Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Лабораторная работа №7. Криптография с использованием эллиптических кривых

Выполнил: cтудент гр. 853501

Яковлев А.Б.

Проверил:

Протько М.И.

Минск 2021

# Постановка задачи и описание алгоритма

**Цель:** Реализовать программное средство формирования электронной цифровой подписи на основе эллиптических кривых ECDSA и программное средство, реализующее простой подход к шифрованию/дешифрованию с использованием эллиптических кривых.

Преимущество подхода на основе *эллиптических кривых* в сравнении с задачей факторизации числа, используемой в RSA, или задачей целочисленного логарифмирования, применяемой в алгоритме Диффи-Хеллмана и в DSS, заключается в том, что в данном случае обеспечивается эквивалентная защита при меньшей длине ключа.

В общем случае уравнение *эллиптической кривой*   Е имеет вид:

y2 + axy + by = x3 + cx2 + dx + e

В качестве примера рассмотрим *эллиптическую кривую*   Е, уравнение которой имеет вид: y2 + y = x3 - x2

На этой кривой лежат только четыре точки, координаты которых являются целыми числами. Это точки

А (0, 0), В (1, -1), С (1, 0) и D (0, -1)

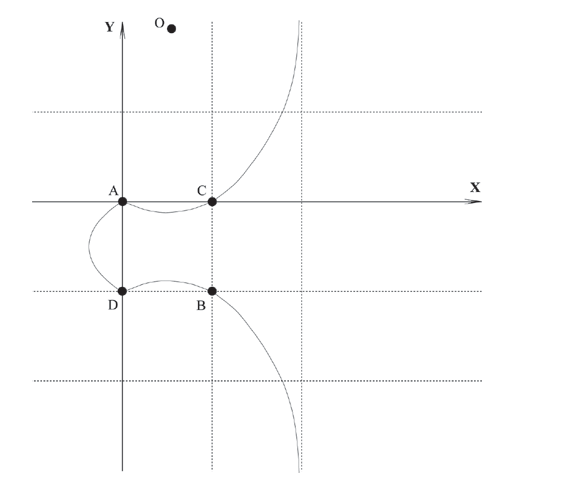


Рисунок 1 – Пример эллиптической кривой с четырьмя точками

Для определения *операции сложения для точек на эллиптической кривой* сделаем следующие предположения:

 На плоскости существует бесконечно удаленная точка 0Е, в которой сходятся все вертикальные прямые.

 Будем считать, что касательная к кривой пересекает точку касания два раза.

 Если три точки *эллиптической кривой* лежат на прямой линии, то их сумма есть 0.

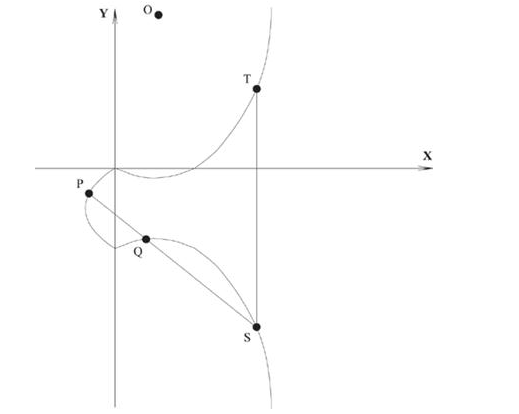


Рисунок 2 – Сложение точек на эллиптической кривой

Введем следующие правила сложения точек на *эллиптической кривой*:

 Точка 0 выступает в роли *нулевого элемента*. Так, 0 = -0 и для любой точки Р на *эллиптической кривой* Р + 0 = Р.

 Вертикальная линия пересекает кривую в двух точках с одной и той же координатой х - скажем, S = (x, y) и T = (x, -y). Эта прямая пересекает кривую и в бесконечно удаленной точке. Поэтому Р1 + Р2 + 0 = 0 и Р1 = -Р2.

 Чтобы сложить две точки P и Q (см. рисунок 11.2) с разными координатами х, следует провести через эти точки прямую и найти точку пересечения ее с *эллиптической кривой*. Если прямая не является касательной к кривой в точках P или Q, то существует только одна такая точка, обозначим ее S. Согласно нашему предположению P + Q + S = О

Следовательно, P + Q = -S или P + Q = T.

Если прямая является касательной к кривой в какой-либо из точек P или Q, то в этом случае следует положить S = P или S = Q соответственно.

 Чтобы удвоить точку Q, следует провести касательную в точке Q и найти другую точку пересечения S с *эллиптической кривой*. Тогда Q + Q = 2 × Q = -S.

