Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчёт

по лабораторной работе №7

**Криптография с использованием эллиптических кривых.**

Выполнил:

Студент гр. 853504

Пресный В.И,

Проверил:

Олисейчик В.В.

Минск 2021

**Теоретические сведения**

**Математические понятия**

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)

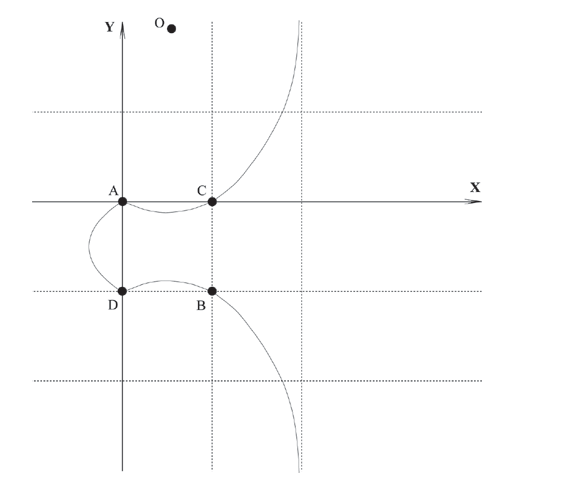


Рисунок 1 – Пример эллиптической кривой с четырьмя точками

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.

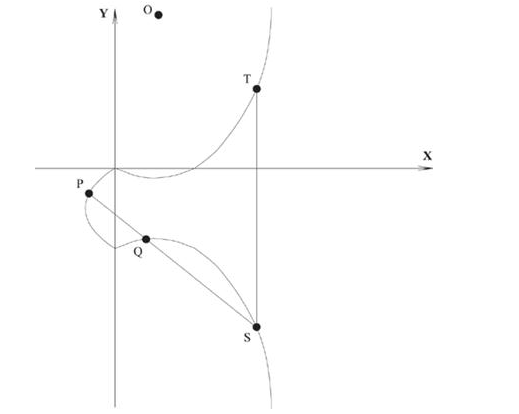


Рисунок 2 – Сложение точек на эллиптической кривой

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.

**Аналог алгоритма Диффи-Хеллмана обмена ключами**

Обмен ключами с использованием *эллиптических кривых* может быть выполнен следующим образом. Сначала выбирается простое число р ≈ 2180 и параметры a и b для уравнения *эллиптической кривой*. Это задает множество точек Ep (a,b). Затем в Ep (a,b) выбирается генерирующая точка G = (x1,y1). При выборе G важно, чтобы наименьшее значение n, при котором n × G = 0, оказалось очень большим простым числом. Параметры Ep (a,b) и G криптосистемы являются параметрами, известными всем участникам.

Обмен ключами между пользователями А и В производится по следующей схеме.

1.           Участник А выбирает целое число nA, меньшее n. Это число является закрытым ключом участника А. Затем участник А вычисляет открытый ключ PA = nA × G, который представляет собой некоторую точку на Ep (a,b).

2.           Точно так же участник В выбирает закрытый ключ nB и вычисляет открытый ключ PB.

3.           Участники обмениваются открытыми ключами, после чего вычисляют общий секретный ключ K

Участник А: K = nA × PB

       Участник В: K = nВ × PА

Следует заметить, что общий секретный ключ представляет собой пару чисел. Если данный ключ предполагается использовать в качестве сеансового ключа для алгоритма симметричного шифрования, то из этой пары необходимо создать одно значение.

**Алгоритм цифровой подписи на основе эллиптических кривых ECDSA**

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.

**Шифрование/дешифрование с использованием эллиптических кривых**

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}

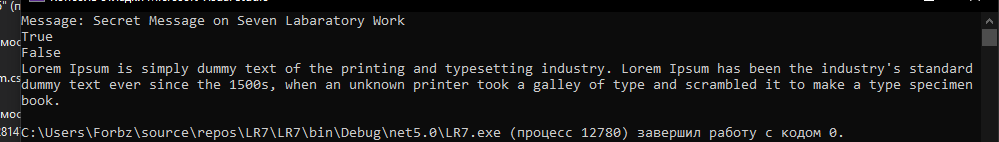
Чтобы дешифровать сообщение, участник В умножает первую координату точки на свой закрытый ключ и вычитает результат из второй координаты:

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.

