

УО «Белорусский государственный университет информатики и радиоэлектроники»
Кафедра ПОИТ

Отчет по лабораторной работе №3
по предмету
Теория Информации
Вариант “Эллиптические кривые”

Выполнил:
Матюшенко А.Д.

Проверил:
Болтак С.В.
Группа 351001

Минск 2025

Задание:

1. Для заданного M (ввести с кл) определить значения a и b , которые позволяют построить эллиптическую группу $EM(a, b)$.
2. Для найденных в задании 1 параметров сгенерировать все элементы эллиптической группы $EM(a, b)$.
3. Реализовать алгоритм обмена ключами для эллиптической группы $EM(a, b)$.
4. Разработать алгоритм цифровой подписи на основе эллиптической группы $EM(a, b)$.

Алгоритм пунктов 1-3 реализован отдельно в виде консольного приложения для более удобной демонстрации.

Алгоритм цифровой подписи с использованием стандартной кривой с известными параметрами реализован в виде учебной веб-страницы.

1. Теоретическая справка

Алгоритмы, основанные на эллиптических кривых, используются в криптографии для обеспечения безопасной передачи данных. Основным объектом — эллиптическая кривая над конечным полем, которая задается уравнением: $y^2 = x^3 + ax + b \pmod{p}$. Условием её корректности является невырожденность: $4a^3 + 27b^2 \neq 0 \pmod{p}$. Элементы группы — это пары (x, y) , удовлетворяющие уравнению, а также специальная точка на бесконечности.

Обмен ключами осуществляется с помощью алгоритма Диффи-Хеллмана в группе точек эллиптической кривой, называемый ECDH. Безопасность алгоритма обеспечивается сложностью задачи дискретного логарифмирования.

2. Поиск параметров эллиптической кривой

Для заданного простого числа M подбираются коэффициенты a и b , такие что $4a^3 + 27b^2 \neq 0 \pmod{M}$.

3. Построение группы точек $EM(a, b)$

После выбора коэффициентов a и b , вычисляются все точки (x, y) , удовлетворяющие уравнению кривой по модулю M . Также включается точка на бесконечности.

4. Обмен ключами на эллиптических кривых (ECDH)

Алгоритм Диффи-Хеллмана с использованием эллиптической группы.

Используются:

1. Общая для всех эллиптическая кривая.
2. Общая базовая точка G (генератор) на этой кривой.

Процесс:

1. Alice:

- Выбирает случайный секретный (приватный) ключ 'a' (целое число).
- Вычисляет свой публичный ключ $A = a * G$ (умножение точки G на скаляр 'a').
- Отправляет A Бобу.

2. Bob:

- Выбирает случайный секретный (приватный) ключ 'b' (целое число).
- Вычисляет свой публичный ключ $B = b * G$.
- Отправляет B Алисе.

3. Вычисление общего секрета:

- Alice вычисляет: $S_A = a * B = a * (b * G) = (a * b) * G$.
- Bob вычисляет: $S_B = b * A = b * (a * G) = (b * a) * G$.

Результат:

S_A и S_B равны ($S_A = S_B = S$). Это и есть общий секретный ключ, который могут использовать Alice и Bob.

Значение S (или его производная) используется для симметричного шифрования.

Безопасность основана на сложности вычисления дискретного логарифма в группе точек эллиптической кривой:

зная G и $A = a * G$ (или G и $B = b * G$), вычислить 'a' (или 'b') практически невозможно для правильно выбранных параметров кривой

5. Алгоритм цифровой подписи

Реализация алгоритма цифровой подписи производилась в виде отдельного приложения. Использовалась кривая secp256k1.

Для эцп используются:

1. Общая для всех эллиптическая кривая:

```
export const curveConfig : {...} = { Show usages
  a: new BN(0),
  b: new BN(7),
  p: new BN(
    "115792089237316195423570985008687907853269984665640564039457584007908834671663"
  )
};
```

Параметры кривой secp256k1

2. Общая базовая точка G (генератор) на этой кривой:

```

export const generatorPoint : {orderN: any, x: any, y: any} = { Show usages
  x: new BN(
    "55066263022277343669578718895168534326250603453777594175500187360389116729240"
  ),
  y: new BN(
    "32670510020758816978083085130507043184471273380659243275938904335757337482424"
  ),
  orderN: new BN(
    "115792089237316195423570985008687907852837564279074904382605163141518161494337"
  )
};

```

Параметры базовой точки G

Генерация подписи (signMessage)

2.1. Входные данные:

- сообщение, представляемое как целое $e = \text{SHA-256}(\text{message})$
- приватный ключ $d \in [1, n-1]$, где n — порядок генератора G

2.2. Шаги алгоритма:

1. Выбор случайного $k \in [1, n-1]$.
2. Вычисление точки $R = k \cdot G$ методом «двойного и сложения».
3. Вычисление $r = R.x \bmod n$.
Если $r = 0$, вернуть к шагу 1 и выбрать новое k .
4. Нахождение обратного элемента $k^{-1} \bmod n$ (расширенный алгоритм Евклида).
5. Вычисление $s = k^{-1} \cdot (e + r \cdot d) \bmod n$.
6. Подпись представляется парами (r, s) .

