МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

ЛАБОРАТОРНАЯ РАБОТА №2.10

“Расшифрование криптограммы на основе эллиптических кривых”

по дисциплине

“Информационная безопасность”

Студент:

Алексеев Даниил Иннокентьевич

Группа P34302

Преподаватель:

Рыбаков Степан Дмитриевич

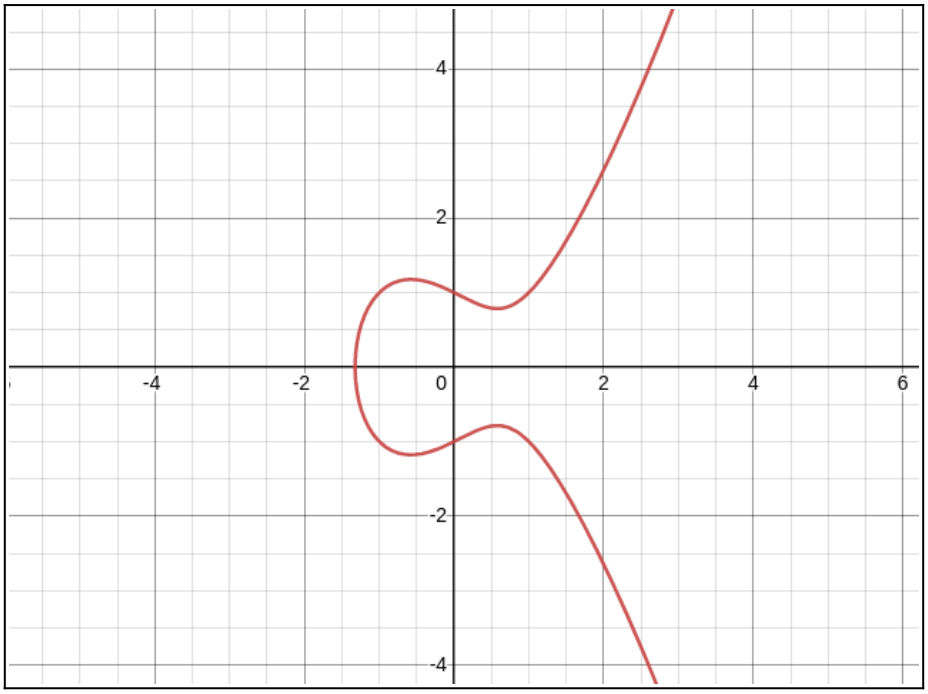
Санкт-Петербург

2023

Цели работы:

- Изучить базовые принципы работы ЭЦП
- Проверить подлинность ECDSA, используя полученные в лабораторной 2.6 знания об эллиптических кривых

вариант	***e***	**Q**	**(r,s)**
16	2	(596,433)	(3,10)



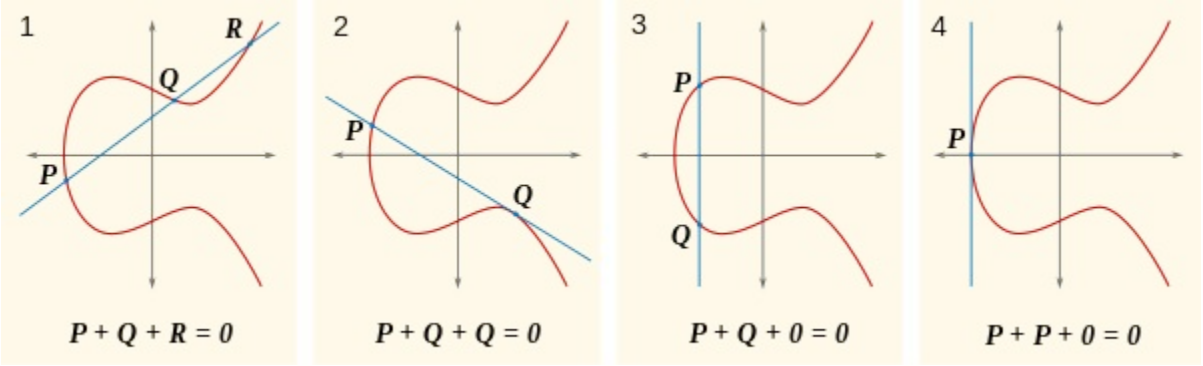
$y^2=x^3-x+1$

Реализация множества (конечного поля) точек на эллиптической кривой и операций над ними

