Project 1: SPDZ

CMSC398U

Due: 2024-11-20, 11:59 PM

1 Introduction

In this project you will be implementing `slows`, a simple client for the online SPDZ maliciously secure MPC protocol.

You may use any language we are able and willing to run. Python, SageMath, Java, C, C++, and Rust are explicitly allowed; please ask if you want to use another language. Some C source code for the `slows` client is provided in the Files tab of ELMS. Note for other languages, you may need an external library for computing SHA256 hashes.

2 Configuration

On startup, all parties will connect to each other via the connections defined in each party's host configuration file. The format of a single party's host configuration file is as follows:

- The host file begins by providing the name of the party.
- Each line afterwards declares a connection to another party as follows:

```
[name] [host port] [client addr] [client port]
```

For example, for party `p0`, we may have

```
p0
p1 8000 127.0.0.1 8010
```

And for `p1`, we have

```
p1
p0 8010 127.0.0.1 8000
```

Every party will connect to every other party, for the purpose of broadcasting messages. Example host files are provided in the example tests on ELMS.

3 Circuits

Each party is required to take in a circuit file; this defines the circuit along with the private inputs of the party. A circuit consists of input and constant wires (along with the input values), wire assignments, and output wires. The circuit file stores wires *in order of evaluation*, (wires referenced must be initialized prior to its use). The format is as follows:

- The circuit file consists first of a set of input wires. For those input wires held by the host party, input values must be provided. The syntax is as follows:

  ```
  [wire] = inp [party] [input]
  ```

  For example, for party 0, the input wires may look like this:

  ```
  w1 = inp p0 1
  w2 = inp p1
  ```

  and for party 1,

  ```
  w1 = inp p0
  w2 = inp p1 32
  ```

  Additionally, one may have constant wire inputs; where are notated as follows:

  ```
  [wire] = con [value]
  ```

- After the input wires are defined, the circuit file consists of a set of gate assignments, *in order of evaluation.* Specifically, a gate assignment can be one of the following:

  ```
  [new wire] = [wire in 1] + [wire in 2]
  ```

  or

  ```
  [new wire] = [wire in 1] * [wire in 2]
  ```

- Finally, the file consists of a set of output wires. The following is used to notate an output wire:

  ```
  out [wire]
  ```

4 Preprocessing

Each party is additionally required to take in a preprocessing file which provides the beaver triples and authenticated randomness. The format is as follows:

- The first line consists of the share of the global MAC key Δ_i, as follows:

  ```
  mac [share]
  ```

- For every wire, a share $([r]_i, [\Delta r]_i)$ of a random value is provided. If the party holds the input for the wire, r is additionally provided.

  ```
  rand [wire] ([share], [share_mac])
  ```

  If the wire is held by the party, we additionally hold the random value

  ```
  rand [wire] ([share], [share_mac]) [input]
  ```

- Finally, for every multiplication gate, a Beaver triple is provided, in the following form:

  ```
  triple ([a], [a_mac]) ([b], [b_mac]) ([ab], [ab_mac])
  ```

5 Implementation

5.1 Source Code

Source code is given in the Files section of ELMS. This code parses the three input files. All of the additional code should be implemented in `online.c`, the rest of the template is already filled out.

5.2 Command Line options

The program should be run as follows :

```
slows -h [host file] -c [circuit file] -p [preprocessed file]
```

Each flag is mandatory; **note that the input files are party dependent**.

6 Resources

Some resources you may find useful:

- The chapter on SPDZ, Pragmatic MPC
- The chapter on Beaver triples and preprocessing, Pragmatic MPC
- MP-SPDZ on Github, an implementation of SPDZ.