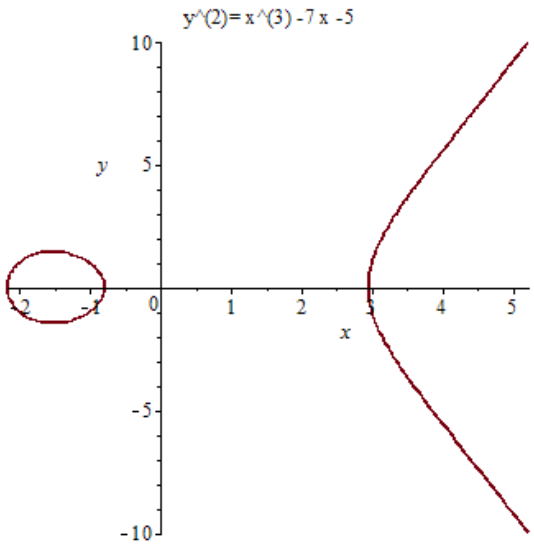
ELLIPTIC CURVE CRYPTOGRAPHY USING MAPLE

Joseph Fadyn
Kennesaw State University
1100 South Marietta Parkway
Marietta, Georgia 30060
jfadyn@spsu.edu

An elliptic curve is one of the form: $y^2 = x^3 + ax + b$ where the coefficients a and b are chosen from some field K. All of our work in this paper will be done with $K=\mathbf{GF}(p)=<\mathbf{Z}_p, +, \times>$. We require that the cubic $x^3 + ax + b$ does not have repeated roots in $\mathbf{Z}_p$ which is equivalent to the condition that $4a^3 + 27b^2 \neq 0 \pmod p$. As an example, consider the graph of: $y^2 = x^3 - 7x - 5$ over $\mathbf{R}$:



If we consider this elliptic curve over the field $\mathbf{Z}_{13}$, then we obtain a *finite* set of points: { (3, ±1), (6, 0), (7, ±9), (8, ±3), (11, ±1), (12, ±1) }. If we add to this set the so-called point at infinity, denoted by **O**, we obtain a set denoted by $E_{13}(-7,-5)$ which contains a total of 12 points. $E_{13}(-7,-5)$ can be made into an abelian (commutative) group. For a general elliptic curve E and prime p we have (see [3]): To begin we define: $P + \mathbf{O} = P$ for all P. Next, if $P_1 = (x_1 , y_1)$ and $P_2 = (x_2 , y_2)$ are points on **E** with $P_1 , P_2 \neq \mathbf{O}$, define $P_1 + P_2 = P_3 = (x_3 , y_3)$ by:

1. If $x_1 \neq x_2$, then $x_3 = m^2 - x_1 - x_2$, $y_3 = m(x_1 - x_3) - y_1$, where $m = (y_2-y_1)/(x_2-x_1)$.
2. If $x_1 = x_2$ but $y_1 \neq y_2$. Then $P_1 + P_2 = \mathbf{O}$.
3. If $P_1 = P_2$ and $y_1 \neq 0$ then $x_3 = m^2 - 2x_1$, $y_3 = m(x_1 - x_3) - y_1$, where $m = (3x_1^2 + a) / (2y_1)$.
4. If $P_1 = P_2$ and $y_1 = 0$ then $P_1 + P_2 = \mathbf{O}$.

I now present a number of Maple procedures to do basic elliptic curve computations. All computations will be done modulo a prime $p > 3$:

1. ecnopts : Computes the number of points on $y^2 = x^3 + ax + b$:

```
ecnopts := proc(a, b, p)
    local f, fla, ct, i, xtm; with(numtheory) :
  fla := 0 : f := x → x^3 + a·x + b : ct := 0 :
  for i from 0 to p − 1 do
   xtm := modp(f(i), p) :
     if xtm = 0 then fla := 1 : fi;
     if legendre(xtm, p) = 1 then ct := ct + 1 : fi;
  end do;
if fla = 0 then ct := 2·ct + 1 : else ct := 2·ct + 2 : fi;
print (Total Points); return ct; end proc;
```