Введенная таким образом *операция сложения* подчиняется всем обычным правилам сложения, в частности коммутативному и ассоциативному законам. Умножение точки Р *эллиптической кривой* на положительное число k определяется как сумма k точек Р.

В криптографии с использованием *эллиптических кривых* все значения вычисляются по модулю р, где р является простым числом. Элементами данной *эллиптической кривой* являются пары неотрицательных целых чисел, которые меньше р и удовлетворяют частному виду *эллиптической кривой*:

y2 ≡x3 + ax + b (mod p)

Такую кривую будем обозначать Ep (a,b). При этом числа а и b должны быть меньше р и должны удовлетворять условию 4a3 + 27b2 (mod p) ≠ 0. Множество точек на *эллиптической кривой* вычисляется следующим образом.

Для каждого такого значения х, что 0≤х≤р, вычисляется x3 + ax + b (mod p).

Для каждого из полученных на предыдущем шаге значений выясняется, имеет ли это значение квадратный корень по модулю р. Если нет, то в Ep (a,b) нет точек с этим значением х. Если корень существует, имеется два значения y, соответствующих операции извлечения квадратного корня (исключением является случай, когда единственным значением оказывается y = 0). Эти значения (x,y) и будут точками Ep (a,b).

Множество точек Ep (a,b) обладает следующими свойствами:

1. Р + 0 = Р

2. Если Р = (x,y), то Р + (x,-y) = 0. Точка (x,-y) является отрицательным значением точки Р и обозначается -Р. Заметим, что (x,-y) лежит на *эллиптической кривой* и принадлежит Ep (a,b).

3. Если Р = (x1,y1) и Q = (x2,y2), где P ≠ Q, то P + Q = (x3,y3) определяется по следующим формулам:

4. x3≡  λ2 - x1 - x2 (mod p)

5. y3 ≡ λ (x1 - x3) - y1 (mod p)

где

(y2 - y1)/(x2 - x1) , если P ≠ Q

Λ = (3x12 + a)/2y1 , если P = Q

Число λ есть угловой коэффициент секущей, проведенной через точки P = (x1, y1) и Q = (x2, y2). При P = Q секущая превращается в касательную, чем и объясняется наличие двух формул для вычисления λ.

Задача, которую должен решить в этом случае атакующий, есть своего рода задача ***"дискретного логарифмирования на эллиптической кривой"***, и формулируется она следующим образом. Даны точки P и Q на *эллиптической кривой* Ep (a,b). Необходимо найти коэффициент k < p такой, что

P = k × Q

Относительно легко вычислить P по данным k и Q, но довольно трудно вычислить k, зная P и Q.

**Алгоритм цифровой подписи**

Алгоритм ECDSA (Elliptic Curve Digest Signature Algorithm) принят в качестве стандартов ANSI X9F1 и IEEE P1363.

Создание ключей:

1. Выбирается *эллиптическая кривая* Ep (a,b). Число точек на ней должно делиться на большое целое n.

2. Выбирается точка РEp (a,b).

3. Выбирается случайное число d  [1, n-1].

4. Вычисляется Q = d × P.

5. Закрытым ключом является d, открытым ключом – (E, P, n, Q).

Создание подписи:

1. Выбирается случайное число k [1, n-1].

2. Вычисляется k × P = (x1, y1)   и  r = x1 (mod n).

Проверяется, чтобы r не было равно нулю, так как в этом случае подпись не будет зависеть от закрытого ключа. Если r = 0, то выбирается другое случайное число k.

3. Вычисляется k-1 mod n

4. Вычисляется s = k-1 (Н(M) + dr) (mod n)

Проверяется, чтобы s не было равно нулю, так как в этом случае необходимого для проверки подписи числа s-1 mod n не существует. Если s = 0, то выбирается другое случайное число k.

Подписью для сообщения М является пара чисел (r,s).

Проверка подписи:

1. Проверить, что целые числа r и s принадлежат диапазону чисел [0, n-1]. В противном случае результат проверки отрицательный, и подпись отвергается.

2. Вычислить w = s-1 (mod n) и H(M)

3. Вычислить u1 = H(M) w (mod n), u2 = rw (mod n)

4. Вычислить u1P + u2Q = (x0, y0), v = x0 (mod n)

5. Подпись верна в том и только том случае, когда v = r.

**Алгоритмы шифрования**

Рассмотрим самый простой подход к шифрованию/дешифрованию с использованием *эллиптических кривых*. Задача состоит в том, чтобы зашифровать сообщение М, которое может быть представлено в виде точки на эллиптической кривой Pm (x,y).

