All exercise numbers from the textbook refer to the second edition.

1. Suppose that $p, q, g$ are DSA public parameters (i.e. $p, q$ are primes, and $g$ has order $q$ modulo $p$), and $A \equiv g^a \pmod{p}$ is Samantha’s public (verification) key, while $a$ is her private (signing) key. As we discussed in class, there are two main sorts of algorithms that Eve might use to extract $a$ from $A$: collision algorithms (whose runtime depends on $q$), and the number field sieve (whose runtime depends on $p$). For simplicity, assume that Eve has a collision algorithm that can extract $a$ in $\sqrt{q}$ steps, and an implementation of the number field sieve that can extract $a$ in $e^{2(\ln p)^{1/3}(\ln \ln p)^{2/3}}$ steps (the true runtimes would involve a constant factor that would depend on implementation, and various other factors depending on the cost of arithmetic modulo $p$ and of finding collisions).

(a) Suppose that Samantha is confident that her private key will be safe as long as Eve does not have time to perform more than $2^{64}$ steps in either algorithm. How many bits long should she choose $p$ to be? How many bits long should $q$ be?

(b) What if she instead wants to be safe as long as Eve doesn’t have time for $2^{128}$ steps?

(c) The NSA’s recommendation for “Top Secret” government communications is to use 3072 bit values of $p$, and 384 bit values of $q$. How does this compare to your answers above? If the difference is significant, what might explain the discrepancy?

For parts (a) and (b), it is sufficient to write a short script to find the minimum safe numbers of bits by trial and error (there are more efficient ways, of course).

Solution.

(a) The value of $p$ must be large enough that $e^{2(\ln p)^{1/3}(\ln \ln p)^{2/3}} \geq 2^{64}$. Taking natural logarithms and simplifying slightly, this is equivalent to

$$(\ln p)(\ln \ln p)^2 \geq (32 \ln 2)^3.$$

Solving the equation $x(\ln x)^2 = (32 \ln 2)^3$ numerically gives $x \approx 325.898$, hence we require $\ln p \geq 325.898$, i.e. $\log_2 p \geq 325.898 / \ln 2 = 470.171$. So $p$ should be at least 470 bits long (roughly).

The value of $q$ must ensure that $\sqrt{q} \geq 2^{64}$, i.e. $\log_2 q \geq 128$. So $q$ must be at least 128 bits long.

(b) The value of $p$ must satisfy $e^{2(\ln p)^{1/3}(\ln \ln p)^{2/3}} \geq 2^{128}$, i.e.

$$(\ln p)(\ln \ln p)^2 \geq (64 \ln 2)^3.$$

This requires $\ln p \geq 1603.039$, i.e. $\log_2 p \geq 2312.696$. So $p$ must be at least 2300 bits long (roughly).

The value of $q$ must ensure that $\sqrt{q} \geq 2^{128}$, i.e. $\log_2 q \geq 256$. So $q$ should be at least 256 bits long.

(c) The NSA’s recommendations are similar to, but noticeably larger than, the answers in part (b). Of course, $2^{128}$ is far too generous as an estimate of how many steps Eve could ever complete; so this appears to suggest that the NSA recommendations are much larger than they need to be. This partly reflects that fact that the NSA recommendations intentionally go far beyond current computational capabilities, in order to maintain security into the future.
2. Exercise 6.15.

The plaintext in this cryptosystem will have format “a point \( P \) on the curve.” Alice and Bob will need to devise a method to encode the actual data they wish to communicate into this format, but we will assume that they have done so. Let \( N \) denote the order of the elliptic curve (the number of elements in the group); this is known to both Alice and Bob since there are efficient algorithms (e.g. Schoof-Elkies-Atkin) to compute it, and both know that the order of \( P \) divides \( N \).

Suppose that Alice has a plaintext \( P \), which she wishes to transmit to Bob. Alice and Bob can follow these steps.

(a) Alice chooses integers \( a, c \), where \( a \) is relatively prime to \( N \) and \( c \) is its inverse modulo \( N \). Similarly, Bob chooses integers \( b, d \), where \( b \) is relatively prime to \( N \) and \( d \) is its inverse modulo \( N \).

(b) Alice computes \( U = a \cdot P \) and transmits this point to Bob.

(c) Bob computes \( V = b \cdot U \) and transmits it to Alice.

(d) Alice computes \( W = c \cdot V \) and transmits it to Bob.

(e) Bob computes \( Q = d \cdot W \).