2. ecpoints: Produces a complete or partial list of points on $y^2 = x^3 + ax + b$:

```
ecpoints := proc(a, b, p, n)
    local f, fla, ct, j, jj, xtm, i, csqr, pointslist ;
       with(numtheory) :
  fla := 0 : j := 1 : jj := 0 : ct := 0 :
  f := x → x^3 + a·x + b :
  pointslist := [ ] :
  for i from 0 to n do
   xtm := modp(f(i), p) :
     if xtm = 0 then fla := 1 : pointslist := [op(pointslist), [i, 0]] : ct
     := ct + 1 : fi;
          csqr := msqrt(xtm, p) :
     if csqr ≠ FAIL and xtm ≠ 0 then pointslist := [op(pointslist),
     [i, csqr]] : pointslist := [op(pointslist), [i , p − csqr]] : ct := ct
     + 2 : fi;
  end do;
 ct := ct + 1 :
print (Total Points);
return pointslist , ct ;
end proc;
```

3. mdexpec(ba,e,n,a,b). Here ba = $[x_1,y_1]$. This program finds the value of e·ba = e·$[x_1,y_1]$ modulo n on $y^2 = x^3 + ax + b \pmod n$ using a form of modular exponentiation:

```
mdexpec := proc(ba, e, a, b, n)
    local d, m, ee;
    global z;
  z := [∞,∞] : m := ba; ee := e :
  d := [0, 0] : Array(d);
  while ee ≠ 0 do
    d := [floor(ee/2), ee - 2· floor(ee/2)];
    if d[2] = 1 then ecsump(z, m, a, b, n) : z := sm; print(`z=`z) : fi;
    ee := d[1] :
    ecsump(m, m, a, b, n) :
    m := sm;
  end do;
 return(z)
end proc;
```

4. ecorder: Finds the order of c = $[x_1,y_1]$ in $E_{pr}(a,b)$ where $y^2 = x^3 + ax + b$ and pr >3 is a prime. The number of points in $E_{pr}(a,b)$ must be known to be n:

```
ecorder := proc(c, a, b, n, pr)
    local t, aone, nm, F, i ;
    t := n : F := ifactors(n); nm := nops(F[2]) :
      for i from 1 to nm do
   t := t / (F[2, i][1]^F[2, i][2]);
  mdexpec(c, t, a, b, pr); aone := z;
   while aone ≠ [∞,∞] do
    mdexpec(aone, F[2, i][1], a, b, pr); aone := z;  t := t·F[2, i][1];
  end do;
 end do;
  return(`order is `,t); end proc;
```

5. ecsump : Finds the sum of points r and s on $y^2 = x^3 + ax + b$ where r = $[x_1,y_1]$ and s = $[x_2,y_2]$:

```
ecsump := proc(r, s, a, b, p)
    local xone, xtwo, xthree, yone, ytwo, ythree, inf, λ, A, B, tmp;
    global sm; with(numtheory) : xone := r[1] : yone := r[2] : xtwo
      := s[1] : ytwo := s[2] : inf := [∞, ∞];
   if [xone, yone] = inf then sm := [xtwo, ytwo] : goto(12) : fi;
   if [xtwo, ytwo] = inf then sm := [xone, yone] : goto(12) : fi;
  xone := modp(r[1], p) : yone := modp(r[2], p) : xtwo
      := modp(s[1], p) : ytwo := modp(s[2], p) : A := modp(a, p) : B
      := modp(b, 2) :
  if xone = xtwo and ( yone = p − ytwo or ytwo = p − yone) then sm
      := inf : goto(12) : fi;
  if xone ≠ xtwo then tmp := (xtwo − xone)^-1 mod p : λ
      := modp((ytwo − yone)·tmp, p) : xthree := modp(λ^2 - xone
     - xtwo, p) :
   ythree := modp((xone − xthree)·λ - yone, p) : sm := [xthree,
     ythree] : goto(12) : fi;
  if xone = xtwo and yone ≠ 0 then tmp := (2·yone)^-1 mod p : λ
      := modp((3·xone^2 + a)·tmp, p) : xthree := modp(λ^2 - xone
     − xtwo, p) :
   ythree := modp((xone − xthree)·λ - yone, p) : sm := [xthree,
     ythree] : goto(12) : fi;
  if xone = xtwo and ytwo ≠ 0 then tmp := (2·ytwo)^-1 mod p : λ
      := modp((3·xtwo^2 + a)· tmp, p) : xthree := modp( λ^2 - xone
     - xtwo, p) :
   ythree := modp((xone − xthree)·λ - ytwo, p) : sm := [xthree,
     ythree] : goto(12) : fi;
  if xone = xtwo and yone = ytwo and yone = 0 then sm := inf : goto(12) :
     fi;
  12 : return (sm); end proc;
```

