Weihnachtsvorlesung 2021

```
####################################
# Suche von Primzahlen und PowerMod #
####################################
```

```
for n in range(180):
    if is_prime(11111111111191111111111111+n):
        print 11111111111191111111111111+n
```

```
p=11111111111191111111111131
q=11111111111191111111111197
n=p*q
print n
```

```
R=Integers(n)
print R
```

```
power_mod(3945339534939,35359387049853253245875,n)
```

```
#######
# RSA #
#######

# Alice waehlt geheim p und q und berechnet
# n = p*q und phi_n = (p-1)*(q-1)

# Alice veroeffentlicht n und eine Zahl m, z.B. m=17
```

```
phi_n=(p-1)*(q-1)
print phi_n
```

```
print n
m=17
print m
```

```
# Bob moechte Alice die Zahl a senden.
# Dazu berechnet er b = a^m (mod n)
```

```
a=39598593538845685609842999999999999934583485738753
b=power_mod(a,m,n)
b
```

```
# Alice rechnet in der Einheitengruppe von Z/nZ
# und berechnet das multiplikativ Inverse e von m.
```

```
ggT=xgcd(m,phi_n)
print ggT
```

```
ggT[1]*m+ggT[2]*phi_n
```

```
e=ggT[1]
print e
```

```
aa=power_mod(b,e,n)
print aa
print aa-a
```

```
#######################################
# Diffie-Hellman Schluesselaustausch #
#######################################
```

```
# Alice und Bob wollen sich ueber eine oeffentliche Leitung auf einen
# gemeinsamen Schluessel einigen, mit dem sie dann ueber ein symmetrisches
# Verschluesselungsverfahren (z.B. DES) grosse Datenmengen austauschen.
```

```
# Alice und Bob waehlen oeffentlich eine Gruppe G,
# z.B. die Einheitengruppe von Z/kZ fuer ein k,
# und ein Element P aus G.
```

```
R=Integers(p)
print R
```

```
P=R(17)
print P
```

```
# Alice waehlt geheim eine Zahl m und sendet oeffentlich a = P^m (mod p) an
```

```
Bob.
#  Bob  waehlt geheim eine Zahl n und sendet oeffentlich b = P^n (mod p) an
Alice.
```

```
m=43573473475934578
a=P^m
print a
n=98345345309845389477
b=P^n
print b
```

```
# Alice berechnet
```

```
print b^m
print a^n
print P^(n*m)
```

```

```

```
########################
#                      #
#  Elliptische Kurven  #
#                      #
########################
```

```
# Eine elliptische Kurve ist weder eine Ellipse noch eine Kurve.
```

```
# cubische Kurve

var('y')
Cubic = -y^2 + x*y -2*y -x^3 + 2*x^2 + 12*x + 16 # 129*x
CubicCurve=implicit_plot(Cubic,(x,-35,20),
(y,-30,50),aspect_ratio=.5,frame=False,axes=True,linewidth=2)
show(CubicCurve)
```

```
# Addition von Punkten auf cubischen Kurven:
```

```
# Wir betrachten die cubische Kurve C: x^3 + y^3 - 1729 = 0

var('y')
Taxicab=x^3+y^3-1729
```

```
C=implicit_plot(Taxicab,(x,-20,20),
(y,-20,20),aspect_ratio=1,frame=False,axes=True,linewidth=2)
show(C)
```

```
# auf C waehlen wir zwei Punkte P=(9,10) & Q=(1,12):

P=point((9,10),color='black',size=30)+text('P',
((9.5,11.3)),color='black',fontsize='large')
Q=point((1,12),color='black',size=30)+text('Q',
(1,13.3),color='black',fontsize='large')
show(C+P+Q)
```

```
# nun legen wir eine Gerade g durch P und Q:

def gerade(P,Q):
    nx=-(P[1]-Q[1])
    ny=P[0]-Q[0]
    if nx != 0:
        d=-(nx*P[0]+ny*P[1])
        return nx*x+ny*y+d
    else:
        return x-nx

gPQ=gerade((9,10),(1,12))
G=implicit_plot(gPQ,(x,-20,20),
(y,-20,20),color='black',frame=False,axes=True)
show(C+P+Q+G)
```

```
# dann bestimmen wir den dritten Schnittpunkt P#Q
# von g mit der Kurve C:

sol=solve(gPQ,y)
Schnittpkt=solve([sol[0],Taxicab==0],x,y,solution_dict=True)
print 'Die drei Schnittpunkte sind:'
print Schnittpkt
print
print 'Der dritte Schnittpunkt P#Q ist also:'
print Schnittpkt[1]
print

Spunkt=(Schnittpkt[1][x],Schnittpkt[1][y])
S=point(Spunkt,color='green',size=30)+text('P#Q',
(Spunkt[0]+1.2,Spunkt[1]+1),color='black',fontsize='large')
show(C+P+Q+G+S)
```

