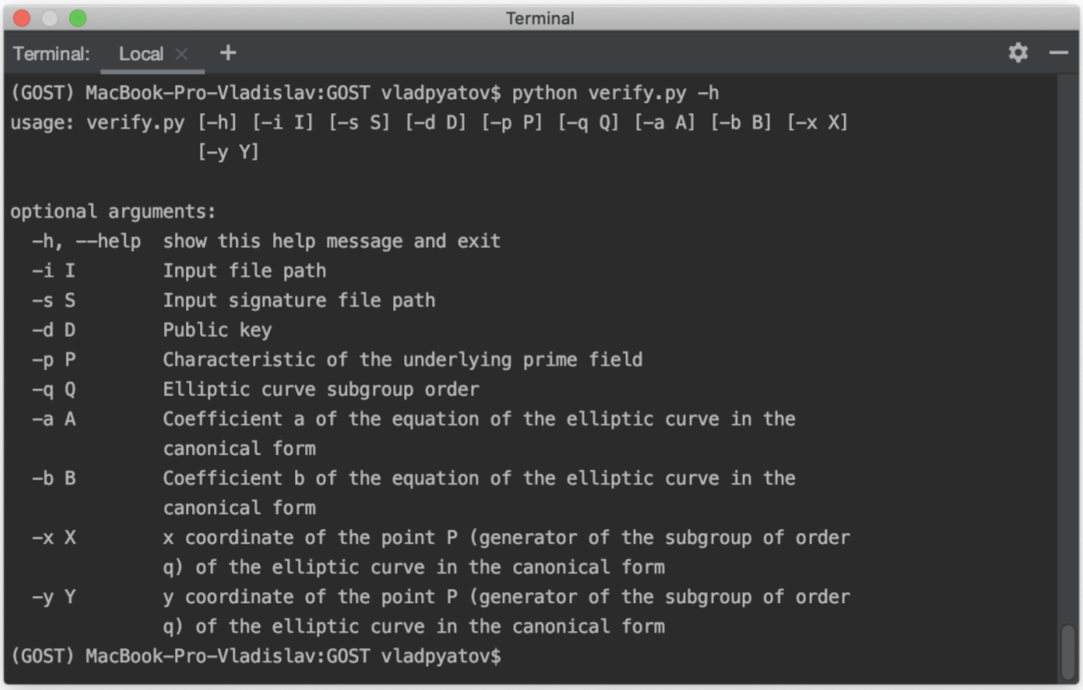
Исходный код доступен в репозитории по адресу https://github.com/VladPyatov/GOST

Пользователю доступны два скрипта:

1) sign.py – подписывает входной файл



2) verify.py – выполняет верификацию по заданным входному файлу, подписи и публичному ключу



В обоих скриптах по умолчанию выставлены параметры эллиптической кривой в соответствии с приложением А1 ГОСТ 34.10-2012, но при желании пользователь может задать свои параметры.