Problem Set 4

This problem set is due **at 10:00pm** on **Monday, April 22, 2019.** Total Points: **167**

Please make note of the following instructions:

- We would recommend attempting EC questions in order.
- Please start the assignment as early as possible - requests for extensions will not be entertained except under exceptional circumstances.
- Ideally submit *only* one zip file, with all your programs appropriately named, and your README should clearly indicate how to run your program.

how they operationally different?How point doubling is different from traditional multiplicatino mod n. A mathematically technical answer not expected.

Problem 4-1. Elliptic Curves - Warmup 1 : Swap knight for rook? (8) [10 points]

Describe briefly the difference between the Elliptic Curve group and the Z_p^* group. What are the differences? What are the similiarities?

Problem 4-2. Elliptic Curves - Warmup 2 : Hope rations will be enough to work at the theatre. (9) [15 points]

Quickly describe how to add two points to an elliptic curve in the modulo n world (call these points x_1, y_1 and x_2, y_2). How would we double these points? Under what situations would the result be the *point at infinity*?

Problem 4-3. Elliptic Curves - Warmup 3 : An instance of my former partner is enough. (7) [10 points] Give an example of a point doubling over the elliptic curve characterized by the parameters (11,3,3), that you computed by hand. (For instance, over the Elliptic curve $y^2 \equiv x^3 + 2x + 2$ mod 17, doubling $(5, 1)$ gives us $(6, 3)$)

You cannot cheat and simply give me the point doubling of the *point at infinity*. After this - you should be ready for the most important question of the assignment, which is...

Problem 4-4. Try a tool. (9) (credits. Niyati Bafna for clue) [100 points]

The goal of this assignment is to implement some of the arithmetic needed for elliptic curve cryptography using the GNU Multiple Precision Arithmetic Library. The GMP integer functions provide most of the tools you will need. Pay special attention to the integer functions portion of GMP.

GMP supports both a C and a C++ interface. You may use either one for this assignment. GMP was originally written for C, and much of the documentation refers to the C APIs. If you choose to use C++, you will have to understand how the C++ class interface works.

Your assignment is to write functions to do the following:

(a) [10 points] Read a file of EC domain parameters as described below. doubt

(b) [10 points] Read an EC point from an open stream (i.e. standard input).

(c) [10 points] Write an EC point to an open stream (i.e. standard output).

(d) [10 points] Add two EC points.

(e) [10 points] Negate an EC point.

(f) [10 points] Subtract two EC points.

(g) [20 points] Perform point multiplication $k \times P$ of a big integer k and a point P by the method of repeated doubling.

Shanks algo

(h) [30 points] Given a number $x \in Z_p$, find y such that the point (x, y) lies on the curve, or return a flag indicating that no such point exists.

In addition to the functions, you should produce one or more programs and data sets to test each of your functions and verify that they work correctly.

For the functions that work on arbitrary size integers, you should test them first with small numbers for which you can verify the answers by hand and make sure they are correct. Then you should test them with numbers that are at least 40 decimal digits long to make sure that the big number arithmetic is working correctly.

test cases !

Tests of the addition function should include test data that exercises all five cases of the definition (from your slides). The test for subtraction of $P - Q$ should verify the result R by checking that $R + Q = P$. Point multiplication should be checked first for small values of k by adding P to itself k times. Then it should be checked for large pairs of number k and m to ensure that $k \times (m \times P) = (km) \times P$. Finally, to verify the point-on-curve function (see reference to Shank's algorithm below), check that the returned point P satisfies equation 1.

Deliverables

You should submit the following items:

- A makefile, all source code and header files needed to build your project.
- Test data files and the output from your code when run on them.
- A brief human-readable document with information about your code such as known bugs, procedures for building and running it, and anything else that might help the TAs.

Some Background on Elliptic Curves

For this assignment, we consider the elliptic curve $\mathcal{E}$ over Z_p whose defining equation is

$$y^2 \equiv x^3 + ax + b \mod p \tag{1}$$

The curve $\mathcal{E}$ comprises all of the points $(x, y) \in Z_p \times Z_p$ which satisfy equation 1, together with a special point "at infinity" denoted $\mathcal{O}$. We assume that p is an odd prime and that a and b satisfy the condition

$$4a^3 + 27b^2 \not\equiv 0 \mod p \tag{2}$$

This ensures that the curve is non-singular.

A special kind of addition can be defined on the points in $\mathcal{E}$ called elliptic curve addition. This operation is usually denoted by the ordinary "+" symbol, but one should remember that its arguments are points, not elements of the field Z_p.

Elliptic curve addition is defined in your lecture slides. Note that there are five defining cases for X + Y depending on whether or not X and Y are the infinity point, and if not, whether or not they agree in their x-coordinates and y-coordinates.

There is also a unary operation of elliptic curve negation, written "$-$". As with ordinary arithmetic, negation has the property that $-P$ is the additive inverse of P, so that $P + (-P) = (-P)$+$P = \mathcal{O}$. Negation is easily computed. $(-O) = O$, and for a point $P = (x, y)$, $(-P) = (x, -y)$. It follows immediately from the definition of addition that $P + (-P) = O$ as desired. For convenience, one usually defines the binary subtraction operator in the usual way as $P - Q = P + (-Q)$.

