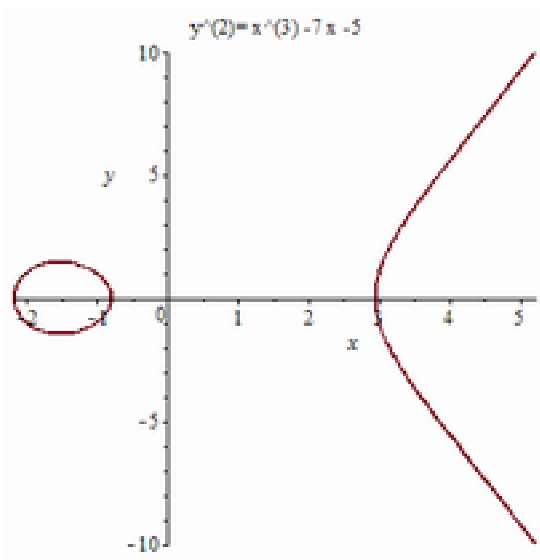
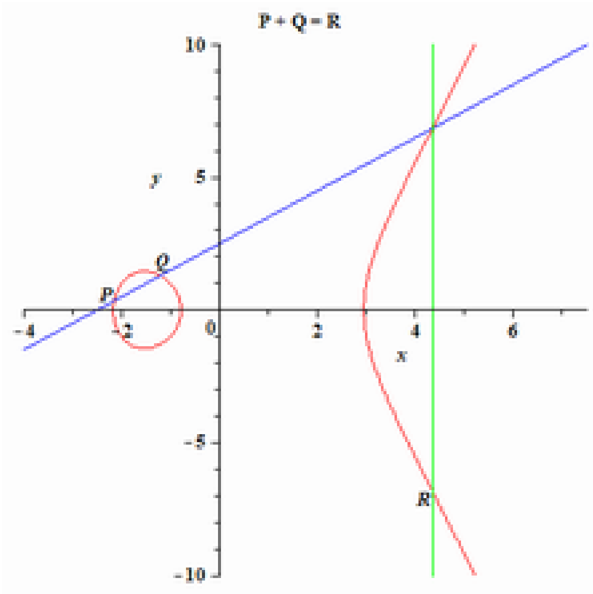
BASIC ELLIPTIC CURVE CRYPTOGRAPHY USING THE TI-89 AND MAPLE

Joseph Fadyn
Southern Polytechnic State University
1100 South Marietta Parkway
Marietta, Georgia 30060

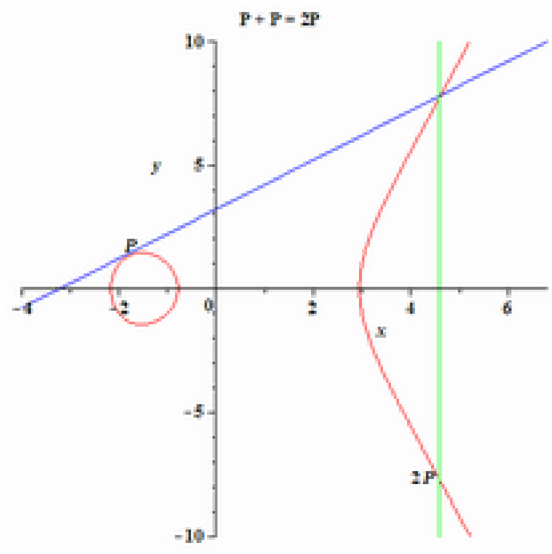
An elliptic curve is one of the form: $y^2 = x^3 + ax + b$ where the coefficients a and b are chosen from some field K. The field may be (among others) the real numbers, the rational numbers, or a finite field **GF**(q), where $q = p^n$ where p is prime and n is a positive integer. All of our work in this paper will be done with K=**GF**(p)=$<\mathbf{Z}_p ,+,\times>$. The form $y^2 = x^3 + ax + b$ is called the Wierstrass form of an elliptic curve. 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 **R** :