Как и в случае обмена ключом, в системе шифрования/дешифрования в качестве параметров рассматривается *эллиптическая кривая* Ep (a,b) и точка G на ней. Участник B выбирает закрытый ключ nB и вычисляет открытый ключ PB = nB × G. Чтобы зашифровать сообщение Pm используется открытый ключ получателя B   PB. Участник А выбирает случайное целое положительное число k и вычисляет зашифрованное сообщение Cm, являющееся точкой на *эллиптической кривой*.

Cm = {k × G, Pm + k × PB}

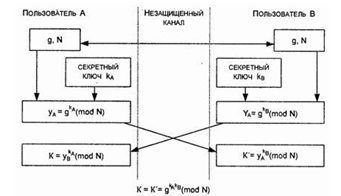
Чтобы дешифровать сообщение, участник В умножает первую координату точки на свой закрытый ключ и вычитает результат из второй координаты:

Pm + k × PB - nB × (k × G) = Pm + k × (nB × G) - nB × (k × G) = Pm

Участник А зашифровал сообщение Pm добавлением к нему kxPB. Никто не знает значения k, поэтому, хотя PB и является открытым ключом, никто не знает k × PB. Противнику для восстановления сообщения придется вычислить k, зная G и k × G. Сделать это будет нелегко.

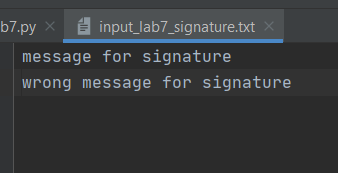
Получатель также не знает k, но ему в качестве подсказки посылается k × G. Умножив k × G на свой закрытый ключ, получатель получит значение, которое было добавлено отправителем к незашифрованному сообщению. Тем самым получатель, не зная k, но имея свой закрытый ключ, может восстановить незашифрованное сообщение.

# Блок-схемы алгоритмов

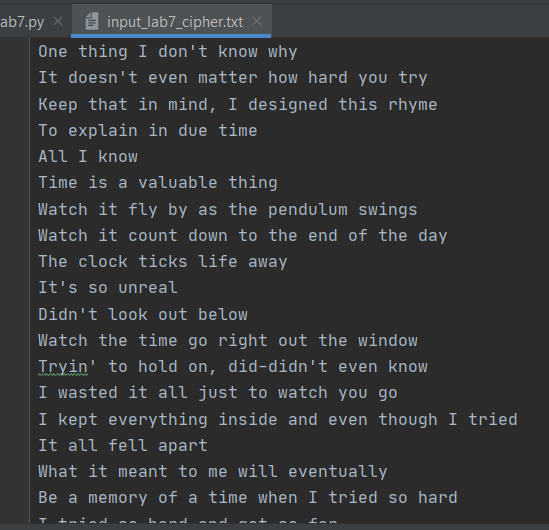


# Результаты выполнения программы

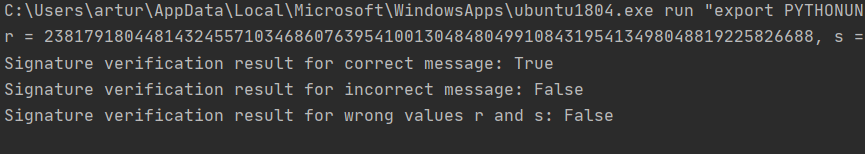
Входной файл для электронной подписи



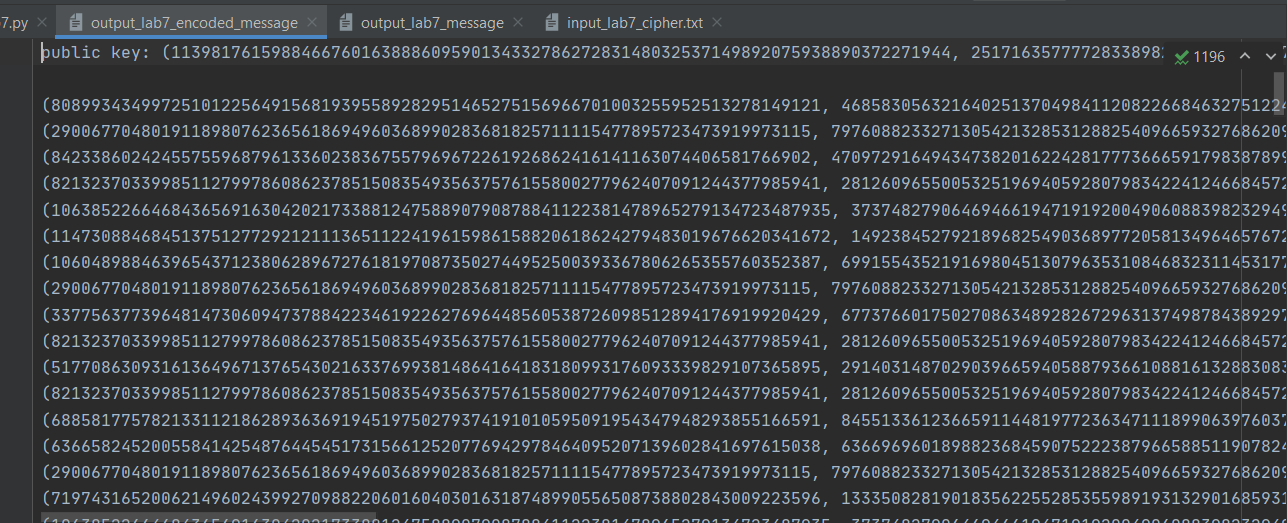
Входной файл с сообщением для шифрования



