Math 3159A Group Assignment 5 Fall 2020

Instructions

- This assignment is due on Tuesday, December 8, 2020 at 2:00 PM EST. Late submissions will **not** be accepted.
- This assignment consists of one problem with two parts. You must submit both parts to receive full credit.
- Your solution needs to be formatted using the LaTeX template available on OWL. Note that there are different templates available for regular assignments and group assignments. You should use the one for group assignments.
- All group members are expected to be working on the solution and every member should attend all group meetings.
- The Scribe will be submitting the assignment on behalf of the group. It is assumed that every member of the group has proofread the submission.
- All solutions must be written in full sentences.
- You are not allowed to use online resources and should only discuss the solution with members of your group.
- This assignment is worth 5 points.

Part 1.

In this assignment, we will investigate a pseudo-random number generator that uses elliptic curves.

A *pseudo-random number generator* is an algorithm F that takes as input a number s_0 (the *seed*) and outputs a pair $F(s_0) = (s_1, t_1)$, in which s_1 is the new seed and t_1 is a *pseudo-random number*. When used repeatedly, such an algorithm generates a sequence:

$$s_0, \qquad F(s_0) = (s_1, t_1), \qquad F(s_1) = (s_2, t_2), \qquad F(s_2) = (s_3, t_3), \qquad \dots$$

This way, two parties that share a common secret s_0 (established for instance using the Diffie–Hellman Key Agreement) have access to the same list of pseudo-random numbers t_1, t_2, t_3, ..., which can be used in communication via one-time pads.

Such an algorithm is considered *secure* if a third party that sees pseudo-random numbers t_1, t_2, t_3, ..., t_k cannot predict t_{k+1} (since they do not have access to s_i's).

We will investigate the following pseudo-random number generator. A trusted public party publishes the following quintuple (p, A, B, P, Q), where:

- p is a large prime;

- $A, B \in \mathbb{F}_p$ such that $4A^3 + 27B^2 \neq 0$;
- $P, Q \in E(\mathbb{F}_p)$ where E is an elliptic curve given by the equation $y^2 = x^3 + Ax + B$.

The user then chooses a secret element $s_0 \in \mathbb{F}_p$ (the *seed*) and computes:

$$\begin{aligned} r &= x(s_0 P) \\ s_1 &= x(rP) \\ t_1 &= x(rQ) \end{aligned}$$

(Here, we write $x(P)$ to indicate the x-coordinate, i.e., an element of $\mathbb{F}_p$, of the point P.) The algorithm returns the pair (s_1, t_1), where s_1 is the new seed and t_1 is a pseudo-random number.

Suppose that the trusted public party is not so trustworthy after all and in particular it knows a natural number e such that $P = eQ$, while not disclosing this knowledge to the public. Show that this party can then identify the value of the new seed s_1 in $\mathbb{F}_p$, given the value of a generated pseudo-random number t_1. As a result, this party would be able to predict the next pseudo-random number t_2.

Part 2.

1. Write a function in Python3 called `solve` that, given the input $(p, A, B, P_x, P_y, Q_x, Q_y, e, t_1)$ where

   - p is a large prime;
   - $A, B \in \mathbb{F}_p$ such that $4A^3 + 27B^2 \neq 0$;
   - $P = (P_x, P_y)$ and $Q = (Q_x, Q_y)$ are points on the elliptic curve E over $\mathbb{F}_p$ given by the equation $y^2 = x^3 + Ax + B$;
   - e is a natural number such that $P = eQ$;
   - t_1 is the x-coordinate of certain multiple of Q on E,

   outputs the value of the next pseudo-random number t_2.

   Your program *must* implement the algorithms described in Part 1 of this assignment. All other functions will receive no credit.

2. Download the file `generate_input.py`, and use it to obtain 3 sets of tuples of the form $(p, A, B, P_x, P_y, Q_x, Q_y, e, t_1)$ by running

   ```
   python generate_input.py [last three digits of your student number]
   ```

3. Run your method `solve` on all these inputs.

As part of your submission, include:

1. The *Python code* implementing your solution;
2. The 3 *inputs you generated*, and the *output of your program* run on these inputs. One input and one output per line.

Examples

Here are some examples of what your function `solve` should do.

```
>>> solve(32416187567, 100, 300, 27957624436, 9381314875, 4926003430, 24870962866, 17242042885, 3564697884)
4690041109
>>> solve(32416188191, 54, 456, 27828875679, 24709261957, 4979256242, 12312669996, 10269365243, 1308878353)
18039526984
>>> solve(32416188127, 19, 33, 16475848216, 5118271045, 19262187694, 2854205065, 13332545241, 18635505014)
16371031443
```

Notes

- Incorrect answers will be penalized more than missing answers. (It is straightforward to verify the correctness of your submission!)
- Make sure that your algorithm terminates on the inputs we provide.
- You may not use any trivial brute-force algorithms. You must implement the algorithm developed in Part 1 of the assignment.
- The file `generate_input.py` is written in Python3, and so should be your solution. Make sure you are using a 64bit version of Python3.
- Your code should not make use of any external libraries such as `numpy` or `math`. All the auxiliary functions should be implemented by you, and should be included in your submission. You should only use the most basic arithmetic operations such as `+`, `-`, `*`, `//`, `%`.
- Comments in the code are not mandatory. However in the case of an incorrect solution, the comments can provide grounds for partial credit.