```
# Der projektive Punkt O = ( 1 : -1 : 0 ) ist ein Wendepunkt der Kurve C.
# Wir legen eine Gerade durch O und P#Q und bestimmen wieder den dritten
# Schnittpunkt dieser Geraden mit der Kurve C.
# In unserem Fall muessen wir den Punkt P#Q an der 1.Winkelhalbierenden
```

```
# spiegeln, damit wir den Punkt P+Q erhalten.

Rpunkt=(Schnittpkt[1][y],Schnittpkt[1][x])
gSR=gerade(Spunkt,Rpunkt)
L=implicit_plot(gSR,(x,-20,20),
(y,-20,20),color='green',linestyle='dashdot',frame=False,axes=True)
R=point(Rpunkt,color='red',size=30)+text('P+Q',
(Rpunkt[0]+2,Rpunkt[1]+.5),color='black',frame=False,axes=True,fontsize='large')
show(C+P+Q+G+S+R+L)
```

```
# Elliptische Kurven:
#
# Die Punkte einer cubische Kurve, zusammen mit einem
# Wendepunkt O und der Addition von Punkten als Operation
# bildet eine abelsche Gruppe. Der Wendepunkt O ist das
# Neutralelement der additiven Gruppe, also die 0.
#
# Gruppen dieser Form heissen "elliptische Kurven".
#
# Cubische Kurven haben verschieden Normalformen,
# eine davon ist folgende:
#
# y^2 + a1*x*y + a3*y = x^3 + a2*x^2 + a4*x + a6
#
# Beim Befehl "EllipticCurve" kann man entweder
# alle fuenf Parameter [a1,a2,a3,a4,a6] eingeben,
# also: EllipticCurve([a1,a2,a3,a4,a6])
# oder bloss [a4,a6]
# also: EllipticCurve([a4,a6])
# im letzteren Fall ist a1=a2=a3=0.
#
# Bei diesen Normalformen ist der Wendepunkt bei (0:1:0),
# also im Unendlichen (man betrachtet die cubischen
# Kurven in der projektiven Ebene). Der Wendepunkt
# ist also dort, wo sich die y-Achse und die Ferngerade
# schneiden; deshalb muss P#Q an der x-Achse gespiegelt
# werden um den Punkt P+Q zu erhalten.
```

```
# Betrachten wir zuerst eine elliptische Kurve
# mit einem Punkt der Ordnung 6:
```

```
v=1  # v=0 ist nicht erlaubt, aber sonst kann v beliebig gewaehlt werden
m=-v; d=4*v^3
E=EllipticCurve([0,m^2,0,2*m*d,d^2])
P=point
print E
show(plot(E,xmin=-4.3,xmax=4.5,aspect_ratio=1))
print 'Der Punkt P =', (4*v^2,-8*v^3), 'ist ein Punkt der Ordnung 6.'