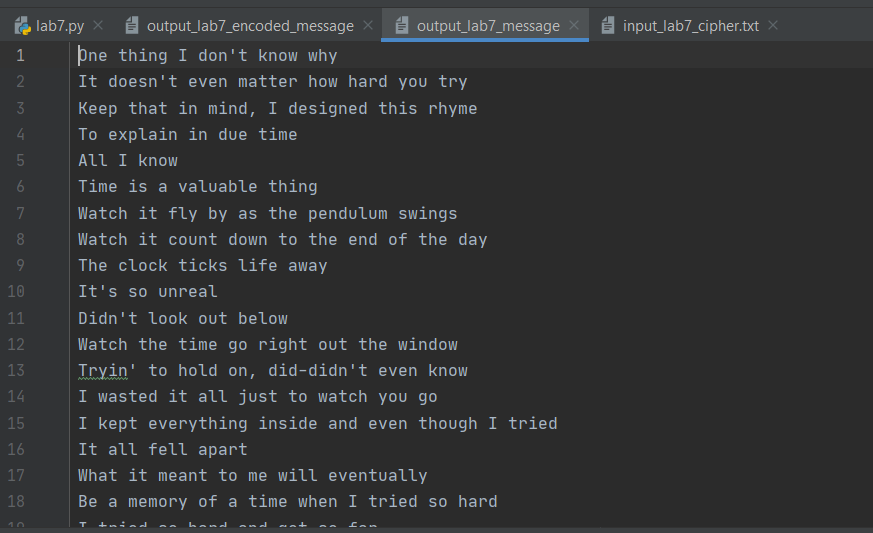
Результат выполнения:



Файл с зашифрованным сообщением:



Файл с расшифрованным сообщением:



# Исходный код

import typing as tp  
from secrets import randbelow  
  
import tinyec.ec as ec  
from tinyec import registry  
  
from functions import get\_message\_hash  
  
  
# making classes Point and Inf from tinyec hashable  
# so they can be used as a dict key in encoder  
ec.Point.\_\_hash\_\_ = lambda self: hash(self.x) ^ hash(self.y)  
ec.Inf.\_\_hash\_\_ = lambda self: hash(self.x) ^ hash(self.y)  
  
  
class BaseEllipticCurveClass:  
 *"""  
 Base class for signature and encoder  
 Implements key pair generation needed for both classes  
 """* def \_\_init\_\_(self, curve: ec.Curve) -> None:  
 self.curve = curve  
  
 def generate\_key\_pair(self) -> tp.Tuple[int, ec.Point]:  
 private\_key = randbelow(self.curve.field.n - 1) + 1  
 public\_key = self.curve.g \* private\_key  
 return private\_key, public\_key  
  
  
class DigitalSignature(BaseEllipticCurveClass):  
 def \_\_init\_\_(self, curve: ec.Curve) -> None:  
 super().\_\_init\_\_(curve)  
 self.\_private\_key = 0  
  
 def get\_signature(self, message: str) -> tp.Tuple[int, int, ec.Point]:  
 key, public\_key = self.generate\_key\_pair()  
 r, s = 0, 0  
 hashed\_message = get\_message\_hash(message)  
 while r == 0 or s == 0:  
 self.\_private\_key = randbelow(self.curve.field.n - 1) + 1  
 point = self.curve.g \* self.\_private\_key  
 r = point.x % self.curve.field.n  
 try:  
 s = pow(self.\_private\_key, -1, self.curve.field.n) \* (hashed\_message + key \* r) % \  
 self.curve.field.n  
 pow(s, -1, self.curve.field.n)  
 except ValueError:  
 s = 0  
 return r, s, public\_key  
  