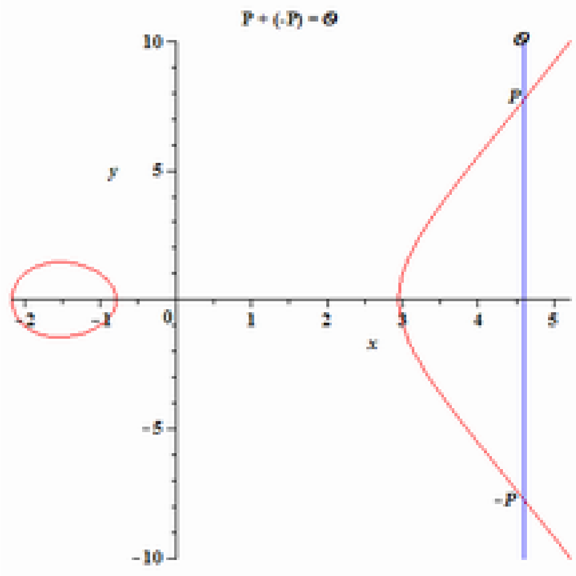
If we consider this elliptic curve over the field $\mathbf{Z}_{13}$, then we obtain a *finite* set of points: { (3, ±1), (6, 0), (7, ±9), (8, ±3), (11, ±1), (12, ±1) }. If we add to this set the so-called point at infinity, denoted by **O**, we obtain a set denoted by $E_{13}(-7,-5)$ which contains a total of 12 points. $E_{13}(-7,-5)$ can be made into an abelian (commutative) group. For insight into how the group operation should be defined we look at the geometry of elliptic curves over **R**. Let **E** denote the elliptic curve. For points $P(x_1,y_1)$ and $Q(x_2,y_2)$ on **E** with $x_1 \neq x_2$, we define P + Q = R to conform with the geometry:



If Q = P, then we deifine P + P = 2P to conform with the geometry:



Finally, if Q = -P, where –P = (x_1,-y_1), then we define P + (-P) = **O**:



It is fairly easy to formulate these rules into algebraic form. See, for example [5]. To begin we define: P + **O** = P for all P. Next, following [5], if $P_1 = (x_1 , y_1)$ and $P_2 = (x_2 , y_2)$ are points on **E** with $P_1 , P_2 \neq$ **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 $P_1 + P_2 =$ **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 $P_1 + P_2 =$ **O**.

I now present a number of TI-89 programs to do basic elliptic curve computations. All computations will be done modulo a prime p > 3:

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

ecnopts(a,b,p): Prgm: DelVar x, aa, bb, xtm, ct: a -> aa: b -> bb: 0 -> fla: x^3 + aa*x + bb -> f(x): 0 -> ct: For I, o, p-1: mod(f(i), p) -> xtm: If xtm = 0 Then: 1 -> fla: End If: jacobi(xtm, p): If k = 1 Then 1 + ct -> ct: EndIf: End For: If fla = 0 Then 2*ct -> ct: Else: 2*ct – 1 -> ct: EndIf: Disp "Total Points:" : Disp ct: Disp "Does Not Include O" : Disp "The Point At Infinity." : EndPrgm

2. jacobi(a,b) returns 1 if a is a quadratic residue mod b and -1 if a is not:

jacobi(a,b): Prgm: mod(a, b) -> a: If a = 0 Then: 1 -> k: Goto stpe: EndIf: 0 -> ta[1]: 1 -> ta[2]: 0 -> ta[3]: -1 -> ta[4]: 0 -> ta[5]: -1 -> ta[6]: 0 -> ta[7]: 1 -> ta[8]: 1 -> k: Lbl stp3: 0 -> v: While mod(a,2) = 0: 1 + v -> v: a/2 -> a: EndWhile: If mod(v,2) = 1 Then ta[mod(b,8) + 1]* k -> k: EndIf: If mod(a,4) ≠1 and mod(b,4) ≠ 1 Then: -1*k -> k: EndIf: abs(a) -> r: mod(b, r) ->a: r -> b: If a = 0 Then: Goto stpe: EndIf: Goto stp3: Lbl stpe: Disp k: EndPrgm

