Embedding SMC 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 computations are trivial but now most of the users request a way to do them while they are keeping their data private without passing them to other parties. One solution is to use a neutral party called "Trusted Authority" but their is no guarantee that this neutral party will keep the data hidden. The theory for Secure Multiparty Computations exists but there is no easy way for programmers to implement a program that uses SMC 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 do not have the capabilities that a modern language has (e.g. GUI libraries, IO libraries, etc).

2 RELATED WORK

Fairplay: Fairplay [1] project created a Domain Specific Language which can be compiled directly into boolean circuits. Then, these circuits are used in order to run the share code in a Secret Multiparty Computations engine. The syntax of Fairplay is similar to Pascal and thus, it is not familiar for most of the programmers. Additionally, Fairplay allow the user to write only share code and thus the local part of the code must be written in a different (host) language.
PCF: The Portable Circuit Format [2] project presented a language which syntax is similar to ansi-C and thus most of the programmers can use it. While the idea of this project to create a DSL which syntax is very close to an actual language, is has two disadvantages. First of all, it is designed only to work with two parties and thus it cannot be scaled to n > 2 parties and secondly it has the same problem as Fairplay, the programmers needs a host language in order to write the code that handle the local variables.
Wysteria: Wysteria [3] is another DSL which can provide to the programmer the power to write local and shared computations inside the same code. This is a significant advantage compared to the previous two approaches because the programmer does not need a host language. On the other hand there are two disadvantages, firstly the syntax of this language is very similar to oCaml and thus it can be strange for some programmers and secondly, it provides to programmers only basic programming capabilities.

3 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 language and it is used to write all sort of applications. Thus, if we embed MPC inside `C++` then it will be possible to create any kind of applications which is impossible with any independent language. 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 want from our DSL. The current idea is that the DSL will be used only to write functions which will contain only secret multiparty computations and the these function will be called from anywhere inside the `C++` code. Also, we need to find out what types, operators, and statements we need to include in our DSL. Finally, we must specify the syntax of the DSL which we want to be as similar to the syntax of `C++` as possible.
The second part will be to implement the interpreter for our language inside `C++`. Currently, the idea is to use the `C++` meta-programming language together with the Boost MPL library which provides some compile-time parsing functions. The implementation of our interpreter is the most difficult part of the project and thus our first implementation will not contain the conversion from DSL code to boolean circuits because we want to make the problem simpler.
The third part will be to add the translation to boolean circuits inside our interpreter. Additionally, the code that will handle the connection with the GMW [4] server must be created in compile time. The GWM is a standalone program which can handle all the communication and the execution of the boolean circuit between all the parties.

4 LANGUAGE

To specify our language first we must write down the types, statements and operations that will have:

- **Types**: bool, int (16 or 32), arrays of int or bool
- **Operators**: <, >, ==, !=, !, +, -, =, &, |, &&, ||, ∧
- **Statements**: if then else

Types: As we can see, our DSL has only two types bool and int. The reason behind this restriction is that the translation to boolean circuits is very difficult for float and double types. Also, if we allow strings as part of our language then their length must be the same for all parties and, at the time, it is simpler to let this type outside of our language.
Operators: Our language support almost all the basic arithmetic and logic operations with the exception of the '*' and '/' arithmetic operators. The reason that our language will not have these operators is because it is difficult to create the appropriate boolean circuit for them.

Statements: The DSL will have only one statement, the if statement. This statements is easy to be translated into boolean circuits compared to the for, while, switch, and do-while statements. We will also need a for statement in order to access all the elements of a matrix. This for must be able to be unfold during compile time. Thus the initialization, condition, and increase part of the loop must be known during compile time in all parties (e.g. they would not contain any secret variable).

5 Project Milestones

In order to complete this project in the two month time window we decided to split it in small pieces and complete one at the time:

1. Milestone 1 (Deadline 30 Oct):
    - Finalize the types, operations and statements of our language
    - Create the appropriate syntax for the language
2. Milestone 2 (Deadline 30 Nov):
    - Design the interpreter of our DSL without the boolean conversion
3. Milestone 3 (Deadline 10 Dec):
    - Finalize interpreter by creating the boolean conversion part

As we can see we allocated one month for the second milestone because it is the most difficult. During the second milestone, we must found how to implement the DSL inside the C++ and also how to evaluate the DSL during compile time. The milestone 3 can be done in a sorter amount of time if the interpreter is functional. The hard deadline for the project is 15th of December and thus we have 5 additional days to allocate if a milestone stalls.

References

[1] Ben-David, Assaf, Noam Nisan, and Benny Pinkas. “FairplayMP: a system for secure multi-party computation.” Proceedings of the 15th ACM conference on Computer and communications security. ACM, 2008.

[2] B. Kreuter, ahbi shelat, B. Mood, and K. Butler, “PCF: A portable circuit format for scalable two-party secure computation.” in USENIX, 2013

[3] Rastogi, Aseem, Matthew A. Hammer, and Michael Hicks. “Wysteria: A programming language for generic, mixed-mode multiparty computations.” (2014).

[4] S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, and D. Rubenstein, âĂIJSecure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces,âĂİ 2011, http://eprint.iacr.org/