```

```
# im Folgenden wird ueberprueft, ob P die Ordnung 6 hat:

P1=E(4*v^2,-8*v^3)
P2=P1+P1
P3=P2+P1
P4=P3+P1
P5=P4+P1
P6=P5+P1
p  =point((P1[0],P1[1]),size=30,color='red')+text('P',
(P1[0]+.3,P1[1]+.5),color='black')
p +=point((P2[0],P2[1]),size=30,color='black')+text('2P',
(P2[0]+.6,P2[1]-.3),color='black')
p +=point((P3[0],P3[1]),size=30,color='black')+text('3P',
(P3[0]+.6,P3[1]+.5),color='black')
p +=point((P4[0],P4[1]),size=30,color='black')+text('4P',
(P4[0]+.5,P4[1]+.5),color='black')
p +=point((P5[0],P5[1]),size=30,color='green')+text('5P',
(P5[0]+.3,P5[1]-.5),color='black')
PlotE=plot(E,xmin=-4.3,xmax=4.5,aspect_ratio=1)
show(PlotE+p)
print '6*P =', P6
```

```
# Die Punkte O,P,2P,3P,4P,5P bilden eine
# zyklische Gruppe der Ordnung 6.

gP14=gerade(P1,P4)
gP34=gerade(P3,P4)
G14=implicit_plot(gP14,(x,-4,4.6),
(y,-10,10),color='black',frame=False,axes=True)
G34=implicit_plot(gP34,(x,-4.6,4.6),
(y,-10,10),color='green',frame=False,axes=True)
show(PlotE+p+G14+G34)
```

```
# Betrachten wir die rationalen Punkte einer cubischen Kurve C,
# d.h. Punkte P = (x,y) auf C mit x,y beide rational, so ist
# die entsprechende elliptische Kurve E(Q) eine endlich erzeugte
# abelsche Gruppe (das ist der Satz von Mordell).
#
# Mit dem Hauptsatz ueber entlich erzeugte abelsche Gruppen ist
# E(Q) also von der Form
#        E(Q) = Z/n1Z x Z/n2Z x ... x Z/nkZ x Z^r
```

```
v=1  # v=0 ist nicht erlaubt, aber sonst kann v beliebig gewaehlt werden
m=-v; d=4*v^3
E=EllipticCurve([0,m^2,0,2*m*d,d^2])
print E
print
tor = E.torsion_subgroup()
print tor
print
```

```
print "die Torsionsgruppe wird generiert durch:"
print tor.gens()
print
G=E.gens(descent_second_limit=10)
L=len(G)
print "der Rang der Kurve ist",L,"mit den Generatoren:"
print G
```

```
a=3; b=4;
E=EllipticCurve([a^4+b^4,a^4*b^4])
print E
print
tor = E.torsion_subgroup()
print tor
print
print "die Torsionsgruppe wird generiert durch:"
print tor.gens()
print
G=E.gens(descent_second_limit=10)
L=len(G)
print "der Rang der Kurve ist",L,"mit den Generatoren:"
print G
```

```
a=3; b=4;
a=21; b=20;
E=EllipticCurve([0,a^4+b^4,0,a^4*b^4,0])
print E
print
tor = E.torsion_subgroup()
print tor
print
print "die Torsionsgruppe wird generiert durch:"
print tor.gens()
print
G=E.gens(descent_second_limit=10)
L=len(G)
print "der Rang der Kurve ist",L,"mit den Generatoren:"
print G
```

```
A=7
E=EllipticCurve([-A^2,0])
print E
print
tor = E.torsion_subgroup()
print tor
print
print "die Torsionsgruppe wird generiert durch:"
print tor.gens()
print
G=E.gens(descent_second_limit=10)
```

```
L=len(G)
print "der Rang der Kurve ist",L,"mit den Generatoren:"
print G
```

```
# Der Satz von Mazur besagt, dass ueber den rationalen Zahlen
# nur folgende 15 Torsionsgruppen moeglich sind:
# Z/nZ fuer n=1,2,3,4,5,6,7,8,9,10,12
# Z/2Z x Z/2mZ fuer m=2,3,4
```

```
######################
# Kongruente Zahlen #
######################
```