3. ecptsnew(a,b,p) produces a complete list of points on $y^2 = x^3 + ax + b$:

ecptsnew(a,b,p): Prgm: ClrIO: DelVar x, ff, ep, cut, fla, kk, nnn: Disp "Number of points? ": Request "Enter p – 1 for All", nn: expr(nnn) -> nnn: a -> aa: b -> bb: randMat(nnn,2) -> ep: x^3 + aa*x + bb -> ff(x): 1 -> j: 0 -> jj: 0 -> cnt: 0 -> fla: While j < nnn + 1 and jj < p: mod(ff(jj),p) -> xtm: If xtm = 0 Then: 1 -> fla: EndIf: jacobi(xtm, p): jj + 1 -> jj: If k = 1 Then: sqrtmdpm(xtm, p): jj – 1 -> ep[j,1]: "±" & string(rot[1]) -> ep[j,2]: Disp "A Point Is:": Disp "(" & string(ep[j,1]) & "," & ep[j,2] & ")" : j+1 -> j: cnt + 1 -> cnt: Pause: EndIf: EndWhile: cnt -> ct: If fla = 0 Then: 2*cnt -> cnt: Else: 2*cnt – 1 -> cnt: EndIf: randMat(2, ct) -> ecp: For I, 1, ct: ep[i,1] -> ecp[1, i] : ep[i,2] -> ep[2,i]: EndFor: 0 -> anw: Disp "Matrix Without ± ?": Disp "Enter 1 if Yes, 0 if No": Input anw: If anw = 0 Then: Goto stpf: EndIf: 1 -> I: 1 ->j: randMat(2,cnt) -> ecpm: While i ≤ cnt: ecp[1,j] -> ecpm[1,i]: abs(expr(ecp[2,j]) -> ecpm[2,i]: If ecpm[2,i] = 0 Then: Goto stpe: EndIf: i + 1 -> i : ecp[1,j] -> ecpm[1,i]: p – abs(expr(ecp[2,j])) -> ecpm[2,i]: Lbl stpe: : i + 1 -> i : j + 1 -> j: EndWhile: Lbl stpf: ep^T -> ep: Disp "Points Are In Variable ecp": Disp "And ecpt (if requested)"": Disp "Total Points:" Disp cnt: If

nnn ≥ p – 1 Then: Disp “Does Not Include O” : Disp “The Point At Infinity.” : EndPrgm
A listing for the TI-89 program sqrtmdpm() can be found in [3].
4. ecsump(r,s,a,b,p): Finds the sum of points r and s on $y^2 = x^3 + ax + b$ where r = $[x_1,y_1]$ amd s = $[x_2,y_2]$:

ecsump(r,s,a,b,p): Prgm: DelVar xone, yone, xtwo, ytwo: r[1,1] -> xone: r[1,2] -> yone: s[1,1] -> xtwo: s[1,2] -> xtwo: If [xone, yone] = [∞,∞] Then: [mod(xtwo,p), mod(ytwo,p)] -> sm: Goto stpe: EndIf: If [xtwo, ytwo] = [∞,∞] Then: [mod(xone,p), mod(yone,p)] -> sm: Goto stpe: EndIf: mod(xone,p) -> xone: mod(yone,p) -> yone: mod(xtwo,p) -> xtwo: mod(ytwo,p) -> ytwo: mod(a,p) -> a: mod(b,p) -> b: If xone = xtwo and (yone = p-ytwo or ytwo = p-yone) Then: [∞,∞] -> sm: Goto stpe: EndIf: If xone ≠ xtwo Then: mod((ytwo-yone)*modinv(mod(xtwo-xone,p),p),p) -> λ : mod(λ^2-xone-xtwo,p) -> xthree: mod((xone-xthree)*λ –yone,p) -> ythree: [xthree,ythree] -> sm: Goto stpe: EndIf: If xone = xtwo and yone ≠ 0 Then: mod((3*xone^2+a) *modinv(2*yone,p),p) -> λ: mod(λ^2-xone-xtwo,p) -> xthree: mod((xone-xthree)* λ-yone,p) -> ythree: [xthree,ythree] -> sm: Goto stpe: EndIf: If xone = xtwo and ytwo ≠ 0 Then: mod((3*xtwo^2+a)*modinv(2*ytwo,p),p) -> λ : mod(λ^2-xone-xtwo,p) -> xthree: mod((xone-xthree)* λ-ytwo,p) -> ythree: [xthree,ythree] -> sm: Goto stpe: EndIf: If xone = xtwo and yone = ytwo and yone = 0 Then: : [∞,∞] -> sm: Goto stpe: EndIf: Lbl stpe: Disp sm: EndPrgm
A listing for the TI-89 program modinv() can be found in [4].
5. 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 $y^2 = x^3 + ax + b$ (mod n) using a form of modular exponentiation:

