Math 640:348 Prof. Kontorovich
Spring 2015, 5/6 review session

We will illustrate a Pollard rho attack on the Elliptic Curve Discrete Log Problem (ECDLP)

In general our curves will be of the form

E : y^2=f(x)

where f(x)=x^3+Ax+B

In[1]:=
```
f[x_] := x^3 + A x + B;
```

and we will study these curves over finite fields F_p, say

In[2]:=
```
p = 13;
```

For concreteness, lets choose

In[3]:=
```
A = 3; B = 7;
```

We must verify that we have not accidentally chosen a singular curve, that is, the discrimant must not be 0 (mod p)

In[4]:=
```
Mod[4 A^3 + 27 B^2, p]
```

Out[4]= 1

Ok good. Now lets get all the points mod p

In[5]:=
```
check[{x_, y_}] := (Mod[y^2, p] == Mod[f[x], p])
```

This function determines whether y^2=f(x)(mod p) for a given pair {x,y}.

Lets get a list of all points on E/F_p, including the point at infinity, which is the "identity" for the group

In[6]:=
```
pts = {Infinity}; (* start with infinity *)
For[x = 0, x < p, x++,
  For[y = 0, y < p, y++,
    If[check[pt = {x, y}], (* check whether y^2=f(x), *)
      Print[pt];
      AppendTo[pts, pt]; (* if so,
      store the pair {x,y} on our list of points *)
     ];
   ];
 ];
Clear[x, y]
```

```
{3, 2}
{3, 11}
{5, 2}
{5, 11}
{8, 6}
{8, 7}
{9, 3}
{9, 10}
{10, 6}
{10, 7}
{12, 4}
{12, 9}
```

Note that the size of E(F_p), that is, the total number of points, is

In[9]:=
```
sizeE = Length[pts]
```

Out[9]= 13

which tells us that all points will have order 13 .

Now let's write down the addition law (as in Theorem 5.5 in the book):

In[10]:=
```
x[P_] := P[[1]]; (* this gives the x-coordinate of P *)
y[P_] := P[[2]]; (* this gives the y-coordinate of P *)
```

In[12]:=
```
add[P_, Q_] := Module[
    (* this function computes the sum of P and Q, which is called S=(xS,yS) *)
    {S, lambda, xS, yS}, (* these are the variable names to be used*)
    If [ Length[P] ≠ 2,  (* that is, if P==Infinity*)
     S = Q;
     ,
     (* otherwise P ≠ Infinity *)
     If [Length[Q] ≠ 2,  (* that is, if Q==Infinity*)
       S = P;
       ,
       (* otherwise both P and Q ≠ Infinity, so we compute *)
       If[ (x[P] == x[Q]) && (y[P] == Mod[p - y[Q], p]),
         (* if the x coordinates of P and Q are the same but the
          y coordinates are negatives, then their sum is Infinity *)
         S = Infinity;
         ,
         (* otherwise, compute the slope *)
         If[P == Q,
          lambda = Mod[ (3 x[P] ^ 2 + A) PowerMod[(2 y[P]), -1, p] , p];
          (* this is the slope if P=Q *)
          ,
          lambda = Mod[(y[Q] - y[P]) PowerMod[(x[Q] - x[P]), -1, p] , p];
          (* this is the slope of P≠Q *)
         ];
         xS = Mod[lambda ^ 2 - x[P] - x[Q], p];
         yS = Mod[lambda (x[P] - xS) - y[P], p];
         S = {xS, yS};
        ];
      ];
    ];
    (* now return the point S as the answer *)
    S
  ];
```

Given the addition law, we can now make the addition table (of course one would not do this for secure -- that is, very large -- primes p!)