```
# Eine positive ganze Zahl A heisst kongruent, falls es ein
# rechtwinkliges Dreieck gibt mit den rationalen Seiten p,q,r
# und der Flaeche A. Sind p,q die Laengen der Katheten, so
# ist A = pq/2.
#
# Es laesst sich zeigen, dass A genau dann eine kongruente
# Zahl ist, wenn die elliptische Kurve y^2 = x^3 - A^2*x
# positiven Rang hat.
```

```
# Ist (x,y) ein rationaler Punkt unendlicher Ordnung
# der elliptischen Kurve y^2 = x^3 - A^2*x, so ist (p,q,r)
# ein rationales pythagoraeisches Tripel mit A = pq/2,
# wobei p = 2xA/y, q = (x^2-A^2)/y, r = (x^2+A^2)/y.
```

```
# A = 6 ist eine kongruente Zahl, denn (3,4,5) ist ein
# rationales pythagoraeisches Tripel und 6 = 3*4/2

A=6
E=EllipticCurve([0,0,0,-A^2,0])
print E
print
tor = E.torsion_subgroup()
print tor
print
print "die Torsionsgruppe wird generiert durch:"
print tor.gens()
print
G=E.gens(descent_second_limit=10)
L=len(G)
print "der Rang der Kurve ist",L,"mit den Generatoren:"
print G
```

```
# (x,y) -> (p,q,r)
```

```
def xy2pqr(x,y):
    return(abs(2*x*A/y), abs((x^2-A^2)/y), abs((x^2+A^2)/y))
```

```
(p,q,r)=xy2pqr(-3, 9)
print (p,q,r)
print p^2+q^2-r^2
print p*q/2
```

```
P=E((-3, 9))
for i in range(5):
    print i,"P =",i*P
```

```
# A = 7 ist eine kongruente Zahl:

A=7
E=EllipticCurve([0,0,0,-A^2,0])
print E
print
tor = E.torsion_subgroup()
print tor
print
print "die Torsionsgruppe wird generiert durch:"
print tor.gens()
print
G=E.gens(descent_second_limit=10)
L=len(G)
print "der Rang der Kurve ist",L,"mit den Generatoren:"
print G
```

```
(p,q,r)=xy2pqr(25, 120)
print (p,q,r)
print p^2+q^2-r^2
print p*q/2
print

P=E((25, 120))
for i in range(5):
    print i,"P =",i*P
```

```
#####################################################
#  Diffie-Hellman Schluesselaustausch mit        #
#  elliptischen Kurven ueber endlichen Koerpern  #
#####################################################
```

```
# Elliptische Kurven koennen ueber beliebigen
```

```
# Koerpern betrachtet werden, z.B. ueber Koerpern
# der Form Zmod(p) fuer p prim:

v=1
m=-v; d=4*v^3
E=EllipticCurve([0,m^2,0,2*m*d,d^2])
P=point
print E
print
show(plot(E,xmin=-4.3,xmax=4.5,aspect_ratio=1))
print 'Der Punkt P =', (4*v^2,-8*v^3), 'ist ein Punkt der Ordnung 6.'
```

```
print "Nun dieselbe Kurve modulo 137:"

p=137
E=EllipticCurve(Zmod(p),[0,m^2,0,2*m*d,d^2])
print E
show(plot(E))
print 'Die Ordnung von E ist', E.cardinality()
```

```
# auf E koennen wir Punkte waehlen und diese
# zusammenzaehlen oder vervielfachen:

P=E(113,48)
Q=E(25,41)
Pp=point((P[0],P[1]),color='red',size=30)
Qp=point((Q[0],Q[1]),color='purple',size=30)
PQ=P+Q
PQp=point((PQ[0],PQ[1]),color='green',size=30)
mP=77*P
PunktP=text("P",(P[0],P[1]+4),color='black',fontsize='large')
PunktQ=text("Q",(Q[0],Q[1]+4),color='black',fontsize='large')
PunktPQ=text("P+Q",(PQ[0],PQ[1]+4),color='black',fontsize='large')
PunktmP=text("77*P",(mP[0],mP[1]+4),color='black',fontsize='large')
mPp=point((mP[0],mP[1]),color='black',size=30)
show(plot(E)+Qp+Pp+PunktP+PunktQ) # +PunktPQ+PQp  +PunktmP+mPp
print 'P =',P,'    (rot)'
print 'Q =',Q,'     (violett)'
print 'P + Q =',PQ,' (gruen)'
print '77*P =',mP,'  (schwarz)'
```

```
P=E(113,48)
Q=E(25,41)
Pp=point((P[0],P[1]),color='red',size=30)
Qp=point((Q[0],Q[1]),color='purple',size=30)