mdexpec(ba, e, n, a, b): Prgm: [∞,∞] -> z: ba -> m: While e ≠ 0: diva(e,2) -> d: If d[1,2] = 1 then ecsump(z,m,a,b,n): sm -> z: EndIf: d[1,1] -> e: ecsump(m,m,a,b,n): sm -> m: EndWhile: Disp z: EndPrgm
A listing for the TI-89 function diva can be found in [4].

6. eclog(ba,s,a,b,p): Finds $\log_{ba}(s)$ via direct search, where $\log_{ba}(s)$ is the smallest “exponent” lg having lg·ba ≡ s (mod p) on the curve $y^2 = x^3 + ax + b$:

eclog(ba,s,a,b,p): Prgm: DelVar lg, tm: 1 -> lg: If ba = [∞,∞] Then: If s = [∞,∞] Then: Goto stpe: Else: Disp “Log Does Not Exist”: Goto stpf: EndIf: : EndIf: If ba = s Then: Goto stpe: EndIf: ba -> tm: While lg ≤ p+1+2*ceiling(√(p)): ecsump(ba,tm,a,b,p): lg + 1 -> lg: sm -> tm: If tm = s Then: Goto stpe: EndIf: EndWhile: Lbl stpe: Disp “Log Is:” : Disp lg: Lbl stpf: EndPrgm

7. ecorder(c,a,b,n,pr): 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(c,a,b,n,pr): Prgm: factors(n): n -> t: For i,1,j: t/p[i]^e[i] -> t: mdexpec(c,t,pr,a,b): z -> a1: While a1 ≠[∞,∞]: mdexpec(a1,p[i], pr,a,b): z -> a1: t*p[i] -> t: EndWhile: EndFor: Disp “Order Is:” : Disp t: EndPrgm
A listing of the TI-89 program factors() can be found in [2].

8. ecnoptsh(a,b,p): computes the number of points in $E_p(a,b)$ using the Hasse bounds (see [5]): $p + 1 - 2\sqrt{(p)} \leq \#(E_p(a,b)) \leq p + 1 + 2\sqrt{(p)}$:

ecnoptsh(a,b,p): Prgm: DelVar tmp, ba, gcn, ysq, gc, pr2, ho, hi, rr, orde: int(2*√(p)) -> pr2: p+1-pr2 -> ho: ho -> hok: p+1+pr2 -> hi: 1 -> gc: 0 -> trls: Lbl stp1: If trls ≥ 5 then ecnopts(a,b,p) : Stop: EndIf: rand(p-1) -> r: r -> rr: mod(r^3+a*r+b,p) -> ysq: jacobi(ysq,p): If k = -1 Then: Goto stp1: EndIf: sqrtmdpm(ysq,p) rot[1] -> tm: [rr,tm] -> ba: eclog(ba, [∞,∞], a,b,p): lcm(lg,gc) -> gcn: gcn -> gc: int(hi/gcn) -> hig: int(hok/gcn) -> hog: hig – hog -> tmp: If tmp = 1 Then: If gcn = hok or gcn = hi Then: gcn -> orde: Goto stpe: ElseIf gcn > hok and gcn < hi Then: gcn -> orde: Goto stpe: ElseIf hig*gcn > hi Then: hog*gcn -> orde: Goto stpe: Else: hig*gcn -> orde: Goto stpe: EndIf: Else: 1 + trls -> trls: Goto stp1: EndIf: Lbl stpe: Disp "Order Is:" : Disp orde: EndPrgm