The points \( Q \) computed by Bob at the end will be equal to \((abcd)P\). Since \( bd \equiv ac \equiv 1 \pmod{N} \) and \( N \cdot P = O \), it follows that \( abcd \equiv 1 \pmod{N} \), so in fact \( Q = P\), because \( abcd = 1 + mN \) for some integer \( m \) and \( Q = P \oplus m(N \cdot P) = P \oplus O = P \). So Bob has determined Alice’s original plaintext \( P \).

3. Exercise 6.16.

Solution.

(a) If \((x, y)\) is one point on an elliptic curve, then the only other point with the same \( x \)-coordinate is \( \ominus(x, y) = (x, -y) \). This is is because any other point \((x, y')\) must satisfy \( y^2 \equiv y'^2 \pmod{p} \), hence \( (y + y')(y - y') \equiv 0 \pmod{p} \), which implies that \( y' \equiv \pm y \pmod{p} \), since a prime \( p \) cannot divide a product of two number without dividing one or the other.

If \( x^3 + Ax + B \equiv 0 \pmod{p} \), then the only possible \( y \)-coordinate is 0, so the point \( R \) is completely determined by \( x \) alone. On the other hand, if \( x^3 + Ax + B \not\equiv 0 \pmod{p} \), then there are two possible values of \( y \); if one \( y_0 \) then the other is \( p - y_0 \) when reduced modulo \( p \). These are two positive integers whose average is \( \frac{1}{2}p \), so one lies below \( \frac{1}{2}p \) and one lies above it. The value of \( \beta_R \) therefore uniquely identifies the actual \( y \)-coordinate.

(b) Bob can compute that \( x_R^3 + Ax_r + B \equiv 216 \pmod{p} \), and then obtain a square root by computing \( 216^{(p+1)/4} \equiv 487 \pmod{p} \) (by proposition 2.26). This is less than \( \frac{1}{2}p \), and \( \beta_R = 0 \), so Alice concludes that \((278, 487)\) is the point that Bob has in mind.

If Alice had instead said \( \beta_R = 1 \), then Bob would know that her point is \((278, 636)\), since \( 636 \equiv -487 \pmod{p} \).


Solution.
(a) The point $T$ is $n_A \cdot R = n_A k \cdot P = k \cdot (n_A \cdot P) = k \cdot Q_A = S$. Therefore $x_T = x_S$ and $y_T = y_S$. Therefore $c_1 \equiv x_T m_1$ and $c_2 \equiv y_T m_2 \mod p$. Multiplying by $x_T^{-1}$ and $y_T^{-1} \mod p$ this shows that $m'_1 \equiv m_1$ and $m'_2 \equiv m_2 \mod p$.

(b) The plaintext is two integers modulo $p$, and the ciphertext is one point on the elliptic curve plus two integers modulo $p$. Since the elliptic curve point has two coordinates, both integers modulo $p$, the ciphertext amounts to four integers modulo $p$. So the message expansion is two-to-one.

Of course, Alice and Bob could agree to use the technique from the previous problem to compress the size of the point $R$ by about half. So if this modification is performed, the message expansion becomes only three-to-two, or 1.5-to-one.

(c) Alice’s public key is $595 \cdot (278, 285) = (1104, 492)$. To decrypt $((1147, 640), 289, 1189)$, Alice computes $T = n_A \cdot R = 595 \cdot (1147, 640) = (942, 476)$, and then finds the plaintext as

\[
m_1 \equiv 942^{-1} 279 \equiv 509 \mod 1201 \\
m_2 \equiv 476^{-1} 1189 \equiv 767 \mod 1201
\]

So the plaintext is $(509, 767)$.

5. Suppose that Eve intercepts a Menezes-Vanstone (see the previous problem and Table 6.13 in the text) ciphertext $(R, c_1, c_2)$ send from Bob to Alice. Suppose that Eve knows, by other means, a list of 100 possible messages that Bob might send to Alice, and that the plaintext must be one of these (e.g. the message might be telling Alice which one of 100 locations they will meet in). Describe a method that Eve can use to determine which of these 100 candidates is the true plaintext, with relatively little computation (your method does not need to be completely fail-safe; it is ok if there are some extremely unlikely situations in which it will fail).