I now describe a basic form of elliptic curve encryption based on the ElGammal cryptosystem [5]. For this, I follow the discussion in [1]. We begin with a plaintext message M encoded as the x-coordinate of the point P_M which lies on the curve $y^2 = x^3 + ax + b \pmod p$. We choose a point G on the curve whose order n in $E_p(a,b)$ is a large prime number. Both $E_p(a,b)$ and G are made public. Each user (Alice and Bob) selects a private key n_A and n_B and forms the public keys $P_A = n_A G$ and $P_B = n_B G$. If Alice (A) encrypts P_M to send to Bob (B), she chooses a random positive integer $k < n$ and sends Bob the ciphertext *pair* of points:
$P_C = \{ kG , (P_M + kP_B)\}$. Upon receiving the ciphertext message P_C , Bob recovers the original plaintext P_M via the computation:
$(P_M + kP_B) – [n_B (kG)] = (P_M + kn_BG) - [n_B (kG)] = P_M$.
Note that a cryptanalyst would know G and also kG (which appears in P_C), so if he could find k from this information, he could decode $P_M + kP_B$ (because he also knows P_B) as follows: $(P_M + kP_B) – kP_B = P_M$. Finding k from G and kG is the elliptic curve discrete logarithm problem: $\log_G (kG) = k$ which is considered to be a computationaly

intractable problem for large values of k and a "generator" point G with large order n. The encoding and decoding schemes are implemented in Maple as:

```
ecencryp := proc(pm, pb, ge, k, a, b, p)
    local kg, kpb, pmkpb, pe;
    mdexpec(ge, k, a, b, p); kg := z; mdexpec(pb, k, a, b, p);
     kpb := z;
     ecsump(pm, kpb, a, b, p); pmkpb := sm; pe := [kg, pmkpb];
    return(`Encrypted Point Is: `, pe); end proc;

ecdecryp := proc(kgpmkpb, nb, a, b, p)
    local pm, nbkg;
     mdexpec(kgpmkpb[1], nb, a, b, p); nbkg := z;
     nbkg[2] := modp(-1·nbkg[2], p);
    ecsump(kgpmkpb[2], nbkg, a, b, p); pm := sm;
return(`Plaintext Point Is: `, pm);end proc;
```

EXAMPLE 1: A naive approach:

We give an example of encryption and decryption using Maple along with the use of some of the supporting procedures written above for a small prime and a short message. We use the prime p = 9883 and $y^2 = x^3 + 765\,x + 871$. We encode the letters A – Z by the shift scheme: E = 1, F = 2, G = 3,… C = 0, D = 1. The message which Alice will send to Bob is "LIKE" which corresponds to plaintext ciphers as 8571. To check that 8571 is the x-coordinate of a point on the elliptic curve we define:

$f(x) = x^3 + 765 \cdot x + 871.$

```
> with(numtheory);
```

```
nextprime(9876);
```

9883

```
f := x→x^3 + 765·x + 871;
```

$x \to x^3 + 765\,x + 871$

Then modp(f(8571), 9883) gives 5791. Then msqrt(5791, 9883) produces the y-coordinate which is 3277:

```
modp(f(8571), 9883);
```

5791

So our message point PM is [8571, 3277]. To find the order of the group E9883 (765, 871), we run ecnopts(765, 871, 9883) which gives 9827 :

```
ecnopts(765, 871, 9883);
```