If we prefer to use Maple© we can code each of these (with the exception of jacobi since Maple already has a built-in jacobi function) as Maple 16 procedures:

```
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;

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;
```

$ecsump := \mathbf{proc}(r, s, a, b, p)$
$\quad \mathbf{local}\ xone, xtwo, xthree, yone, ytwo, ythree, inf, \lambda, A, B, tmp;$
$\quad \mathbf{global}\ sm;\ with(numtheory) : xone := r[1] : yone := r[2] : xtwo$
$\qquad := s[1] : ytwo := s[2] : inf := [\infty, \infty];$
$\quad \mathbf{if}\ [xone, yone] = inf\ \mathbf{then}\ sm := [xtwo, ytwo] : goto(12) : \mathbf{fi};$
$\quad \mathbf{if}\ [xtwo, ytwo] = inf\ \mathbf{then}\ sm := [xone, yone] : goto(12) : \mathbf{fi};$
$\quad xone := modp(r[1], p) : yone := modp(r[2], p) : xtwo$
$\qquad := modp(s[1], p) : ytwo := modp(s[2], p) : A := modp(a, p) : B$
$\qquad := modp(b, 2) :$
$\quad \mathbf{if}\ xone = xtwo\ \mathbf{and}\ (yone = p - ytwo\ \mathbf{or}\ ytwo = p - yone)\ \mathbf{then}\ sm$
$\qquad := inf : goto(12) : \mathbf{fi};$
$\quad \mathbf{if}\ xone \neq xtwo\ \mathbf{then}\ tmp := (xtwo - xone)^{-1}\ \mathbf{mod}\ p : \lambda$
$\qquad := modp((ytwo - yone) \cdot tmp, p) : \quad xthree := modp\big(\lambda^2 - xone$
$\qquad - xtwo, p\big) :$
$\quad ythree := modp\big((xone - xthree) \cdot \lambda - yone, p\big) : \ sm := [xthree,$
$\qquad ythree] : goto(12) : \mathbf{fi};$
$\quad \mathbf{if}\ xone = xtwo\ \mathbf{and}\ yone \neq 0\ \mathbf{then}\quad tmp := (2 \cdot yone)^{-1}\ \mathbf{mod}\ p : \lambda$
$\qquad := modp\big((3 \cdot xone^2 + a) \cdot tmp, p\big) : xthree := modp\big(\lambda^2 - xone$
$\qquad - xtwo, p\big) :$
$\quad ythree := modp\big((xone - xthree) \cdot \lambda - yone, p\big) : \ sm := [xthree,$
$\qquad ythree] : goto(12) : \mathbf{fi};$
$\quad \mathbf{if}\ xone = xtwo\ \mathbf{and}\ ytwo \neq 0\ \mathbf{then}\ \ tmp := (2 \cdot ytwo)^{-1}\ \mathbf{mod}\ p : \lambda$
$\qquad := modp\big((3 \cdot xtwo^2 + a) \cdot tmp, p\big) : xthree := modp\big(\lambda^2 - xone$
$\qquad - xtwo, p\big) :$
$\quad ythree := modp\big((xone - xthree) \cdot \lambda - ytwo, p\big) : \ sm := [xthree,$
$\qquad ythree] : goto(12) : \mathbf{fi};$
$\quad \mathbf{if}\ xone = xtwo\ \mathbf{and}\ yone = ytwo\ \mathbf{and}\ yone = 0\ \mathbf{then}\ sm := inf : goto(12) :$
$\qquad \mathbf{fi};$
$\quad 12 : \ \mathbf{return}\ (sm);\ \ \mathbf{end\ proc};$