 def verify\_signature(self, message: str, r: int, s: int, public\_key: ec.Point) -> bool:  
 hashed\_message = get\_message\_hash(message)  
 w = pow(s, -1, self.curve.field.n)  
 u1 = hashed\_message \* w % self.curve.field.n  
 u2 = r \* w % self.curve.field.n  
 point = self.curve.g \* u1 + public\_key \* u2  
 return point.x == r  
  
  
class Encoder(BaseEllipticCurveClass):  
 def \_\_init\_\_(self, curve: ec.Curve):  
 super().\_\_init\_\_(curve)  
 multiplier = randbelow(self.curve.field.n - 1) + 1  
 self.\_encoding\_table = {  
 i: curve.g \* multiplier \* i for i in range(256)  
 }  
 self.\_decoding\_table = {  
 curve.g \* multiplier \* i: i for i in range(256)  
 }  
  
 def \_message\_to\_points(self, message: str) -> tp.List[ec.Point]:  
 *"""  
 This method allows to convert string into points on the elliptic curve.  
 It is required for encrypting, as only curve points can be encrypted* ***:param*** *message: message for further encoding* ***:return****: message, converted to points  
 """* result: tp.List[ec.Point] = []  
 for byte in message.encode('UTF-8'):  
 result.append(self.\_encoding\_table[byte])  
 return result  
  
 def \_points\_to\_message(self, points: tp.List[ec.Point]) -> str:  
 *"""  
 Inverse of the previous method* ***:param*** *points: decrypted sequence of points* ***:return****: decrypted message  
 """* message: tp.List[str] = []  
 for point in points:  
 message.append(self.\_decoding\_table[point])  
 return ''.join(map(chr, message))  
  
 def encode(self, message: str, public\_key: ec.Point) -> tp.Tuple[tp.List[ec.Point], ec.Point]:  
 assert self.curve == public\_key.curve  
 assert len(message)  
 points = self.\_message\_to\_points(message)  
 k = randbelow(self.curve.field.n - 1) + 1  
 encoded\_message = [point + public\_key \* k for point in points]  
 return encoded\_message, self.curve.g \* k  
  
 def decode(self, encoded\_message: tp.List[ec.Point], private\_key: int, public\_key: ec.Point) -> str:  
 assert self.curve == encoded\_message[0].curve  
 assert self.curve == public\_key.curve  
 return self.\_points\_to\_message([encoded\_char - public\_key \* private\_key  
 for encoded\_char in encoded\_message])  
  
  
def test\_signature() -> None:  
 curve = registry.get\_curve('secp256r1')  
 ds = DigitalSignature(curve)  
 with open('input\_files/input\_lab7\_signature.txt', 'r') as f:  
 message = f.readline()  
 wrong\_message = f.readline()  
 r, s, public\_key = ds.get\_signature(message)  
 print(f'r = {r}, s = {s}')  
 print(f'Signature verification result for correct message: {ds.verify\_signature(message, r, s, public\_key)}')  
 print(f'Signature verification result for incorrect message: {ds.verify\_signature(wrong\_message, r, s, public\_key)}')  
 print(f'Signature verification result for wrong values r and s: {ds.verify\_signature(wrong\_message, r - 1, s + 5, public\_key)}')  
  
  
def test\_encoder() -> None:  
 curve = registry.get\_curve('secp256r1')  
 encoder = Encoder(curve)  
 first\_private\_key, first\_public\_key = encoder.generate\_key\_pair()  
 with open('input\_files/input\_lab7\_cipher.txt', 'r') as f:  
 message = f.read()  
 encoded\_message, public\_key = encoder.encode(message, first\_public\_key)  
 with open('output\_files/output\_lab7\_encoded\_message', 'w') as f:  
 f.write('public key: ' + str(public\_key) + '\n\n')  
 f.writelines([str(point) + '\n' for point in encoded\_message])  
 decoded\_message = encoder.decode(encoded\_message, first\_private\_key, public\_key)  
 with open('output\_files/output\_lab7\_message', 'w') as f:  
 f.write(decoded\_message)  
  
  
def main() -> None:  
 test\_signature()  
 test\_encoder()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# Вывод

В результате выполнения работы была реализована программа, позволяющая формировать и проверять электронную цифровую подпись на основе алгоритма с использованием эллиптических кривых. Также был реализован подход к шифрованию текстовых данных при помощи точек на эллиптической кривой.