9827 *Total Points.*

To see some of the actual points in E9883 (765, 871) we can use ecpoints. For example if we want to see the first (approximately) 40 points, we run:

ecpoints(765, 871, 9883, 40);

43 *Total Points:*

[[0, 2688], [0, 7195], [3, 3998], [3, 5885], [4, 2091], [4, 7792], [5, 4051], [5, 5832], [7, 2813], [7, 7070], [9, 4340], [9, 5543], [15, 3743], [15, 6140], [17, 4136], [17, 5747], [19, 2768], [19, 7115], [20, 8434], [20, 1449], [21, 3325], [21, 6558], [23, 3612], [23, 6271], [24, 6413], [24, 3470], [26, 7667], [26, 2216], [27, 203], [27, 9680], [28, 253], [28, 9630], [32, 9217], [32, 666], [34, 340], [34, 9543], [36, 3121], [36, 6762], [37, 8184], [37, 1699], [39, 2604], [39, 7279], [∞, ∞]]

For a base point G we choose a random x, $0 \le x \le 9883$. For example, if x = 7 then modp(f(7), 9833) gives 6569 and then msqrt(6569, 9883) yields the y-coordinate which is 2813:

modp(*f*(7), 9833);

6569

msqrt(6569, 9883);

2813

Our prospective G is [7, 2813]. Next we find the order of G in the group E9883 (765, 871) by running ecorder([7, 2813], 765, 871, 9827, 9883) which produces "9827 is the order":

> *ecorder*([7, 2813], 765, 871, 9827, 9883);

9827 *is the order*

If Bob's secret key is nB = 873, then his public key is PB = nBG, so that: PB = 873 • [7, 2813]. For this computation we use: mdexpec([7, 2813], 873, 9883, 765, 871) to obtain PB = [7516, 1555] which is Bob's public key:

mdexpec([7, 2813], 873, 765, 871, 9883);

[7516, 1555]

If Alice's random number k (which she selects) if k = 2477, then the encrypted message which Alice sends to Bob can be obtained by: ecencrypt([8471, 3277], [7516, 1555], [7, 2813], 2477, 765, 871, 9883). This produces the ciphertext pair: PC = [[4225, 3276], [27, 203]]:

ecencryp([8571, 3277], [7516, 1555], [7, 2813], 2477, 765, 871, 9883);

Encrypted Point Is: [[4225, 3276], [27, 203]]

Now Bob may decrypt PC by using ecdecryp(kg, pmkpb, nb, a, b, p) which in our example is: ecdecryp([4225, 3276], [27, 203], 873, 765, 871, 9883). This outputs the plaintext point PM as [8571, 3277]:

> *ecdecryp*([[4225, 3276], [27, 203]], 873, 765, 871, 9883);

Plaintext Point Is: [8571, 3277]

So the message is in the x-coordinate, which is 8571 which we map back into "LIKE".

We now produce a somewhat more sophisticated approach which will handle much larger primes. For this, we will need a number of additional procedures.

```
numtopointnew := proc(x)
 local kappa, xp, c, yp, j;
 global shift;
 shift := [op(shift)];
  kappa := 2^8; j := 0;
 if p - x > kappa then
 for xp from x + j to x + kappa - 1 do
  c := xp^3 + a·xp + b mod p;
 yp := numtheory[msqrt](c, p);
    j := j + 1;
 if yp ≠ FAIL then
   shift := [op(shift), j - 1];
     return [modp(xp, p), yp]
 fi;
 od;
 else
 for xp from x to x - kappa - 1 do
  c := xp^3 + a·xp + b mod p;
 yp := numtheory[msqrt](c, p);
 j := j - 1;
 if yp ≠ FAIL then
   shift := [op(shift), j + 1];
     return [modp(xp, p), yp]
 fi;
 od;
 fi;
 print("Kappa Value Too Small. Please Increase Kappa Value.");
  return FAIL
 end proc;
```

```
sendmessagenew := proc(T)
 local i, pl, c, C, shift;
 global ii; shift;
 C := [ ]; ii := 1; shift := [ ];
 for i from 1 to nops(T) do
   pl := numtopointnew(T[i]);
   c := ecencrypt(pl, pb, G, k, a, b, p);
  ii := ii + 1;
 C := [op(C), c];
  end do;
 return (C);
end proc;
```

```
receivemessagenew := proc(C)
  local T, t, i, pl, decipher2, decipher3;
  global shift, pla;
    pla := [ ];
  for i from 1 to nops(C) do
   pl := ecdecrypt(C[i], nb, a, b, p);
   pl[1] := pl[1] - shift[i];
     pla := [op(pla), pl[1]];
    od;
 decipher2 := convert(pla, base, p, 256);
   decipher3 := convert(decipher2, bytes);
    return(decipher3);
end proc;
```

