
Solution.
Throughout, we use the notation of Theorem 6.6.

(a) We wish to compute \((0, 2) \oplus (3, -5)\).

Secant slope is:
\[
\lambda = \frac{(2) - (-5)}{(0) - (3)} = -\frac{7}{3}
\]
\[
x_3 = (-\frac{7}{3})^2 - (0) - (3) = 22/9
\]
\[
y_3 = (-\frac{7}{3}) \cdot (0 - 22/9) - 2 = 100/27
\]
\[
\Rightarrow (0, 2) \oplus (3, -5) = (22/9, 100/27)
\]

(b) First, we wish to compute \(P \oplus P = (0, 2) \oplus (0, 2)\).

Tangent slope is:
\[
\lambda = \frac{3 \cdot (0)^2 + -2}{2 \cdot (2)} = -1/2
\]
\[
x_3 = (-1/2)^2 - (0) - (0) = 1/4
\]
\[
y_3 = (-1/2) \cdot (0 - 1/4) - (2) = -15/8
\]
\[
\Rightarrow (0, 2) \oplus (0, 2) = (1/4, -15/8)
\]

Next, we wish to compute \(Q \oplus Q = (3, -5) \oplus (3, -5)\).
Tangent slope is:
\[
\lambda = \frac{3 \cdot (3)^2 + (-2)}{2 \cdot (-5)} = -\frac{5}{2}
\]
\[
x_3 = (-\frac{5}{2})^2 - (3) - (3) = \frac{1}{4}
\]
\[
y_3 = (-\frac{5}{2}) \cdot (3 - \frac{1}{4}) + 5 = -\frac{15}{8}
\]
\Rightarrow (3, -5) \oplus (3, -5) = (1/4, -15/8)

This is not a typo; in fact \( P \oplus P = Q \oplus Q \), which is an unusual coincidence.

(c) To find \( P \oplus P \oplus P \), use the answer to part (b); we wish to compute \( (1/4, -15/8) \oplus (0, 2) \).

Secant slope is:
\[
\lambda = \frac{(-15/8) - (2)}{(1/4) - (0)} = -31/2
\]
\[
x_3 = (-31/2)^2 - (1/4) - (0) = 240
\]
\[
y_3 = (-31/2) \cdot (0 - 240) - 2 = 3718
\]
\Rightarrow (1/4, -15/8) \oplus (0, 2) = (240, 3718)

To find \( Q \oplus Q \oplus Q \), use the answer to part (b) again; we wish to compute \( (1/4, -15/8) \oplus (3, -5) \).

Secant slope is:
\[
\lambda = \frac{(-15/8) - (-5)}{(1/4) - (3)} = -\frac{25}{22}
\]
\[
x_3 = (-25/22)^2 - (1/4) - (3) = -\frac{237}{121}
\]
\[
y_3 = (-25/22) \cdot (3 + 237/121) + 5 = -\frac{845}{1331}
\]
\Rightarrow (1/4, -15/8) \oplus (3, -5) = (-237/121, -845/1331)

Due the night of Thursday 11/17 (hard deadline 4am on 11/18).
2. Exercise 6.2 (Note: the notation $P \ominus Q$ means the same thing as $P \oplus (\ominus Q)$, and the notation $2P$ means the same thing as $P \oplus P$).

Solution.

(a) To find $P \oplus Q$, we wish to compute $(-1, 4) \oplus (2, 5)$.

Secant slope is:

$$\lambda = \frac{(4) - (5)}{(−1) - (2)} = \frac{1}{3}$$

$$x_3 = \frac{(1/3)^2 - (-1) - (2)}{−8/9}$$

$$y_3 = \frac{(1/3) \cdot (-1 + 8/9) - 4}{−109/27}$$

$$\Rightarrow (-1, 4) \oplus (2, 5) = \left(-\frac{8}{9}, -\frac{109}{27}\right)$$

To find $P \ominus Q$, note that $\ominus Q = (2, −5)$. So we wish to compute $(-1, 4) \ominus (2, −5)$.