Проверка подписи (verifySignature)

3.1. Входные данные:

- сообщение $\rightarrow e = \text{SHA-256}(\text{message})$
- подпись (r, s)
- публичный ключ $Q = d \cdot G$

3.2. Шаги алгоритма:

1. Проверить, что $1 \leq r < n$ и $1 \leq s < n$ (иначе — неверная подпись).
2. Вычислить $s^{-1} \bmod n$.
3. Вычислить два скаляра:

$$u_1 = e \cdot s^{-1} \bmod n,$$

$$u_2 = r \cdot s^{-1} \bmod n.$$

4. Вычислить точку $C = u_1 \cdot G + u_2 \cdot Q$.

5. Извлечь $C.x$ и проверить условие
 $r \equiv C.x \pmod{n}$.

Если оно выполняется, подпись считается корректной.

Демонстраций работы программы пункты 1-3:

```
Введите простое число M: 997

==> [1] Поиск параметров эллиптической кривой
Параметры эллиптической кривой: a = 125, b = 297
Найдена кривая:  $y^2 = x^3 + 125x + 297 \pmod{997}$ 

==> [2] Генерация всех точек эллиптической группы EM(a, b)
Найдено 981 точек:
('0', '0')
(2, 377)
(2, 620)
(4, 236)
(4, 761)
(6, 339)
(6, 658)
(8, 238)
(8, 759)
(11, 311)
```

```
(996, 478)
(996, 519)

==> [3] Алгоритм обмена ключами ECDH
Выбрана базовая точка G = (2, 377)
Приватный ключ Alice: 854
Приватный ключ Bob: 380
Открытый ключ Alice: (901, 236)
Открытый ключ Bob: (659, 981)
Общий ключ, вычисленный Alice: (880, 983)
Общий ключ, вычисленный Bob: (880, 983)
```

Ввод простого числа, корректное выполнение операций

```
Введите простое число M: 100
Ошибка: M должно быть простым числом.
```

Ввод не простого числа, выдает ошибку

Демонстрация ЭЦП

Вычислить публичный ключ

❗ **Приватный ключ в ECDSA** - это любое случайное целое число. Обычно оно хранится в секрете. Оно используется чтобы **подписывать сообщения**. **Публичный ключ** - это точка на эллиптической кривой, которой вы можете поделиться с кем угодно, и кто угодно может её использовать, чтобы **проверить ваши подписи**

Приватный ключ *

9678856182658236846306387150978888372669245868729777046147912954

ВЫЧИСЛИТЬ

ГЕНЕРИРОВАТЬ & ВЫЧИСЛИТЬ

Публичный ключ:

```
{
  "x": "4886144836635737249122540510289341541980226621019397066191350482138765038523",
  "y": "40981490105941350656193013217834391329873005335016777319381184712963251133920"
}
```

Формат

☒ JSON decimal ☐ JSON hexadecimal

Подписать сообщение

❗ Вы можете подписать любое **сообщение**, используя ваш **Приватный ключ**. Например, "хочу отправить 1 биткойн на следующий адрес: ...". Любой, кто знает ваш **Публичный ключ**, оригинальное **сообщение** и **подпись**, сможет **проверить**, действительно ли **вы** подписали данное **сообщение**.

Приватный ключ *

8373608826584060511483614962612257007000100189886782266789246468

Сообщение *

хочу отправить 1 биткойн на следующий адрес: 2389du3208dj2349urh293uhr972dw9uyd3h2u9

Тип сообщения

☒ Хэш строки (sha256) ☐ Целое число

ПОДПИСАТЬ

Подпись:

```
{
  "r": "98781887819254460235682098359893258071781755033018783609486953465743858588244",
  "s": "113804090920876471196277209021967902278074816453952923556424570961412941255525"
}
```

Формат

☒ JSON decimal ☐ JSON hexadecimal

Проверить подпись

❗ Имея **Публичный ключ**, **подпись**, и оригинальное **сообщение**, вы можете **проверить**, было ли данное сообщение действительно **подписано** соответствующим **Приватным ключом**, из которого бы **вычислен** данный **Публичный ключ**.