Example 2: To illustrate the use of these procedures, we will use the NIST (National Institute of Standards and Technology) standard curve called P-192 along with the suggested base point and parameters for this curve (see [4]).

```
p
      := 6277101735386680763835789423207666416083908700390324961279;

x
      := 602046282375688656758213480587526111916698976636884684818;

 y
       := 174050332293622031404857552280219410364023488927386650641;

b
      := 2455155546008943817740293915197451784769108058161191238065

 a := -3;

G := [x, y];
```

We now generate random values for na, nb and k as follows:

```
  tmp := rand(1..10^60);
na := tmp( );
 na
      := 5783437625944840362920103694094366060602686903018669884702 78

tmp := rand(1..10^60);
nb := tmp( );
 nb
      := 6025570818056360529994278535263629338023996993620639183883 86
```

$tmp := rand(1..10^{60});$

$k := tmp();$

k

$:= 112803239119634755360157120101891514484381580384478933592666$

We find Bob's public key:

$pb := mdexpec(G, nb, a, b, p);$

[
294101517035892746266140741872855505275489857101040842
1892,
336185249762276871910174017961999048611977575502150062
3681]

We now illustrate the encrypting/decrypting process:

$plaintext :=$ "The derivative of a sum is the sum of the derivatives."

"The derivative of a sum is the sum of the derivatives."

$List1 := convert(plaintext, bytes);$

[84, 104, 101, 32, 100, 101, 114, 105, 118, 97, 116, 105, 118, 101, 32,
111, 102, 32, 97, 32, 115, 117, 109, 32, 105, 115, 32, 116, 104, 101,
32, 115, 117, 109, 32, 111, 102, 32, 116, 104, 101, 32, 100, 101,
114, 105, 118, 97, 116, 105, 118, 101, 115, 46]

$List2 := convert(List1, base, 256, p);$

[
574610914663818795995364656592186440306006410118601430
7734,
238977468639178451336949567475222274321146054119956738
2198, 51073158375796]

$shift := [\ ];$

[ ]

$E := sendmessagenew(List2)$

```
[[[165743369629337116395029041102480291710571318997070020521 2,
  215514931101502721870901995765576671889605719015554871095 3],
 [581051197826843666874157108488579504216681713920129957095 8,
  688331379519775905694374186455965492813194229483493061219]],
 [[165743369629337116395029041102480291710571318997070020521 2,
  215514931101502721870901995765576671889605719015554871095 3],
 [315924925592844400859490220961084376460919727907991777901 9,
  212901634766163670519807909415273728188287195142713220944 1]],
 [[165743369629337116395029041102480291710571318997070020521 2,
  215514931101502721870901995765576671889605719015554871095 3],
 [490290437374372963603902944055317325544229661660340648197 3,
  318735561513822507548395921610540145219628636798722719596 9]]]
```

receivemessagenew(*E*);

"The derivative of a sum is the sum of the derivatives."

References

1. Chouinard, Jean Yves, *Notes on Elliptic Curve Cryptography*, 2002, http://www.site.uottawa.ca/~chouinar/Handout_CSI4138_ECC_2002.pdf
2. J.N. Fadyn, *Basic Elliptic Curve Cryptography using the TI-89 and Maple,* Proceedings of the ICTCM 2014.
3. Washington, L.C., *Elliptic Curves : Number Theory and Cryptography, Second Edition* , CRC Press, 2008.
4. Recommended Elliptic Curves for Federal Government Use, July, 1999, http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf