Module 04: Elliptic Curve Cryptography and Lattice Cryptanalysis

Interacting with the Server

Jan Gilcher, Kenny Paterson, and Kien Tuong Truong
jan.gilcher@inf.ethz.ch, kenny.paterson@inf.ethz.ch, kitruong@ethz.ch

November 2023

Hi all! Welcome to the last module for the semester!

This module's lab session will work slightly differently from the previous modules. Instead of splitting up students and TAs over several rooms, we will have one big lab session every week with all the TAs for this modules present. At the beginning of each lab session we will give a short presentation with helpful information. In this first lab for example, that will include how to setup the SageMath environment you'll be using to solve the labs.

The room for the lab sessions is: **ML D 28**.

This lab will also follow a Capture The Flag (CTF) approach, that you might already know from a previous module. For this you will be required to interact with our CTFd instance, where you will find the code and be able to submit flags. You have already received the credentials to for your account on the server via email (if you have not message us ASAP!). The server is available at:

`https://isl.aclabs.ethz.ch`

As with previous modules, you are also welcome to post your questions and to join ongoing discussions on the Moodle forum at the following link:

`https://moodle-app2.let.ethz.ch/mod/forum/view.php?id=911956`

1 Overview of the Server

This sheet is intended to give you an overview of the communication between you and the server. The server essentially exposes a JSON RPC interface that you can use.

For each challenge, we provide you with two files: a `boilerplate.py` file and a `server.py` file. If you read this document, you will not need to read anything from the `boilerplate.py` file.

Let us look at the server for the actual exercise: M0.2: Signing, with Errors.

The server is entirely contained in a class `SignServer` (the name can change) which subclassess `CommandServer`. We then have a few methods: `__init__`, `handle_signature`, `handle_getpubkey`, `initialize_new_round`, `solve`, and `flag`.

1.1 The `__init__` method

This method will be called *every time you connect to the server*. What this means is that the server will be initialized with a flag, it will set the score to 0, it will create an ECDSA instance, as well as *generating a new ECDSA keypair* every time you connect to it.

In the server code we provide you, the server is initialized at the bottom of the file with a test flag. This allows you to run the server locally by running `$ sage server.py`. If you make the attack work locally, you will receive the test flag `flag{test_flag}`. Remotely, we will change this to a different flag that you will have to recover.

1.2 Message handlers

When you connect to the server, you can ask it to execute a command. To choose a command, send a JSON containing a "command" field with the name of the command as the value. *The name of the command is the one contained in the decorator* `@on_command` *above the method.*

For instance if you wanted to call the first command, `get_signature`, you would send a JSON like:

```
{"command": "get_signature"}
```

This will automatically call the method `handle_signature`.

The handler may require other arguments. To see the list of arguments you should check the body of the method. For instance, the command `solve` requires that your message contains a field called `"b"`. In this case, `b` is an integer 0 or 1, which gets checked against another integer `self.correct_signature`.

Let's say that you want to guess that `self.correct_signature` is 1, then you would send a JSON like:

```
{"command": "solve", "b": 1}
```

Note that we send the value without quotes (i.e. as an int, rather than a string). This will automatically be deserialized by the server as an integer. *Be careful about the types that you send, as objects of two different types will be regarded as different by Python* (i.e. `"1"` $\neq 1$).

Your objective for the challenge will be to be able to call the `flag` command when you have enough correct guesses.

1.3 Startup handler

The method decorated with the `@on_startup` decorator will be executed before the server begins listening for commands. In this case, it is necessary to initialize a new round with a new message, and a new signature. This method is then called again to initialize a new round everytime you guess correctly.

1.4 Running the server locally

To run the server locally you can simply run:

```
$ sage server.py
```

In our case, the server also has a dependency on a `ecdsa2.py` file and a `boilerplate.py`, which you should place in the same folder as the server.

Your server is now reachable at `("localhost", 40102)`.

2 The client

Your client can be based on the template we give you in challenge M0.1. The template has two functions ready for you: `json_send` and `json_recv`.

Remember to change the value of `PORT` before attempting the challenge.

The function `json_send` can be used to send a command to the server. It expects a Python dictionary and converts it for you to a JSON, and sends it to the server. Be mindful that not all types can be serialized to JSON.

On the other hand, the function `json_recv` receives a byte stream from the socket and automatically converts it to a Python dict for your convenience.

The client will usually try to run against a local server. Set the environment variable `REMOTE` to true in order to change this behaviour. For example:

```
$ REMOTE=true sage solve.py
```

You are not forced to use our template, **but do make sure that, when running your script with the above command, your script connects to the remote server, NOT the local server**.