Secant slope is:

$$\lambda = \frac{(4) - (-5)}{(−1) - (2)} = -3$$

$$x_3 = \frac{(-3)^2 - (-1) - (2)}{8}$$

$$y_3 = \frac{(-3) \cdot (-1 - 8) - 4}{23}$$

$$\Rightarrow (-1, 4) \ominus (2, −5) = (8, 23)$$

(b) To find $2P = P \oplus P$, we wish to compute $(-1, 4) \oplus (-1, 4)$.

Tangent slope is:

$$\lambda = \frac{3 \cdot (-1)^2 + 0}{2 \cdot (4)} = \frac{3}{8}$$

$$x_3 = \frac{(3/8)^2 - (-1) - (-1)}{137/64}$$

$$y_3 = \frac{(3/8) \cdot (-1 - 137/64) + 4}{-2651/512}$$

$$\Rightarrow (-1, 4) \oplus (-1, 4) = \left(\frac{137}{64}, -\frac{2651}{512}\right)$$
To find $2Q = Q ⊕ Q$, we wish to compute $(2, 5) ⊕ (2, 5)$.

Tangent slope is:

$$\lambda = \frac{3 \cdot (2)^2 + 0}{2 \cdot (5)} = \frac{6}{5}$$

$$x_3 = (6/5)^2 - (2) - (2) = -\frac{64}{25}$$

$$y_3 = (6/5) \cdot (2 + \frac{64}{25}) - 5 = \frac{59}{125}$$

$$\Rightarrow (2, 5) ⊕ (2, 5) = \left(-\frac{64}{25}, \frac{59}{125}\right)$$

For the bonus, here is a complete list of integral points:

$$(-2, 3), (-1, 4), (2, 5), (4, 9), (8, 23), (43, 282), (52, 375), (5234, 378661)$$

$$(-2, -3), (-1, -4), (2, -5), (4, -9), (8, -23), (43, -282), (52, -375), (5234, -378661)$$

3. Exercise 6.5, parts (a) and (b) (Hint. You can save some time by making two lists in advance: values of $y^2$ for various $y$ and values of $x^3 + Ax + B$ for various values of $x$, then checking for numbers occurring in both lists).

Solution.

(a) We can list possible values of the left hand side

$$X \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline X^3 + 3X + 2 \pmod{7} & 2 & 6 & 2 & 3 & 1 & 2 & 5 \end{bmatrix}$$

and of left hand side

$$Y \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline Y^2 \pmod{7} & 0 & 1 & 4 & 2 & 2 & 4 & 1 \end{bmatrix}$$

Checking for numbers occurring in both lists (and including $O$), we obtain the following list of nine points.

$$(0, 3), (0, 4), (2, 3), (2, 4), (4, 1), (4, 6), (5, 3), (5, 4), (O)$$

(b) We can list possible values of the left hand side

$$X \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline X^3 + 2X + 7 \pmod{11} & 7 & 10 & 8 & 7 & 2 & 10 & 4 & 1 & 7 & 6 & 4 \end{bmatrix}$$

$$Y \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline Y^2 \pmod{11} & 0 & 1 & 4 & 9 & 5 & 3 & 5 & 9 & 4 & 1 \end{bmatrix}$$

Checking for numbers in both lists, we obtain the following list of seven points.

$$(6, 2), (6, 9), (7, 1), (7, 10), (10, 2), (10, 9), (O)$$
Solution.