Введем класс точки эллиптической кривой. Точка может являться точкой в бесконечности.

```
data class EllipticCurvePoint (val x: Long, var y:Long, val pointAtInfinity: Boolean){
    fun isEq(p: EllipticCurvePoint): Boolean{
        return p.x == x && p.y == y
    }
}
```

Введем операцию сложения точек на эллиптической кривой:



Над полем целых чисел по модулю p это реализуется так:

```
fun pointAddition(
    p1: EllipticCurvePoint,
    p2: EllipticCurvePoint,
    primeNumber: Long,
    a: Long
): EllipticCurvePoint {
    if (p2.pointAtInfinity) {
        return p1
    }
    if (p1.pointAtInfinity) {
        return p2
    }
    val lambda: Long
    if (mod(p1.x - p2.x, primeNumber) == 0L) {
        if (mod(p1.y - p2.y, primeNumber) == 0L) {
            lambda = (3 * (p1.x) * (p1.x) + a) * invMod(2 * p1.y, primeNumber)
        } else {
            return EllipticCurvePoint(0, 0, true)
        }
    } else {
```

```
        lambda = mod((p2.y - p1.y), primeNumber) * invMod(p2.x - p1.x, primeNumber)
    }

    val x3 = mod(lambda * lambda - p1.x - p2.x, primeNumber)
    val y3 = mod((lambda * (p1.x - x3) - p1.y), primeNumber)
    return EllipticCurvePoint(x3, y3, false)
```

Операция умножения на скалярное n вводится как выполнение сложения точки самой с собой n раз. Оптимизированным способом выполнить умножения является удвоение-сложение:

```
private fun multiplyPoint(point: EllipticCurvePoint, m: Long): EllipticCurvePoint {
    var p = EllipticCurvePoint(0, 0, true)
    var bits = Integer.toBinaryString(m.toInt())
    var i = bits.length
    while (i > 0) {
        p = pointAddition(p, p, primeNumber, a)
        if (bits[bits.length - i] == '1') {
            p = pointAddition(point, p, primeNumber, a)
        }
        i--
    }
    return p
}
```

Вычитание вводится как сложение с инвертированной точкой:

```
private fun subPoints(p1: EllipticCurvePoint, p2: EllipticCurvePoint): EllipticCurvePoint {
    return pointSub(p1, p2, primeNumber, a)
}
```

ECDSA

Генерация подписи

Метод позволяет генерировать подпись (число), зависящее от текста и от секретного ключа, но для проверки подлинности не требующее этот ключ.

В первую очередь рассмотрим процесс создацния ЭЦП.

Следующие значения являются общедоступными и определяются стандартом ECDSA.

Эллиптическая кривая	Генерирующая точка	Порядок генерирующей точки
$E_{751}(-1,1)$ $y^2=x^3-x+1$	(562,89)	13

Эллиптическая кривая задается тремя значениями: простое число p, которое определяет эллиптическую группу, и два коэффициента кубического многочлена. Генерирующая точка - точка принадлежащая этой кривой. Порядок показывает сколько раз генерирующую точку сложить саму с собой, чтобы получить ее же (GP*(n+1) == GP).

Эти значения принадляжат лицу, подписывающему сообщение.

Секретный ключ	Открытый ключ
Случайно простое **число**	**Точка** PrivateKey * GP

Во время сообщения также используется еще одно случайное число K, известное подписателю.