Публичный ключ *

```
{
  "x": "4886144836635737249122540510289341541980226621019397066191350482138765038523",
  "y": "40981490105941350656193013217834391329873005335016777319381184712963251133920"
}
```

Подпись *

```
{
  "r": "98781887819254460235682098359893258071781755033018783609486953465743858588244",
  "s": "113804090920876471196277209021967902278074816453952923556424570961412941255525"
}
```

Сообщение *

хочу отправить 1 биткойн на следующий адрес: 2389du3208dj2349urh293uhr972dw9uyd3h2u9

Тип сообщения

☒ Хэш строки (sha256) ☐ Целое число

ПРОВЕРИТЬ

❗ Подпись некорректна!

Использовались разные приватные ключи для подписи и генерации публичного ключа. Некорректно.

Подписать сообщение

❗ Вы можете подписать любое **сообщение**, используя ваш **Приватный ключ**. Например, "хочу отправить 1 биткойн на следующий адрес: ..." Любая, кто знает ваш **Публичный ключ**, оригинальное **сообщение** и **подпись**, сможет **проверить**, действительно ли **вы** подписали данное **сообщение**.

Приватный ключ *

9678856182658236846306387150978888372669245868729777046147912954

Сообщение *

хочу отправить 1 биткойн на следующий адрес: 2389du3208dj2349urh293uhr972dw9uyd3h2u9

Тип сообщения

☒ Хэш строки (sha256) ☐ Целое число

ПОДПИСАТЬ

Подпись:

```
{
  "r": "88296077059961176747331174350753943601995838244975904057417973888089005156181",
  "s": "48819575537245839285045207998262518121286863689823195677996638867431475360905"
}
```

Формат

☒ JSON decimal ☐ JSON hexadecimal

Проверить подпись

❗ Имея **Публичный ключ**, **подпись**, и оригинальное **сообщение**, вы можете **проверить**, было ли данное сообщение действительно **подписано** соответствующим **Приватным ключом**, из которого бы **вычислен** данный **Публичный ключ**.

Публичный ключ *

```
{
  "x": "4886144836635737249122540510289341541980226621019397066191350482138765038523",
  "y": "40981490105941350656193013217834391329873005335016777319381184712963251133920"
}
```

Подпись *

```
{
  "r": "88296077059961176747331174350753943601995838244975904057417973888089005156181",
  "s": "48819575537245839285045207998262518121286863689823195677996638867431475360905"
}
```

Сообщение *

хочу отправить 1 биткойн на следующий адрес: 2389du3208dj2349urh293uhr972dw9uyd3h2u9

Тип сообщения

☒ Хэш строки (sha256) ☐ Целое число

ПРОВЕРИТЬ

✔ Подпись корректна!

Одинаковые ключи. Корректно.

Проверить подпись

Имея **Публичный ключ**, **подпись**, и оригинальное **сообщение**, вы можете **проверить**, было ли данное сообщение действительно **подписано** соответствующим **Приватным ключом**, из которого бы **вычислен** данный **Публичный ключ**.

Публичный ключ *

```
{
  "x": "4886144836635737249122540510289341541980226621019397066191350482138765038523",
  "y": "40981490105941350656193013217834391329873005335016777319381184712963251133920"
}
```

Подпись *

```
{
  "r": "88296077059961176747331174350753943601995838244975904057417973888089005156181",
  "s": "48819575537245839285045207998262518121286863689823195677996638867431475360905"
}
```

Сообщение *

хочу отправить 91 биткойн на следующий адрес: 2389du3208dj2349urh972dw9uyd3h2u9

Тип сообщения

☒ Хэш строки (sha256) ☐ Целое число

ПРОВЕРИТЬ

Подпись некорректна!

Изменилось сообщение. Нарушена целостность.

Проверить подпись

Имея **Публичный ключ**, **подпись**, и оригинальное **сообщение**, вы можете **проверить**, было ли данное сообщение действительно **подписано** соответствующим **Приватным ключом**, из которого бы **вычислен** данный **Публичный ключ**.

Публичный ключ *

```
{
  "x": "4886144836635737249122540510289341541980226621019397066191350482138765038523",
  "y": "40981490105941350656193013217834391329873005335016777319381184712963251133920"
}
```

Подпись *

```
{
  "r": "ф88296077059961176747331174350753943601995838244975904057417973888089005156181",
  "s": "48819575537245839285045207998262518121286863689823195677996638867431475360905"
}
```

Сообщение *

хочу отправить 1 биткойн на следующий адрес: 2389du3208dj2349urh972dw9uyd3h2u9

Тип сообщения

☒ Хэш строки (sha256) ☐ Целое число

ПРОВЕРИТЬ

Подпись некорректна!

Изменилась подпись. Некорректно

Иные комбинации тест-кейсов, демонстрируют, что алгоритм находит нарушения целостности и аутентичности при передаче цифровой информации.