Elliptic curves support another kind of operation that we call *point multiplication*. We denote it using the $\times$ symbol, but this is not the usual meaning of multiplication, as the first argument must be a non-negative integer k and the second a point $P \in mathcalE$. It is defined to be the result of adding P to itself k times. Thus, $3 \times P = P + P + P$, where P is a point on the elliptic curve. Note that $\times$ is "associative" in the sense that the following equation holds for all non-negative integers k and m:

$$k \times (m \times P) = (km) \times P$$

where km is an ordinary integer product.

Point multiplication on elliptic curves is analogous to exponentiation. a^n means "multiply a by itself a total of n times". Similarly, $n \times P$ means "add P to itself a total of n times". In both cases, these are easily computed by a loop that performs n separate operations, and in both cases, this approach becomes infeasible when n is large.

For exponentiation, a fast algorithm based on repeated squaring was presented in lecture. A similar technique based on repeated doubling will work to reduce the work to compute $n \times P$ from $O(n)$ to $O(logn)$.

Another operation on elliptic curves we might need is to find a point on the curve with a given x-value, if it exists. This entails solving equation 1 for y. Of course, if the right hand side is not a quadratic residue modulo p, then no such solution is possible.

We will cover Shank's algorithm (in review, if not directly in lecture), which will give us a reasonably efficient means of finding modular square roots when they exist. Because the algorithm as presented assumes that the number a (whose square root is desired) is in QR_p, you should either compute the Legendre symbol $\left(\frac{a}{p}\right)$ to verify that $a \in QR_p$ before running Shank's algorithm, or you should modify Shank's algorithm appropriately so that it detects when $a \in QNR_p$ and gives an appropriate output.

Data Representation

An elliptic curve is described by three domain parameters p, a, b from equation 1. We will specify an elliptic curve by a text file that contains a sequence of whitespace-separated unsigned decimal big numbers, the first three of which are p, a, b, respectively. As usual, you may assume that p is an odd prime and that a and b satisfy constraint 2.

External representations: Numbers in files are represented as arbitrary precision optionally signed decimal integers. They can be read and written using `mpz_inp_str` and `mpz_out_str`, respectively. (C++ users should be able to use the standard operators >> and <<.) EC points should be represented by pairs of signed decimal arbitrary precision integers. The special point $\mathcal{O}$ is represented by the string "`-1 -1`". The point (x, y) is represented by the string "`x y`", where x and y are the decimal representations of the nonnegative numbers x and y, respectively.

Internal representations Big integers should be represented internally using GMP big integers (type `mpz_t` if using the C interface, and class `mpz_class` if using the C++ interface). An EC point should be represented by a struct or class containing two big integers representing the x and y components of the point. Because x and y are always non-negative, the special point $\mathcal{O}$ can be represented unambiguously by the pair $(-1, -1)$. However, $\mathcal{O}$ must still be tested for and treated specially in your code since it is not correct to compute on this pair as if it were an ordinary EC point.

Problem 4-5. Elgamal Digital Signatures [32 points]

Write a program to implement the following variant of the ElGamal digital signature scheme. You can use functions that you wrote for ElGamal encryption scheme in the previous assignment. [20 points]

- **KeyGen** : This algorithm proceeds as follows,
  - –Choose two primes p and q such that $q = 2p + 1$, and p is a 300-bit prime
  - –Choose g such that g is the square of a primitive root mod q
  - –Choose a random element $a \in \{2, 3, ..., p - 1\}$
  - –Compute $h = g^a \mod q$
  - –Set $\mathsf{PK} = (q, g, h); \mathsf{SK} = a$

write to a public_key.txt

- **Sign** : To sign a message m under the secret $\mathsf{SK} = a$, this algorithm proceeds as follows:
  - –Choose a random element $k \in \{2, ..., p - 1\}$
  - –Compute $r = g^k \mod q$
  - –Compute $s = (m - ar)k^{-1} \mod p$
  - –Output signature $\sigma = (r, s)$

read from public_key.txt

- **Decrypt** : To verify a signature $\sigma = (r, s)$ under the public key $\mathsf{PK} = (q, g, h)$, this algorithm proceeds as follows:
  - –Output True if $g^m \equiv h^r r^s \mod q$
  - –Else output False

1. Using the above program, show that an existential forgery attack is possible as follows [5 points]:
   - Choose a random element $z \in \{2, 3, ..., p - 1\}$
   - Compute $r = g^z h \mod q$
   - Compute $s = -r \mod p$
   - Compute $m = zs \mod p$
   - Verify($\sigma = (r, s), m$) = True
2. Now replace m by $H(m)$ in the program where H is a SHA-256 hash function. Test your program for correctness. You can use hashing libraries for this task. [5 points]
3. After the above change, is your program still susceptible to existential forgery attack as labelled in 1? Comment. [2 points]