(a) There are only four points on the elliptic curve described by $Y^2 \equiv X^3 + X + 2 \pmod{5}$. I have created the following addition table using the code in the solution to problem 8 below.

\[
\begin{array}{c|cccc}
0 & (1,2) & (1,3) & (4,0) \\
1 & 0 & (1,2) & (1,3) & (4,0) \\
2 & (1,2) & (1,2) & 0 & (1,3) \\
3 & (1,3) & (1,3) & 0 & (4,0) \\
4 & (4,0) & (4,0) & (1,3) & 0 \\
\end{array}
\]

(b) There are six points on the elliptic curve described by $Y^2 \equiv X^3 + 2X + 3 \pmod{7}$. I have used the addition code in problem 8 to create the following addition table.

\[
\begin{array}{c|cccccccc}
0 & (2,1) & (2,6) & (3,1) & (3,6) & (6,0) \\
1 & 0 & (2,1) & (2,6) & (3,1) & (3,6) & (6,0) \\
2 & (2,1) & (2,6) & 0 & (3,1) & (3,6) & (6,0) \\
3 & (3,1) & (3,6) & (6,0) & 0 & (2,1) & (3,6) \\
4 & (3,6) & (6,0) & (2,1) & 0 & (3,1) & (2,6) \\
5 & (6,0) & (6,0) & (3,1) & (3,6) & (2,1) & 0 \\
\end{array}
\]

5. Exercise 6.7.

Solution.

The following bit of code defined a couple functions that list the points on an elliptic curve, then count them. This code would need to be modified if it is intended to work for primes much larger than 16 bits, as it currently just tries all $p^2$ possible points $(x, y)$, but that is certainly not a problem for 1-digit primes.

```python
>>> def pointlist(A,B,p):
...     res = [0]
...     for x in xrange(p):
...         for y in xrange(p):
...             if (y*y % p) == (x**3 + A*x + B) % p:
...                 res += [(x,y)]
...     return res
... >>> def count(A,B,p):
...     pts = len(pointlist(A,B,p))
...     print 'For p = %d, there are %d points; t_p = %d' % (p, pts, p+1-pts)
... >>> for p in [3,5,7,11]:
...     count(1,1,p)
... For p = 3, there are 4 points; t_p = 0
For p = 5, there are 9 points; t_p = -3
For p = 7, there are 5 points; t_p = 3
For p = 11, there are 14 points; t_p = -2
```
Indeed, each of these traces of Frobenius are less in absolute value than twice the square root of the prime.


**Solution.**
Below, I perform this computation in the Python terminal, making use of the implementation of elliptic curve addition from problem 8 below. This bit of code computes the first several multiples of $P$, stopping once it reaches $Q$.

```python
>>> P = (4,2)
>>> Q = (0,1)
>>> nP = 0
>>> n = 0
>>> while nP != Q:
...     nP = add(nP,P,1,1,5)
...     n += 1
...     print n,nP
...
1 (4, 2)
2 (3, 4)
3 (2, 4)
4 (0, 4)
5 (0, 1)
```

As seen in this code, the discrete logarithm is 5, since $5 \cdot P = Q$.


**Solution.**
First observe that if $n = is + r$, where $r$ is the remainder when $n$ is divided by $s$, then $nP = i(sP) \oplus rP = iO \oplus rP = O \oplus rP = rP$. So $nP$ depends only on the remainder when $n$ is divided by $s$. 

Suppose that $Q = nP$. Then $nP = n_0 P$. Adding $\ominus n_0 P$ to both sides of this equation yields $(n - n_0)P = O$. Let $r$ be the remainder when $n - n_0$ is divided by $s$. Then it follows that $rP = O$. Since $0 \leq r < s$, and $s$ is the *minimum* positive integer such that $sP = O$, it follows that $r = 0$, since none of $1 \cdot P, 2 \cdot P, \ldots, (s-1)P$ are equal to $O$. In other words, $n - n_0$ is *divisible* by $s$. Hence $n \equiv n_0 \pmod{s}$, from which it follows that $n = n_0 + is$ for some integer $i$.

**Programming problems**
Full formulation and submission: [https://www.hackerrank.com/m158-2016-pset-8](https://www.hackerrank.com/m158-2016-pset-8)

8. Given a prime $p$, an elliptic curve over $\mathbb{F}_p$ and two points on the curve, compute the sum of the two points.

