Imperial College London
Department of Computing

Multi-party Computation Coursework

https://www.doc.ic.ac.uk/~nd/peng/coursework.pdf

01 The goal of this coursework is to gain a better understanding of multi-party computation by writing a program that implements the BGW protocol to securely (privately) evaluate a function defined as an arithmetic circuit in the semi-honest model.

02 In particular, you will implement the method described in the last chapter (section 22.3) of Nigel Smart's ***Cryptography Made Simple*** book but as a distributed program[1].

03 The supported[2] programming language for the coursework is **Python3** running on Linux or MacOS. You can use you own computer if you have a recent version of Python3. The lab machines have version 3.8.5 installed. The latest stable version at the time of writing this specification was 3.9.0.

04 Your submitted code must be runnable on a CSG Linux machine or a Mac (running Catalina or Mojave).

05 A typical Python solution will be around 500 lines including about 300 lines of supplied code.

06 You will be provided several Python files to help you. In particular, (i) to create processes and send messages between parties, (ii) to carry out arithmetic modulo a prime number, (iii) a sample configuration file with two circuits for testing.

07 Start early. Although the protocol is not particularly difficult to code, it is new to you, and will require careful development to get it working correctly. Your program will also be running as multiple Linux/Mac processes which may also be new to you.

08 You can work on and submit the coursework either **individually or jointly with one classmate.**

09 The deadline for submission is **Thursday 19th November 2020**.

10 Use **Piazza** if you have questions about the coursework.

11 **You must not post your solutions or share them with others.** Email me directly if you have a specific question about your solution. The most up-to-date version of this specification will be on my course webpage (see URL above).

[1] Each MPC party will run in its own Linux/Mac process on the same machine not on different machines.

[2] Contact me if you really want to use a different programming language.

GETTING STARTED

12 Download the e-book version of ***Cryptography Made Simple*** by Nigel Smart from the College library.

13 Also download the **errata** for the book at https://homes.esat.kuleuven.be/~nsmart/errata.html In particular, please **note the corrections for page 447** (point 38). These corrections are important if you are trying to understand the example and/or replicate some of the tables in the example.

14 Review the slides and relevant parts of chapter 22 (in particular section 22.3) before starting on the implementation. If you're very technically minded, you may also wish to read the original paper[3] - warning it is of poor-quality print and requires a mathematical/cryptography background. There are many other descriptions on the web.

15 Develop and test your program step-by-step. It is easy to make mistakes and get in a muddle. Don't add too much functionality in one go. Implement a single ADD gate first, then circuits with multiple ADD gates, and then extend to include MUL gates. Use a small prime number and repeatable random numbers for debugging. Use the testing, debugging and version control techniques and tools that you're familiar with.

16 The classes provided in **`pubsub.py`** (see below) can be used for sending messages between parties. The classes use **ZeroMQ** (ØMQ), an elegant message-passing library that has bindings for numerous programming languages (https://zguide.zeromq.org/). If you're using your own Python3 installation you should be able to install ZeroMQ with `pip3 install pyzmq`.

[3] *Completeness theorems for non-cryptographic fault-tolerant distributed computation* *https://doi.org/10.1145/62212.62213*

SUPPLIED FILES

17 The following files are provided to help you get started. Download and unzip the files in https://www.doc.ic.ac.uk/~nd/peng/cwfiles.zip. You are permitted to change these files or not use them at all. I've tried to keep the supplied code very simple. Do review it before starting and please inform me of any mistakes.

Makefile	Use the command **`make sort`** to run mpc.py. The output is buffered then sorted to allow outputs from different parties to be sorted. If you use **`log.write`** to print lines, each output line will be prefixed by the number of the party and a line number to facilitate sorting. Actual execution is in parallel across the CPU cores of your computer. You can use **`make`** for a normal interleaved printout (useful for debugging). The program will terminate all party processes after a set max time (set in config.py). If for any reason you still have party processes running, use **`make pkill`** to remove.
config.py	Sets a number of general parameters. Adjust as required for your setup.
circuit.py	Defines the MPC function(s) to be evaluated as circuits. Also defines the private values of each party, the polynomial degree (T) and the prime number to use for modular arithmetic. Two circuits are provided. Vary for testing and extend with your own circuit.
log.py	Includes print functions to help print sorted output lines.
modprime.py	Functions to do arithmetic modulo the prime number defined in **`circuit.py`**
mpc.py	Invoked by the **`Makefile`** to run the program. Will create processes for the parties and terminate them after a set time. Each party process will call **`party.bgw_protocol`** with setup parameters to run your code.
network.py	Provides methods to send and receive Shamir secret shares between parties (i.e. between Linux/Mac processes). Parties run asynchronously. So, shares are tagged with their gate when sent and buffered for their gate when received.
party.py	**You need to write this!** The function **`party.bgw_protocol`** will be called from **`mpc.py`** when the party process starts. Do not write a monolithic function, split you code into suitable modules and functions. One decision you will need to make is your choice for representing arrays. Most descriptions of the protocol use 1 as the base for 1 and 2 dimensional arrays. Python lists start at 0, so be careful when coding. Add a redundant 0-th element if it helps. Alternatively, use Python dictionaries indexed from 1. I'm a fan of dictionaries and used them in my implementation. The choice is yours however.

SUBMISSION

18 Submit your work as a single zip file (`mpc.zip`) using **CATE**. Do a **`make clean`** before zipping. I will be marking electronically, so please do not use very long lines. The Python documentation recommends 4 spaces for indents and lines less than 80 characters (I use 2 spaces for indents).

19 Include in your submission:

(a) Your program plus any additional instructions (`readme.txt`) for running it. I will run your program either on a CSG lab machine or on my Mac.

(b) Your code must include your name(s) and login(s) at the beginning of any files that you write.

(c) A short report (`report.pdf` - max 4 A4 pages) that demonstrates that your solution works for the 2 circuits provided in **`circuit.py`** plus **one** circuit of your own e.g. it may be good for validating solutions, or implement an interesting function/topology. Do not tell others what circuit you've implemented. How you structure and present your report is left to you to decide. Again include your name(s) and login(s) at the beginning.

(d) Optional: Any feedback you have about the coursework, good or bad, including how it could be improved. Submit this as a `feedback` text file or `feedback.md` markdown file. Your feedback will not affect your mark either way!

MARKING SCHEME

20 The coursework will be marked out of 20 using the following scheme:

	Marks
Code for BGW protocol	12
Report	6
Your circuit	2

21 I will be looking for correct implementation, clean design, readable code, helpful comments, plus a well-written report, and a good/orginal/interesting circuit.

22 If you don't manage to complete all parts of the coursework, clearly state what's incomplete or not working e.g. '*my solution is not working for XX*'. If you have some idea where the problem lies in the code, indicate it in your submission.