Embedding MPC into C++

Nikolaos Kofinas

1 MOTIVATION

Various modern applications use Multiparty Computations to join the data from various parties/users in order to extract a result over the total data. These computation are trivial but now-days most of the users request a way to do them while they are keeping their data private without passing them to other parties or a trusted authority. The theory for Secure Multiparty Computations exists but there is no easy way for programmers to implement them in a popular language like `C`, `C++`, `Java`, `Python`, etc. Resent works tried to create new languages that let the programmer to write code which have the capability to handle both local and share computations. Unfortunately, these languages are independent and their is no easy way to embed them in a popular language.

2 PROPOSAL

In this project, we would like to embed Secure Multiparty Computations inside the `C++` programming language. We chose `C++` because it is a very popular languages and it used to write all sort of applications. Thus, if we embed MPC inside `C++` will be possible to create any kind of applications which is impossible with the other, independent languages. Some examples of such applications are: a poker application with GUI which can be used from real users, an application that requires some dynamic input from the user, etc.

The first stage of this project is to formalize the problem and see what exactly we need to embed into `C++` During this step we need to find out what functionality do we need from our DSL. The current idea is that the DSL will be used only to write a function which will contain all the multiparty computations and it will be callable from anywhere inside the `C++` code. Also, we need to find out what operations we need to include in our DSL language (in order to do this one we must find out what it can be translated into boolean functions).

The second part will be to find out how to implement the Domain Specific Language inside `C++`. Currently, we do not know if we must use the boost.proto and/or boost.spirit libraries which let the user create DSLs inside `C++` or we can use the meta-programming functionality of `C++` to describe completely the DSL.

The third stage will be to write down the actual implement of the DSL into `C++`. At first we must have an implementation that will simulate MPC without using the translation to boolean circuits (our DSL will just do multiparty computations). In this stage we need to find out how to exactly translate the DSL into `C++` understanding code and how the communication of different parties will happen. The second milestone will be to add the translation to boolean circuits inside our implementation and validate that everything run correctly.