**Результат работы**

****

**Код приложения**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Numerics;

using System.Text;

using System.Threading.Tasks;

namespace LR7

{

public class EllipticCurve

{

// Field characteristic.

public BigInteger p { get; } = BigInteger.Parse("340282366762482138434845932244680310783");

// Curve coefficients.

public BigInteger a { get; } = BigInteger.Parse("340282366762482138434845932244680310780");

public BigInteger b { get; } = BigInteger.Parse("308990863222245658030922601041482374867");

// Base point.

public BigIntegerPoint G { get; } = new BigIntegerPoint(

BigInteger.Parse("29408993404948928992877151431649155974"),

BigInteger.Parse("275621562871047521857442314737465260675"));

// Subgroup order.

public BigInteger n { get; } = BigInteger.Parse("340282366762482138443322565580356624661");

// Subgroup cofactor.

public int h { get; } = 1;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Numerics;

using System.Text;

using System.Threading.Tasks;

namespace LR7

{

public class ECIE

{

public static List<Tuple<BigIntegerPoint, BigIntegerPoint>> Encryption(string message, EllipticCurve E, BigIntegerPoint PublicKey)

{

List<BigIntegerPoint> plainText = Utility.EmbededTextOnCurve(message, E);

var curve = new EllipticCurve();

var ecdsa = new ECDSA(curve);

List<Tuple<BigIntegerPoint, BigIntegerPoint>> cyphertext = new List<Tuple<BigIntegerPoint, BigIntegerPoint>>();

foreach (var point in plainText)

{

BigInteger k = Utility.RandomIntegerBelow(E.n);

BigIntegerPoint m1 = ecdsa.ScalarMult(k, E.G);

BigIntegerPoint m2 = ecdsa.PointAdd(point, ecdsa.ScalarMult(k, PublicKey));

cyphertext.Add(new Tuple<BigIntegerPoint, BigIntegerPoint>(m1, m2));

}

return cyphertext;

}

public static string Decryption(List<Tuple<BigIntegerPoint, BigIntegerPoint>> cyphertext, BigInteger PrivateKey, EllipticCurve E)

{

StringBuilder plaintext = new StringBuilder();

var curve = new EllipticCurve();

var ecdsa = new ECDSA(curve);

foreach (var text in cyphertext)

{

BigIntegerPoint S = ecdsa.PointAdd(text.Item2, ecdsa.NegatePoint(ecdsa.ScalarMult(PrivateKey, text.Item1)));

plaintext.Append(Utility.GetTextFromProcessedNumber(S.X, 64));

}

return plaintext.ToString();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Numerics;

using System.Text;

using System.Threading.Tasks;

namespace LR7

{

public class BigIntegerPoint

{

public BigInteger X { get; set; }

public BigInteger Y { get; set; }

public BigIntegerPoint(BigInteger x, BigInteger y)

{

X = x;

Y = y;

}

public static BigIntegerPoint GetEmpty()

{

return new BigIntegerPoint(0, 0);

}

public bool IsEmpty() => X == BigInteger.Zero && Y == BigInteger.Zero;

public static BigIntegerPoint Multiply(BigIntegerPoint P, BigInteger n, BigInteger a, BigInteger p)

{

BigIntegerPoint Q = new BigIntegerPoint(0, 0);

BigIntegerPoint R = P;

string n\_bin = Utility.byteArrayToBinaryString(n.ToByteArray());

for (int i = n\_bin.Length - 1; i >= 0; i--)

{

if (n\_bin[i] == '1')

{

Q = Add(Q, R, p);

}

R = Doubling(R, a, p);

}

return Q;

}

public static BigIntegerPoint Add(BigIntegerPoint a, BigIntegerPoint b, BigInteger p)

{

if (a.X == 0 && a.Y == 0)

{

return b;

}

else if (b.X == 0 && b.Y == 0)

{

return a;

}

else if (a.X == b.X)

{

return new BigIntegerPoint(0, 0);

}

else

{

BigInteger lamda = Utility.Modulo((b.Y - a.Y) \* Utility.Reverse\_modulo(b.X - a.X, p), p);

BigInteger cx = Utility.Modulo(lamda \* lamda - a.X - b.X, p);

BigInteger cy = Utility.Modulo(lamda \* (a.X - cx) - a.Y, p);

return new BigIntegerPoint(cx, cy);

}

}

public static BigIntegerPoint Doubling(BigIntegerPoint P, BigInteger a, BigInteger p)

{

BigInteger lamda = Utility.Modulo((3 \* P.X \* P.X + a) % p \* Utility.Reverse\_modulo(2 \* P.Y, p), p);

BigInteger cx = Utility.Modulo(lamda \* lamda - 2 \* P.X, p);

BigInteger cy = Utility.Modulo(lamda \* (P.X - cx) - P.Y, p);

return new BigIntegerPoint(cx, cy);

}

public static BigIntegerPoint Subtract(BigIntegerPoint P, BigIntegerPoint Q, BigInteger p)

{

BigIntegerPoint R = Add(P, Reverse(Q, p), p);

return R;

}

public static BigIntegerPoint Reverse(BigIntegerPoint P, BigInteger p)

{

BigInteger Qx = P.X;

BigInteger Qy = Utility.Modulo(-P.Y, p);

return new BigIntegerPoint(Qx, Qy);

}

}

}