In[13]:=
```
(addTab = Table[

    add[pts[[i]], pts[[j]]]

    , {i, 1, sizeE}, {j, 1, sizeE}]) // MatrixForm
```

Out[13]//MatrixForm=

∞	{3, 2}	{3, 11}	{5, 2}	{5, 11}	{8, 6}	{8, 7}	{9, 3}	{9, 10}	{10, 6}	{10,
{3, 2}	{8, 6}	∞	{5, 11}	{9, 10}	{12, 9}	{3, 11}	{5, 2}	{10, 6}	{12, 4}	{9,
{3, 11}	∞	{8, 7}	{9, 3}	{5, 2}	{3, 2}	{12, 4}	{10, 7}	{5, 11}	{9, 10}	{12,
{5, 2}	{5, 11}	{9, 3}	{3, 11}	∞	{9, 10}	{10, 7}	{8, 7}	{3, 2}	{8, 6}	{12,
{5, 11}	{9, 10}	{5, 2}	∞	{3, 2}	{10, 6}	{9, 3}	{3, 11}	{8, 6}	{12, 9}	{8,
{8, 6}	{12, 9}	{3, 2}	{9, 10}	{10, 6}	{10, 7}	∞	{5, 11}	{12, 4}	{8, 7}	{5,
{8, 7}	{3, 11}	{12, 4}	{10, 7}	{9, 3}	∞	{10, 6}	{12, 9}	{5, 2}	{5, 11}	{8,
{9, 3}	{5, 2}	{10, 7}	{8, 7}	{3, 11}	{5, 11}	{12, 9}	{12, 4}	∞	{3, 2}	{10,
{9, 10}	{10, 6}	{5, 11}	{3, 2}	{8, 6}	{12, 4}	{5, 2}	∞	{12, 9}	{10, 7}	{3, 1
{10, 6}	{12, 4}	{9, 10}	{8, 6}	{12, 9}	{8, 7}	{5, 11}	{3, 2}	{10, 7}	{9, 3}	∞
{10, 7}	{9, 3}	{12, 9}	{12, 4}	{8, 7}	{5, 2}	{8, 6}	{10, 6}	{3, 11}	∞	{9, 1
{12, 4}	{8, 7}	{10, 6}	{12, 9}	{10, 7}	{3, 11}	{9, 10}	{8, 6}	{9, 3}	{5, 2}	{3,
{12, 9}	{10, 7}	{8, 6}	{10, 6}	{12, 4}	{9, 3}	{3, 2}	{9, 10}	{8, 7}	{3, 11}	{5, 1

Note here again that each row is a permutation of the points (as is each column).

Finally, we'll need the analog of "fast powering", that is, the "double and add" algorithm, as described in Table 5.3 in the book

In[14]:=
```
fastadd[P_, n_] := Module[
  (* this adds P to itself n times *)

  {Q, R, m}, (* these are the variables to be used *)
  (* First set Q=P and R=Infinity and m=n *)
  Q = P;
  R = Infinity;
  m = n;

  While[m > 0,
   If[Mod[m, 2] == 1, (* If n is odd *)
    R = add[R, Q];
   ];
   Q = add[Q, Q];
   m = Floor[m / 2];
  ];

  R (* At the end, R = n P *)
 ]
```

Let's check that it works: add P=(3,2) to itself 13 times

In[15]:=
```
P = {3, 2};

Table[
 fastadd[P, j]
 , {j, 1, sizeE}]
```

Out[16]=
```
{{3, 2}, {8, 6}, {12, 9}, {10, 7}, {9, 3}, {5, 2},
 {5, 11}, {9, 10}, {10, 6}, {12, 4}, {8, 7}, {3, 11}, ∞}
```

Great, so P is a point of order 13=#(E/F_p); that is, it's a generator of all of E/F_p. (Recall generators do not exist in general for elliptic curves (!), though points of large order do.)

That means we can take any other point, say

In[17]:=
```
Q = {5, 11};
```

and try to find n so that Q=nP. (Of course we can look at our table above and see that n=7 works. Again since P has order 13, the full answer is: n=7(mod 13).)

Let's use the Pollard rho algorithm to solve this ECDLP. The expected runtime is about sqrt[E/F_p]=sqrt[13]=4, so it shouldn't take long. First we need a "random" function, so why not use the analogous one to the classical DLP as in Chapter 4.5.2 of the book.