**Solution.**

Let the 100 possible plaintexts be $(m_{1,i}, m_{2,i})$ for $i = 1, 2, \ldots, 100$. For each candidate plaintext, Alice can determine a candidate for the point $S = kQ_A$ that Bob used in the encryption process. Namely, he can compute

\[
x_i \equiv c_1 m_{1,i}^{-1} \mod p \\
y_i \equiv c_2 m_{2,i}^{-1} \mod p
\]

for each $i$. He knows that for the correct value of $i$, the coordinates $(x_i, y_i)$ will in fact describe a point on the elliptic curve, i.e. they will satisfy

\[
y_i^2 \equiv x_i^3 + Ax_i + B \mod p.
\]

On the other hand, for the other 99 values of $i$, the values of the “candidate plaintext” is unlikely to be linked in a meaningful way to equation of the elliptic curve, so the left and right side of this congruence will be essentially unrelated to each other. Since $p$ is larger than the number of particles in the universe, it is highly unlikely that two unrelated numbers will be congruent modulo $p$ by coincidence.
Therefore it stands to reason that most likely only the “true plaintext” will produce coordinates \((x_i, y_i)\) of a point that actually lies on the elliptic curve. Eve can simply compute these 100 pairs of coordinates, check the congruence for each candidate, and identify the value of \(i\) for which these coordinates give a point on the curve. It is extremely unlikely that there will be more than one such value of \(i\). If there is, Eve will at least have narrowed high list of candidates to those that satisfy the elliptic curve congruence.

6. Samantha and Victor agree to the following digital signature scheme. The public parameters and key creation are identical to those of ECDSA. The verification procedure is different: to decide whether \((s_1, s_2)\) is a valid signature for a document \(d\), Victor computes

\[
\begin{align*}
    w_1 & \equiv s_1^{-1}d \pmod{q} \\
    w_2 & \equiv s_1^{-1}s_2 \pmod{q},
\end{align*}
\]

then he check to see whether or not

\[x(w_1G \oplus w_2V)\%q = s_1.\]

If so, he regards \((s_1, s_2)\) as a valid signature for \(d\).

(a) Describe a signing procedure that Samantha can follow to produce a valid signature on a given document \(d\). The procedure should be randomized in such a way that it will generate different signatures if executed repeatedly on the same document.

(b) Describe a “blind forgery” procedure that Eve can follow to create a signature \((s_1, s_2)\) and a document \(d\) such that \((s_1, s_2)\) is a valid signature for \(d\) under this scheme. Note that Eve does not need to be able to choose \(d\) in advance. The procedure should be randomized in such a way that it can generate many different forgeries (on many different documents).

**Solution.**

(a) Suppose that Samantha creates \(s_1\) in the usual way: she chooses an ephemeral key \(e\) and sets \(s_1 = x(eG)\%q\). Then to satisfy the signing equation can be rewritten

\[x(w_1G \oplus w_2V)\%q = x(eG)\%q.\]

To satisfy this, it suffices to ensure that

\[w_1G \oplus w_2V = eG\]

which is equivalent (since \(V = sV\)) to \((w_1 + w_2s)G = eG\). Since the multiples of \(G\) are periodic with period \(q\) (since the order of \(G\) is \(q\)), this is equivalent to the congruence

\[
\begin{align*}
    w_1 + w_2s & \equiv e \pmod{q} \\
    s_1^{-1}d + s_1^{-1}s_2s & \equiv e \pmod{q} \\
    d + s_2s & \equiv s_1e \pmod{q} \\
    s_2 & \equiv s^{-1}(s_1e - d) \pmod{q}
\end{align*}
\]

This line congruence is what Samantha can use to compute \(s_2\). In summary, her signed equations are the following (once she has chosen an ephemeral key \(e\)).
\[
s_1 \equiv xe \cdot G \pmod q \\
s_2 \equiv s^{-1}(s_1e - d) \pmod q
\]

(b) We follow the same strategy as exercise 4.7 in the textbook (Problem Set 7, problem 5) and problem 11 on problem set 8 (which concerned analogous constructions for Elgamal signatures and DSA).

First, select the first part of the signature, not using a multiple of \(G\), but instead a combination of \(G\) and the verification key \(V\). Eve can select two numbers \(i, j\) at random and set

\[
s_1 = x(iG \oplus jV) \pmod q.
\]

Similar to part (a), Eve can guarantee that the signing equation holds by ensuring that

\[
\begin{align*}
w_1G \oplus w_2V &= iG \oplus jV \\
{s_1}^{-1}dG \oplus s_1^{-1}s_2V &= iG \oplus jV
\end{align*}
\]

One easy way for Eve to ensure that this equation holds is to simply ensure that the coefficients of \(G\) and of \(V\) agree individually (modulo \(q\)), i.e. \(s_1^{-1}d \equiv i \pmod q\) and \(s_1^{-1}s_2 \equiv j \pmod p\). Even can arrange this by setting \(d \equiv is_1 \pmod q\) and \(s_2 \equiv js_1 \pmod q\).