PQ=P+Q
PQp=point((PQ[0],PQ[1]),color='green',size=30)
mP=77*P
PunktP=text("P",(P[0],P[1]+4),color='black',fontsize='large')
PunktQ=text("Q",(Q[0],Q[1]+4),color='black',fontsize='large')
```

```
PunktPQ=text("P+Q",(PQ[0],PQ[1]+4),color='black',fontsize='large')
PunktmP=text("77*P",(mP[0],mP[1]+4),color='black',fontsize='large')
mPp=point((mP[0],mP[1]),color='black',size=30)
show(plot(E)+Qp+Pp+PunktP+PunktQ+PunktPQ+PQp+PunktmP+mPp)
print 'P =',P,'    (rot)'
print 'Q =',Q,'     (violett)'
print 'P + Q =',PQ,' (gruen)'
print '77*P =',mP,'  (schwarz)'
```

```
P=E((4,129)) # ist ein Punkt der Ordnung 6
for i in range(1,7):
    print i,i*P
```

```
# Diffie-Hellman Schluesselaustausch mit elliptischen Kurven:
#
# Alice & Bob einigen sich auf die elliptische Kurve E ueber einem
# endlichen Koerper (das ist eine additive abelsche Gruppe),
# z.B. E: y^2 = x^3 - 9*x + 17 ueber dem Koerper Zmod(p)
# mit p = 11111111111191111111111197,
# und bestimmen dann einen Punkt Q auf E, zum Beispiel
# Q = (16 : 63 : 1).
#
# Sowohl die Kurve E (inkl. Primzahl p) wie auch der Punkt Q sind
oeffentlich.
#

p=11111111111191111111111197
E=EllipticCurve(Zmod(p),[-9,17])
print E
Q=E(16,63)
```

```
# Alice und Bob waehlen nun jeweils eine geheime Zahl m bzw. n.
#
# Die geheime Zahl von Alice sei

m=77777777777777777777777777

# und die geheime Zahl von Bob sei

n=33333333333333333

# Alice sendet Bob den Punkt mQ, Bob sendet Alice den Punkt nQ,
# und der gemeinsame Schluessel, den beide (und nur sie) berechnen
# koennen ist die x-Koordinate des Punktes mnQ.
```

```
# Das sieht dann etwa so aus:

print 'Alice sendet Bob den Punkt mQ =',m*Q
print ' '
```

```
print 'Bob sendet Alice den Punkt nQ =',n*Q
print ' '
mnQ=(m*n)*Q
print 'mnQ =',mnQ
print ' '
print 'Der gemeinsame Schluessel von Alice und Bob ist',mnQ[0]
```

```
#########################################################################
# Die elliptische Kurve secp256k1, die fuer Bitcoins verwendet wird #
#########################################################################
```

```
p=2^256 - 2^32 - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 -1
print p
print
print "p prime?",is_prime(p)
print
x0=55066263022277343669578718895168534326250603453777594175500187360389116729240
y0=32670510020758816978083085130507043184471273380659243275938904335757337482424
print (x0,y0)
print
E=EllipticCurve(Zmod(p),[0,7])
print E
print
print 'Die Ordnung von E ist', E.cardinality()
print
print E.abelian_group()
```

```
#########################################################
# Elliptische Kurven ueber Koerpern der Charakteristik 2 #
#########################################################
```

```
q=2^123
F.<Z>=GF(q,'Z')
```

```
A=F(Z^17+Z^13+1)
A^1502
```

```
E=EllipticCurve(F,[Z^2,Z^2+1,0,0,1])
print E
```

```
P=E(1,1)
print P
```

```
9347987492874092874902847*P
```

```
# Alice waehlt wieder eine Zahl m und Bob eine Zahl n (geheim).
# Dann sendet Alice an Bob m*P und Bob an Alice n*P
# der gemeinsame Schluessel ist dann z.B. die erste
# Koordinate des Punkts m*n*P:

m=2746876583764
n=9238709873583
AB=m*P
BA=n*P
print "Alice -> Bob:"
print m*P
print
print "Bob -> Alice:"
print n*P
print
print "Der gemeinsame Schlüssel ist dann:"
print
print (n*AB)[0]
```