$mdexpec := \mathbf{proc}(ba, e, a, b, n)$
$\quad \mathbf{local}\ \ d, m, ee;$
$\quad \mathbf{global}\ z;$
$\quad z := [\infty, \infty] : m := ba;\ ee := e :$
$\quad d := [0, 0] : Array(d);$
$\quad \mathbf{while}\ ee \neq 0\ \mathbf{do}$
$\qquad d := \left[\mathrm{floor}\left(\frac{ee}{2}\right), ee - 2 \cdot \mathrm{floor}\left(\frac{ee}{2}\right) \right];$
$\qquad \mathbf{if}\ d[2] = 1\ \mathbf{then}\ ecsump(z, m, a, b, n) : z := sm;\ print(`z=`z) : \mathbf{fi};$
$\qquad ee := d[1] :$
$\qquad ecsump(m, m, a, b, n) :$
$\qquad m := sm;$
$\quad \mathbf{end\ do};$
$\quad \mathbf{return}(z)$
$\quad \mathbf{end\ proc};$

```
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;

eclog := proc(ba, s, a, b, p)
  local tm, limi; global lg; lg := 1 :
  if ba = [∞,∞] then goto(12) : fi;
  tm := ba; limi := p + 2·ceil(sqrt(p)) :
  while lg ≤ limi do
   ecsump(ba, tm, a, b, p); lg := lg + 1; tm := sm;
   if tm = s then goto(12) : fi;
  end do;
 return (FAIL);
 12 : return (`log is `, lg); end proc;

ecnoptsh := proc(a, b, p)
  local ho, hi, trls, gc, gcn, orde, ba, logar, tmp, pr2, ysq, r, l, hig, hog,
   tm; global lg;
  pr2 := floor(2·sqrt(p)); ho := p + 1 − pr2; hi := p + 1 + pr2; gc
   := 1; trls := 0;
 12 : if trls ≥ 5 then ecnopts(a, b, p) : goto(14) : fi;
  r := modp(rand( ), p); ysq := modp(r^3 + a·r + b, p);
   with(numtheory) : l := legendre(ysq, p);
  if l = −1 or l = 0 then goto(12) : fi; tm := msqrt(ysq, p); ba := [r,
   tm]; eclog(ba, [∞,∞], a, b, p); gcn := lcm(lg, gc); gc := gcn;
  hig := floor(hi/gcn); hog := floor(ho/gcn); tmp := floor(hi/gcn)
   − floor(ho/gcn);
   if tmp = 1 then
    if gcn = ho or gcn = hi then
     orde := gcn; goto(15);
    elif gcn > ho and gcn < hi then
     orde := gcn; goto(15);
    elif hig·gcn > hi then
     orde := hog·gcn; goto(15);
    else orde := hig·gcn;
    fi;
  else
   trls := trls + 1; goto(12);
  fi;
15 : return(`Number of Points Is: `, orde); 14 : end proc;
```

We now describe a basic form of elliptic curve encryption based on the ElGammal cryptosystem [5]. For this, we 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 $P_A = n_AG$ and $P_B = n_BG$. 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 on the TI-89 as:

ecencryp(pm,pb,ge,k,a,b,p): Prgm: mdexpec(ge,k,p,a,b): z -> kg: mdexpec(pb,k,p,a,b): z -> kpb: ecsump(pm, kpb, a,b,p): sm -> pmkpb: Disp "Encrypted P Is:" : Disp "kg = ": Disp kg: Disp "pmkpb =" : Disp pmkpb: EndPrgm

ecdecryp(kg, pmkpb, nb, a, b, p): Prgm: mdexpec(kg, nb, p, a, b): z -> nbkg: mod(-1*nbkg[1,2],p) -> nbkg[1,2]: ecsump(pmkpb, nbkg, a, b, p): sm -> pm: Disp "Plaintext Point Is:" : Disp pm: EndPrgm