In summary, Eve can use the following steps.

i. Choose integers \(i, j\) at random.
ii. Compute \(s_1 = x(iG \oplus jV) \pmod q\).
iii. Compute \(s_2 \equiv js_1 \pmod q\) and \(d \equiv is_1 \pmod q\).
iv. Present \((s_1, s_2)\) as a signature on the document \(d\).

The analysis above shows that \((s_1, s_2)\) is indeed a valid signature on the document \(d\). Note that there are two degrees of freedom \((i, j)\), so this method will produce a wide range of signatures and a wide range of documents. Of course, as in other “blind signature” schemes, Eve does not get to choose in advance which document she will be signing.

**Programming problems**

Full formulation and submission: [https://www.hackerrank.com/m158-2016-pset-9](https://www.hackerrank.com/m158-2016-pset-9)

*Note.* Both of these problems will make use of public parameters specified in this document.


You certainly do not need to read and understand the whole document, just find the information you need for the algorithms. You’ll need to look up how to convert hexadecimal strings to integers.

7. Write a program that determines whether a given ECDSA signature \((s_1, s_2)\) for a document \(d\) is valid or not, given parameters and an ECDSA public key (notation as on page 322 of the textbook). The signatures will all use curve P-384 from the document above.
Solution.

Using previously written functions for addition and multiplication on elliptic curves, the verification procedure can be implemented by following Table 6.7. It is necessary to extract the necessary parameters from the NIST document, and in particular to convert several number from hexadecimal, which can be accomplished using the syntax \texttt{int(S,16)} (where \texttt{S} is a hexadecimal string). An implementation is below (with source for previously written functions omitted).

```python
def verify(d,s1,s2,V):
    # Parameters for P-384, from standard document
    p = 39402006196394479212279040100143613805079739270465446667948293404245721771496
     870329047266088258938001861606973112319
    q = 39402006196394479212279040100143613805079739270465446667946905279627659399113
     263569398956308152294913554433653942643
    A = (-3)%p
    Bstring = 'b3312fa7 e23ee7e4 988e056b e3f82d19 181d9c6e fe814112 0314088f 5013875a
     c656398d 8a2ed19d 2a85c8ed d3ec2aef'.replace(' ','')
    B = int(Bstring,16)
    Gxstring = 'aa87ca22 be8b0537 8eb1c71e f320ad74 6e1d3b62 8ba79b98 59f741e0 82542a38
     5502f25d bf55296c 3a545e38 7276ob7'.replace(' ','')
    Gx = int(Gxstring,16)
    Gystring = '3617de4a 96262c6f 5d9e98bf 9292dc29 f8f41dbd 289a147c e9da3113 b5f0b8e3
     0ada60b1ce 1d7e819d 7a431d7c 90ea0e5f'.replace(' ','')
    Gy = int(Gystring,16)
    G = (Gx,Gy)
    w1 = inverse(s2,q)*d % q
    w2 = inverse(s2,q)*s1 % q
    Q = add(mult(G,w1,A,B,p),mult(V,w2,A,B,p),A,B,p)
    return Q[0]%q == s1
```

```python
### I/O
xv,yv = map(int,raw_input().split())
d,s1,s2 = map(int,raw_input().split())

if verify(d,s1,s2,(xv,yv)):
    print 'valid'
else:
    print 'invalid'
```

8. Write a program to decipher messages enciphered with the Menezes-Vanstone cryptosystem described in problems 6 and 7 (textbook exercises 6.17 and 6.18), given a private key and a ciphertext. The public parameters will be those of curve P-192 from the above document.
Solution.
We can follow the steps and notation of Table 6.13. Here is an implementation (omitting source code for previously written functions).

```python
### Omitted: source of functions "add" and "mult" (elliptic curve operations),
### and "inverse" (modular inverse)

def decipher(c1,c2,xr,yr,na):
    # Parameters of P-198
    p = 6277101735386680763835789423207666416083908700390324961279
    q = 6277101735386680763835789423176059013767194773182842284081
    bstring = '64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1'.replace(' ','')
    Gxstring = '188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012'.replace(' ','')
    Gystring = '07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811'.replace(' ','')
    A = (-3)%p
    B = int(bstring,16)
    Gx = int(Gxstring,16)
    Gy = int(Gystring,16)

    R = (xr,yr)
    xt,yt = mult(R,na,A,B,p)
    m1 = inverse(xt,p)*c1 % p
    m2 = inverse(yt,p)*c2 % p
    return m1,m2
```

### I/O
na = int(raw_input())
xr,yr = map(int,raw_input().split())
c1,c2 = map(int,raw_input().split())

m1,m2 = decipher(c1,c2,xr,yr,na)
print m1,m2
```