**Solution.**
The following implementation essentially follows the steps in theorem 6.6.
### Omitted: code for function ext_euclid (extended Euclidean algorithm)

```python
def add(P, Q, A, B, p):
    if P == 0: return Q
    if Q == 0: return P
    if P[0] == Q[0] and ((P[1] + Q[1]) % p) == 0:
        return 0
    if P[0] == Q[0] and P[1] == Q[1]:
        lam = (3*P[0]**2 + A) * ext_euclid(2*P[1], p)[0] % p
    else:
        lam = (P[1] - Q[1]) * ext_euclid(P[0]-Q[0], p)[0] % p
    nu = P[1] - lam*P[0]
    x = (lam*lam - P[0] - Q[0]) % p
    y = -(lam*x + nu) % p
    return (x, y)

def print_point(P):
    if P == 0: print 0
    else:
        print P[0], P[1]

def read_point():
    ls = map(int, raw_input().split())
    if len(ls) == 2: return tuple(ls)
    else: return 0
```

9. Given a prime \( p \), an elliptic curve over \( \mathbb{F}_p \), a point \( P \) on the curve and an integer \( n \) (up to 64 bits), compute the point \( n \cdot P \) on the elliptic curve (you will want to implement the double-and-add algorithm, or something similar).

**Solution.**

The following implementation mimics the fast-powering algorithm; the exponent is repeatedly dividing in two while the point is doubled.

```python
def mult(P, n, A, B, p):
    Q = 0
    while n > 0:
```

### Omitted: code for ext_euclid, add(P, Q, A, B, p), print_point, read_point (see previous)
if n % 2 == 1:
    Q = add(Q,P,A,B,p)
    P = add(P,P,A,B,p)
    n /= 2
return Q

# I/O
A,B,p = map(int,raw_input().split())
P = read_point()
n = int(raw_input())
print_point(mult(P,n,A,B,p))

10. Given a prime $p$, an elliptic curve over $\mathbb{F}_p$ of order $q$ (where $q$ is provided for you), and two points $P,Q$ on the curve, find the smallest positive integer $n$ such that $n \cdot P = Q$ (i.e. solve the elliptic curve discrete logarithm problem). The value of $p$ will be up to 28 bits long, so trial and error is unlikely to solve all of the test cases; you will most likely want to adapt the BSGS algorithm to the setting of elliptic curves.

Solution.

We can mimic the BSGS algorithm (e.g. problem 8 on Problem Set 4). We select a step size $B$, and create two lists: one contains $O, P, 2P, \cdots, (B-1)P$, while the other contains $Q, Q \ominus BP, Q \ominus 2P, \cdots, Q \ominus (B-1)P$. Then an overlap between these two lists corresponds precisely to a solution $(i + Bj)P = Q$ to the discrete logarithm problem. The possible values of $i+j$ for $0 \leq i,j < B$, are precisely $0, 1, 2, \cdots, B^2-1$, so this will find any solution to the discrete logarithm problem less than $B^2$; as long as $B^2 \geq q$ this will find some solution.

As on problem set 4, it is necessary to use some efficient method to find a collision between two lists; the approach below uses a dict as a reverse lookup table for the giantstep list, and then tries all values from the babystep list to see if they occur.

```python
# Omitted: all functions from the previous problem.

import math
def dlp(P,Q,A,B,p,q):
    B = int(math.sqrt(q))+1
    PB = mult(P,B,A,B,p)
    PBinv = (PB[0],-PB[1]%p)
    bs = []
    Pi = 0
    for i in xrange(B):
        bs += [Pi]
        Pi = add(Pi,PBinv,A,B,p)
    gsrev = dict()
    rhs = Q
    for i in xrange(B):
        if rhs not in gsrev:
            gsrev[rhs] = i
        rhs = add(rhs,PBinv,A,B,p)
```