Порядок подписания следующий:

1. Выбрать число **K** из отрезка **[1, n-1]**
2. Вычислить точку **R = GP*K**
3. Вычислить **$r = R_x \bmod n$**
4. Вычислить **$s = (msg + r * PrivateKey) * k^{-1} \bmod n$**

Проверка подписи

Проверить подпись значит узнать использовался ли истинный секретный ключ при создании подписи **(r,s)**. В существующем стандарте ECDSA порядок генерирующей точки описывается числом с десятичной записью в 79 знаков. Поэтому подобрать ключ крайне трудозатратно. Но подпись можно проверить, зная открытый ключ **Q = PrivateKey*GP**:

1. Проверить соответствие **r** и **s** порядку генерирующей точки. При генерации подписи операции выполнялись по модулю n, поэтому должно выполняться $\mathbf{1 \leqslant r,s \leqslant n-1}$
2. Вычислить $\mathbf{u = msg * s^{-1} \bmod n}$
3. Вычислить $\mathbf{v = r * s^{-1} \bmod n}$
4. Получить точку **C = u * GP + v * Q**
5. Проверить выполнение **C.x mod n = r**

Покажем, что точка **C** является точкой **R = GP*K**, используемой при генерации подписи:

$C = u*GP + v*Q$

$C = (msg*s^{-1})*GP + (r*s^{-1})*(GP*PrivateKey)$ – замена u,v,Q

$C = GP*s^{-1}(msg + r*PrivateKey)$ – вынесение общего множителя

$C = GP*[(msg+r*PrivateKey)*k^{-1}]^{-1}(msg + r*PrivateKey)$ – замена s

$C = GP*(msg+r*PrivateKey)^{-1}*k*(msg + r*PrivateKey)$ – избавление от скобки

$C = GP*\cancel{(msg+r*PrivateKey)^{-1}}*k*\cancel{(msg + r*PrivateKey)}=GP*k$ – уничтожение взаимообратных.

Таким образом, действия выполняемые при проверке подписи, определяют точку $C=R=GP*K$.

Код алгоритма проверки подлинности ЭЦП на кривых:

```
fun checkSignature(signa: Pair<Long, Long>): Boolean {
    if(verbose) println("\n--------------------------------")
    if (signa.first < 1 || signa.first >= curveFactor || signa.second < 1 || signa.second >= curveFactor)
return false

    val u1 = e % curveFactor * invMod(signa.second, curveFactor)
    val u2 = signa.first * invMod(signa.second, curveFactor)

    val p1 = multiplyPoint(basePoint, u1)
    if (verbose)
        println(String.format("(%d,%d) * %d = (%d, %d)", basePoint.x, basePoint.y, u1, p1.x, p1.y))

    val p2 = multiplyPoint(publicPoint, u2)
    if (verbose)
        println(String.format("(%d,%d) * %d = (%d, %d)", publicPoint.x, publicPoint.y, u2, p2.x, p2.y))

    val resPoint = pointAddition(p2, p1, primeNumber, a)
    if (verbose)
        println(String.format("(%d,%d) + (%d,%d) = (%d,%d)", p1.x, p1.y, p2.x, p2.y, resPoint.x, resPoint.y))

    if(verbose) println("--------------------------------\n")
    return resPoint.x % curveFactor == signa.first
}
```

```
--------------------------------
(562,89) * 8 = (416, 696)
(596,433) * 12 = (596, 318)
(416,696) + (596,318) = (562,662)
--------------------------------

signature (3,10) is CORRECT
```

Вывод

В ходе выполнения лабораторной работы я укрепил знания об эллиптических кривых и узнал об алгоритмах создания и проверки электронной цифровой подписи на эллиптических кривых.

весь код:

https://github.com/danANDla/InfoSec/tree/master/labs/l2.10_elliptic_curves_digital_signature_algorithm