<u>Example 1 (TI-89).</u>
We use p = 653 and $y^2 = x^3 – 7x + 145$. We'll encode the letters A—Z by the scheme: A = 1, B = 2, …, Y = 25, Z = 0. For Alice to encode the message "BFF" to send to Bob, we see that BFF corresponds to ciphers 266. To check that 266 is the x-coordinate of a point on the elliptic curve, define $f(x) = x^3 – 7x + 145$, so that $f(266) = 18819379 \equiv 572 \pmod{653}$. So $y^2 \equiv 572 \pmod{653}$ and sqrtmdpm(572, 653) gives 35 as a square root of 572 modulo 653. Note that is 572 were not a quadratic residue modulo 653 we might do a simple shift of the encoding of the letters, such as (for example) A = 2, B = 3 …, Y = 0, Z = 1, or see the method first proposed by Koblitz on page 174 of [5]. We will use (266, 35) as our message point: $P_M = (266,35)$. To find the order of $E_{653}(-7, 145)$ we run ecnopts(-7,145,653) to obtain 650 total points (including **O**, the point at infinity). For a generator point we may chose a random value of x, $0 \le x \le 652$. For example, if x = 0 then f(0) = 145 and sqrtmdpm(145,653) gives 168. Running ecorder([0,168], -7, 145, 650, 653) yields an order of 650 for this point. Although 650 is not a prime, using the "generator" point G = [0,168] will serve our purpose for this small example. If Bob's secret key is $n_B = 97$, then his public key is $P_B = n_BG$, so that: $P_B = 97 \cdot [0,168]$. For this computation we employ: mdexpec([0,168],97,653,-7,145) to obtain $P_B = [349,254]$. If Alice's random number k (which she selects) is k = 243, then the encrypted message which Alice sends to Bob can be obtained by:

ecencryp([266,35], [349,254], [0,168], 243, -7, 145, 653) to produce the ciphertext pair: kg = [268,62] and pmkpb = [101,258], or P_C = [[268,62] , [101,258]]. Now Bob may decrypt P_C by using ecdecryp(kg, pmkpb, nb, a, b, p) which in our example is: ecdecryp([268,62], [101,258], 97, -7, 145, 653). This outputs the plaintext point P_M as [266, 35]. So the message is in the x-coordinate, which is 266 which we map back into "BFF".

Using Maple 17, encryption and decryption procedures are given by:

```
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 2 (Maple 16)

We use the prime p = 9883 and $y^2 = x^3 +765x + 871$. 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^3 +765x + 871. Then modp(f(8571), 9883) gives 5791. Then msqrt(5791, 9883) produces the y-coordinate which is 3277, so our message point P_M is [8571, 3277]. To find the order of the group $\mathbf{E}_{9883}$ (765, 871), we run ecnopts(765, 871, 9883) which gives 9827. For a "generator" 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. Our prospective G is [7, 2813]. Next we find the order of G in the group $\mathbf{E}_{9883}$ (765, 871) by running ecorder([7, 2813], 765, 871, 9827, 9883) which produces "order is 9827". As an alternative to find the order we may run eclog([7, 2813], [∞,∞], 765, 871, 9883) which produces "log is 9827". This alternate method does not require knowledge of the order of $\mathbf{E}_{9883}$ (765, 871) but may run longer than ecorder. If Bob's secret key is n_B = 873, then his public key is $P_B = n_B G$, so that:
P_B = 873 • [7, 2813]. For this computation we use: mdexpec([7, 2813], 873, 9883, 765, 871) to obtain P_B = [7516, 1555] which is Bob's public key. 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: P_C = [[4225, 3276], [27, 203]]. Now Bob may decrypt P_C 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 P_M as [8571, 3277]. So the message is in the x-coordinate, which is 8571 which we map back into "LIKE".

It should be clear that the programs presented here are intended for demonstration purposes and only as a learning tool. For real applications the prime should be 100 or more digits. Our TI-89 programs can handle (without excessive waits) 3 or 4 digit primes and the Maple procedures can handle up to 5 or 6 digit primes only. Clearly we need better algorithms to deal with more realistic applications of elliptic curve cryptography. However this basic introduction should give you a reasonably good idea of how it works.

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, *Accessing the Primes and Exponents from the TI-89 "Factor" Command*, Proceedings of the ICTCM 2009.
3. J.N. Fadyn, *Faster Square Roots Modulo a Prime on the TI-89*, Proceedings of the ICTCM 2013.
4. J.N. Fadyn, *Solving Quadratic Congruences Modulo a Prime on The TI-89,* Proceedings of the ICTCM 2010.
5. Washington, L.C., *Elliptic Curves : Number Theory and Cryptography, Second Edition* , CRC Press, 2008.