In[18]:=
```
f[{R_, alpha_, beta_}] :=
 Module[{R1, alpha1, beta1}, (* the point R is sent to a "random" point R1,
  and alpha/beta's are incremented *)

  If[Length[R] ≠ 2 (* that is, if R=Infinity, add P *),
   R1 = P;
   alpha1 = Mod[alpha + 1, sizeE]; (* remember that both P and Q
    have order a divisor of 9, so alpha and beta are numbers mod 9 *)
   beta1 = beta;
   ,
   (* otherwise test the three ranges of x_R *)
   If[x[R] < p / 3,
     R1 = add[P, R]; (* if in the low range, add P *)
     alpha1 = Mod[alpha + 1, sizeE]; (* and increment alpha *)
     beta1 = beta;
     ,
     (* or else if we're in the middle range, double R *)
     If[x[R] < 2 p / 3,
       R1 = add[R, R];
       alpha1 = Mod[2 alpha, sizeE]; (* double alpha *)
       beta1 = Mod[2 beta, sizeE]; (* double beta *)
       ,
       (* finally, in the high range, add Q *)
       R1 = add[R, Q];
       alpha1 = alpha; (* leave alpha alone *)
       beta1 = Mod[beta + 1, sizeE]; (* increment beta *)
      ];
    ];
  ];
  {R1, alpha1, beta1} (* is the new triple *)
 ]
```

We initialize things by

In[19]:=
```
X = Infinity; alpha = 0; beta = 0;
```

for the tortoise and

In[20]:=
```
Y = Infinity; gamma = 0; delta = 0;
```

for the hare. The initial index is

In[21]:=
```
i = 0;
```

Now we increment (as in our tables in the homework, X_i, alpha_i, beta_i, Y_i, gamma_i, delta_i)

In[22]:=
```
i = 1;
{{X, alpha, beta}, {Y, gamma, delta}} =
 {f[{X, alpha, beta}], f[f[{Y, gamma, delta}]]}
(* the tortoise steps once while the hare steps twice *)
X == Y
```

Out[23]= {{{3, 2}, 1, 0}, {{8, 6}, 2, 0}}

Out[24]= False

In[25]:=
```
i = 2;
{{X, alpha, beta}, {Y, gamma, delta}} =
 {f[{X, alpha, beta}], f[f[{Y, gamma, delta}]]}
X ==
 Y
```

Out[26]= {{{8, 6}, 2, 0}, {{8, 7}, 4, 1}}

Out[27]= False

In[28]:=
```
i = 3;
{{X, alpha, beta}, {Y, gamma, delta}} =
 {f[{X, alpha, beta}], f[f[{Y, gamma, delta}]]}
X ==
 Y
```

Out[29]= {{{10, 7}, 4, 0}, {{12, 9}, 8, 3}}

Out[30]= False

In[31]:=
```
i = 4;
{{X, alpha, beta}, {Y, gamma, delta}} =
 {f[{X, alpha, beta}], f[f[{Y, gamma, delta}]]}
X ==
 Y
```

Out[32]= {{{8, 7}, 4, 1}, {{10, 7}, 8, 5}}

Out[33]= False

In[34]:=
```
i = 5;
{{X, alpha, beta}, {Y, gamma, delta}} =
 {f[{X, alpha, beta}], f[f[{Y, gamma, delta}]]}
X ==
 Y
```

Out[35]= {{{10, 6}, 8, 2}, {{10, 6}, 3, 12}}

Out[36]= True

Aha! We have a collision, X=Y. Let's look at what this collision means. It tells us

that:

8P+2Q = 3P+12Q

so

10Q = 5P.

Lets just check:

In[37]:=
```
fastadd[Q, 10]
fastadd[P, 5]
```

Out[37]= {9, 3}

Out[38]= {9, 3}

Great, but what do we do now? We want just Q on one side, and we have 10Q. But we know that 13Q = Infinity, since that's true for any point on E (because the number of points is 13).

In[39]:=
```
fastadd[Q, sizeE]
```

Out[39]= ∞

So we need the multiplicative inverse of 10 mod 13!

In[40]:=
```
PowerMod[10, -1, 13]
```

Out[40]= 4

By hand, you would do this with the extended Euclidean algorithm. What this tells us is that we should add the equation

10Q = 5P

to itself 4 times. (Like taking a classical DLP equation and raising both sides to the 4th power.) We get:

4*10Q = 4*5P

or

40Q = 20 P

or, since 40=3*13+1

Q = 20 P,

and since 20=13+7, we recover the value of n

Q = 7P.

So n=7!

Always check your work:

In[41]:=
```
fastadd[P, 7]
Q
```

Out[41]= {5, 11}

Out[42]= {5, 11}