SLTBP

Lukas Helminger, Fabian Schmid

May 2020

Introduction

Over the past half-year, we developed a multi-party computation (MPC) use case. In our project, we want to enable small companies to buy from a much larger enterprise. With MPC (Multi Party Computation), small companies would be able to keep their requested amount secret from their competition. Even the vendor will only discover the required items and suggested prices when the protocol finishes with success.

In the first version, every customer submits their requested amount and suggests a price. Then everyone participates to secretly compare the required amounts with the available stock of the vendor. In parallel, all participants determine the average offered cost per unit and weigh it to the minimum price of the vendor. If the offers meet the prerequisites of a deal, everyone reveals their distinct amount and the total cost of the order to the vendor. The clients only learn whether the transaction was possible or not, even if they try to attack the protocol.

As a foundation for our project, we use FRamework for Efficient and Secure COmputation (FRESCO). This framework provides the necessary functionality to build new MPC protocols such as ours. From the demonstrator projects, we could quickly derive how to make our protocol. From there, we needed to add the security guarantees which we claimed. We did not only want to write our protocol but also provide a starting point for future work. Providing a starting point seemed necessary to us, since setting up security against malicious adversaries in FRESCO can be quite tedious.

Since FRESCO is available in the public maven repository, we also developed our extension using maven. The use of maven assures ease of integration. In the end we want to have everything assembled into a single jar with dependencies called the priceFinder. Parameters on startup define whether the priceFinder acts as the host or the client.

FRESCO

As a foundation for our project we use FRamework for Efficient and Secure COmputation (FRESCO). This framework provides the basic functionality to build new MPC protocols such as ours. This framework can be understood as our main computation engine. We need to setup all the parameters needed and also provide some initialization code to fully enable active security. Having done this groundwork, this project can now also be used to kick-start other, actively secure, MPC projects, since these steps are use case independent and can be reused.

In the remaining chapter there will be a quick overview of the main components of FRESCO.

Project Structure

FRESCO is available as a maven project and as a docker image. Including the framework as a maven dependency, as in our case, only takes several lines of code. The framework is separated into two main parts, the core project and the protocol suites, which are both separate maven projects. This design allows for additional protocol suites to be developed.

The FRESCO core

FRESCO core provides the general functionality needed to run an MPC program. The main entry point to the library is the `SecureComputationEngine`, this class is the main driver of the computation. Here all the MPC protocols are stored and evaluated over the network. This class heavily relies on the `ResourcePool`, which serves as a container for runtime variables and objects. Specific protocol suites need to implement their version of the `ResourcePool` so they have all the state variables they need. Finally there is the Protocol suites interface which has to be implemented externally.

Protocol Suites

There are many known techniques to implement MPC. These techniques are generally known as secure computation protocols. Since there are often a lot of these protocols required for a specific strategy, FRESCO implements all sub protocols which are related, together in so called protocol suites. These protocol suites contain the code to enable the specific cryptographic technique used to ensure secure multi party computation.

The IFX project

This project implements the use case described in the introduction, using the both the FRESCO core and SPDZ protocol suite. The project consists of two main entry points, which will from now on be referred to as Host and Client (having no network implications). Both of these can be executed by the main function in the priceFinder, depending on startup parameters. We will elaborate on this design desicion later in this chapter.

Initialization

Upon start, the user can give some customizing input, determining some computation strategies for the application. In the current state the input necessary is given in the Makefile and a Command Line Parser class takes care of it, inside of the project. The data structure created by this Parser class can easily be generated from different sources, e.g. configuration files from databases. Next to configuration, the input of the program also consists of a price, a volume and a date for each client and the host.

- **Price** The price either indicates the amount of money a client is willing to pay per each unit, or in case of the host, it indicates the minimum amount of money expected per unit.
- **Volume** The amount that is either requested or provided by a given date.
- **Date** in a later enhancement, several selling windows with different delivery date options are possible. The date either indicates the date of order for the client or indicates the date from which on the units are available.

After creating the required data structure, the initialization begins. This initialization has essentially two main parts. Firstly, the components necessary for the application (i.e., this specific use case) need to be set. This is done by setting date, price, volume and also starting up the network and sharing some information with the other participants. Secondly, there are some things to be done to start the framework. Generally, the framework can be started rather quickly, all the needed code can also be found in their demonstrator project. Still these Demos are all not actively secure. They do not use the secure SPDZ protocol, but instead have an insecure method of generating the multiplication triples. To fix this issue we need to include the MASCOT preprocessing strategy which requires some boilerplate code for its initialization. After having initialized both the application as well as the framework, we can start our application by running it in the `SecureComputationEngine`.

The PriceFinder

To have a cleaner interface for our demonstrator project, we wanted to be able to start the project as a host or as a client. This led to the executable class of the price Finder. Following up on discussions with our partner, we wanted to be able to quickly edit the logic of the price agreement. In this class we define the price finding method and pass it to our framework. This way, we aim to make this method easily accessible without having to dig into our framework code.

The Process Flow

When running our application, we pass the instance to the `SecureComputationEngine` in the `runApplication` function, alongside our previously instantiated protocol suite and `ResourcePool` objects. Every class implementing the application interface, needs to have a `buildComputation` function. This function is called by the framework and used to build the MPC protocol. Building the protocol is achieved by returning an arbitrary number of lambda functions, which will be executed consecutively later on. In other words, in `buildComputation` the code of our application is submitted to the framework. This is done, because the real computation has to be done over the network, and the function calls to the framework have to be replaced by native protocols.

The logic for this specific application can mostly be found in the `ATPManager`, it also contains the `ATPUnit` subclass. A single `ATPUnit` contains all the info about one order. The stocks provided by the host are also stored as single units per date. This abstraction allows for quick sorting of secretly shared values and to simplify operations and storage.

Secure Channel

After implementing a specific use case for our parnering company, we wanted to extract as much functionality as possible into a framework. This framework can be used to quickly prototype FRESCO projects. On the one hand, we already described that a lot of boilerplate code is necessary to enable the security of spdz. On the other hand, we provide some network capabilities based on the Java Cryptographic extension. It is therefor possible to have a TCP connection with the other parties in the protocol. The Sender and Receiver classes can be handed to the internal structure, such that FRESCO only uses this authenticated and encrypted communication channel. As it best fit our use case, at the moment the certificates are distributed beforehand.

Running the Demo

We tested our protocol using different settings. Therefore the project comes with several predefined demonstrator setups. In all of these there is one party as the host and several parties as clients started in parallel. All of these processes work in different sub directories on the same machine. The Makefile, which provides these demonstration setups, also states the inputs for all the clients. Only the host process reads its input from a json data file. In the end one can see the output of the individual processes in the log file of their respective directory.