Due the night of Thursday 11/17 (hard deadline 4am on 11/18).
for i in xrange(B):
    elt = bs[i]
    if elt in gsrev:
        return (i + B*gsrev[elt]) % q

A,B,p,q = map(int,raw_input().split())
P = read_point()
Q = read_point()
print dlp(P,Q,A,B,p,q)

11. Devise a method to create “blind forgeries” for a given DSA public key. That is, given \( p, g \) and \( A \) as in DSA, generate integers \( S_1, S_2, \) and \( D \) such that \((S_1, S_2)\) is a valid signature for \( D \). You will likely want to adapt the strategy of exercise 4.7 from Elgamal to DSA. Your method should be non-deterministic; the grading script will give the same test case multiple times to check that the same answer is not returned each time.

Solution.

We can attempt to begin as in the ElGamal version, by choosing \( i \) and \( j \) at random and setting
\[
S_1 = g^i A^j \pmod{p}\pmod{q}.
\]
Since the order of \( g \) is the second prime \( q \) in DSA, we can choose \( i \) and \( j \) from \( \mathbb{Z}/q \).

Now, to decide how to write down \( S_2 \) and \( D \), we can consider the verification equation that we wish to satisfy.

\[
g^{S_2^{-1}D} A^{S_2^{-1}S_1} \pmod{p}\pmod{q} = S_1
\]
It is very hard to work with this equation: the \( \%p\%q \) term rules doing virtually any manipulation of the exponent. However, we know that since \( S_1 = g^i A^j \pmod{p} \), it is sufficient to solve the following (mod \( p \)) congruence instead.

\[
g^{S_2^{-1}D} A^{S_2^{-1}S_1} \equiv g^i A^j \pmod{p}
\]
This congruence can be transformed from a (mod \( p \)) congruence into a (mod \( q \)) congruence by expressing both sides as powers of \( g \) (and recalling that \( g \) has order \( q \)).

\[
g^{S_2^{-1}D + aS_2^{-1}S_1} \equiv g^{i + aj} \pmod{p}
\]
\[\Leftrightarrow S_2^{-1}D + aS_2^{-1}S_1 \equiv i + aj \pmod{q}\]

Now, the essential difficulty in solving this congruence, for Eve, is that she doesn’t know what \( a \) is. So she needs to somehow solve a congruence that includes a completely unknown quantity. The most foolproof way to do this is to choose the variables in such a way that the desired congruence is true no matter what \( a \) is. There turns out to be a way to do this: make sure that all of the \( a \) terms cancel. This can be arranged by making sure that the following congruences hold.
\[ S_2^{-1}S_1 \equiv j \pmod{q} \] (to force the a terms to cancel
\[ S_2^{-1}D \equiv i \pmod{q} \] (to ensure that the remaining terms match)

Remember that we’ve already chosen \( i, j, \) and \( S_1, \) so we just solve for \( S_2 \) and \( D \) as follows.

\[ S_2 \equiv j^{-1}S_1 \pmod{q} \]
\[ D \equiv iS_2 \pmod{q} \]
\[ \equiv ij^{-1}S_1 \pmod{q} \]

These choices of \( D, S_1, S_2 \) will indeed satisfy the verification equation, hence form a valid signed document. Note that in the end, this was virtually the same as the approach in the Elgamal version, except that we remove two minus signs and replace \((\pmod{p - 1})\) with \((\pmod{q})\). An implementation is below.

```python
### Omitted: code for ext_euclid

import random
random.seed()

def forge(p,q,g,A):
    i = random.randrange(1,q)
    j = random.randrange(1,q)
    s1 = pow(g,i,p)*pow(A,j,p) % p % q
    s2 = ext_euclid(j,q)[0]*s1 % q
    d = i * s2 % q
    return d,s1,s2

### I/O
p,q,g,A = map(int,raw_input().split())
d,s1,s2 = forge(p,q,g,A)
print d,s1,s2
```

Due the night of Thursday 11/17 (hard deadline 4am on 11/18).