

Embedding Secure Multiparty Computations into C++



Nikolaos Kofinas

Department of Computer Engineering
University of Maryland, College Park

December 12, 2014

Presentation Outline

- 1 Problem
- 2 C++ Template Meta-programming
- 3 Our approach
- 4 Conclusion



Presentation Outline

- 1 Problem
- 2 C++ Template Meta-programming
- 3 Our approach
- 4 Conclusion

What is Secure Multiparty Computations?

- What it is?

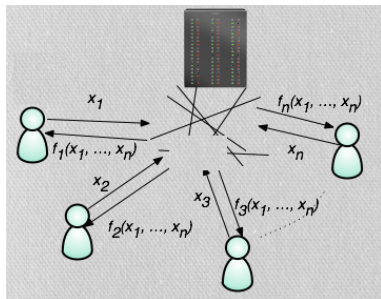


Figure: Source: <http://www.cs.columbia.edu/~mariana/noncollude.htm>

- Why is useful?

Related work

- Various variables for executing the programs
- Languages designed for these kind of problems
 - Wysteria
 - Fairplay
 - Ansi C

Presentation Outline

- 1 Problem
- 2 C++ Template Meta-programming**
- 3 Our approach
- 4 Conclusion

C++ Templates

- Generalize classes and functions

```
template <typename T>
bool greater(T a, T b) {
    return a > b ? true : false;
}
```

Templates and meta-programming

- Turing complete
- Discovered by accident

```
template <int n>
struct factorial {
enum { value = n * factorial<n - 1>::value };
};

template <>
struct factorial<0> {
enum { value = 1 };
};
```



Basics of meta-programming

- Variables

```
const int x = 2;  
enum { y = 14 };
```

- Functions

- Definition

```
template <typename x, int y>  
struct doSomething{};
```

- Implementation and specializations

```
template <int y>  
struct doSomething<int>{  
    static const int result = y*2;  
};  
template <int y>  
struct doSomething<float>{  
    static const float result = y*202;  
};
```

Presentation Outline

- 1 Problem
- 2 C++ Template Meta-programming
- 3 Our approach**
- 4 Conclusion



What was the plan?

- What we needed?
 - Needed compile time verification of the user code
- How can we do that?
 - Create a Domain Specific Language
- What about the boolean translation?
 - Can be handled during run time

Specification of the DSL

- Variables
 - SMCvalue [bool or int]
 - sharedSMCvalue [bool or int]
 - forSMCvalue [bool or int]
 - idSMCvalue [used only to store ids]
- Statements
 - Plus, Minus [in: int, out: int]
 - Greater, Lesser, And, Or...
 - Set
 - Sequence
 - If then else
 - For
 - Return

Syntax of the DSL

- Based on the languages that we did in class
- It is a bit awkward for someone without template experience

```
typedef SMCvalue<int, 1> s1;  
typedef SMCvalue<int, 2> s2;  
If<Greater<s1,s2>,Ret<Ip<s2> >, Ret<Id<s2> > >
```

```
typedef SMCvalue<int, 1> s1;  
typedef SMCvalue<int, 2> s2;  
typedef sharedSMCvalue<int, 1> sh1;  
Seq<  
    Set<sh1,s1>,  
    If<  
        Greater<sh1,s2>,  
        Ret<Id<s1> >,  
        Ret<Id<s2> >  
    >  
>
```



A complete Example

```
typedef mySMCvalue<int, 1> s1;  
typedef SMCvalue<int, 2> s2;  
typedef sharedSMCvalue<int, 1> sh1;  
s1 v1;  
s2 v2;  
v1.value = 2;  
v1.ip = myip;  
v2.ip = hisip;  
typedef Seq<  
    Set<sh1, s1>,  
    If<  
        Greater<sh1, s2>,  
        Ret<Id<s1> >,  
        Ret<Id<s2> >  
    >  
> func;  
int idOfWinner = wrapper<func>(v1, v2);
```



Implementation

- We need an Environment handler

```
template <typename Name, int Value, typename Env>
struct Binding {
    ...
} ;

typedef Binding<int, 2,
               Binding<float, 3, EmptyEnv>> Env;

template<typename Name, int Value, typename Env>
struct EnvLookup <Name, Binding<Name, Value, Env>>
{ static const int res = Value; };

template<typename Name, typename Name2,
         int Value2, typename Env>
struct EnvLookup<Name, Binding<Name2, Value2, Env>>{
    static const int res=EnvLookup<Name, Env>::res;
} ;
```



Implementation

- We followed the left side first eval approach

```
template <typename Exp, typename Env, bool
        IsReturnLegal>
struct Eval {} ;

template <typename Expr1, typename Expr2>
struct Plus {} ;

template <typename Expr1, typename Expr2,
        typename Env, bool IsReturnLegal>
struct Eval<Plus<Expr1,Expr2>, Env, IsReturnLegal>
{
    ...
};
```


Errors

- During run-time we are certain that the code is correct
- All Errors are generated during compile time!

```
SMC_lang.hpp:214:9: error: static assertion failed:  
    After Id there must be an SMVvalue!  
        static_assert(!std::is_same<Expr1, Expr1>::  
            value, "After Id there must be an  
                SMVvalue!");
```

Real Error

```

In file included from main.cpp:7:0:
SMC_lang.hpp: In instantiation of 'static constexpr decltype(auto) Eval<Ret<Id<
    Expr1> >, Env, true>::result() [with Expr1 = Greater<SMCvalue<int, 1>,
    SMCvalue<int, 1> >; Env = Binding<SMCvalue<int, 1>, 1, Binding<SMCvalue<int
    , 2>, 2, EmptyEnv> >]':
SMC_lang.hpp:135:48:   required from 'static constexpr decltype(auto) Eval<If<
    Cond, Then, Else>, Env, IsReturnLegal>::result() [with Cond = Greater<
    SMCvalue<int, 1>, SMCvalue<int, 2> >; Then = Ret<Id<Greater<SMCvalue<int,
    1>, SMCvalue<int, 1> > > >; Else = Ret<Id<SMCvalue<int, 2> > >; Env =
    Binding<SMCvalue<int, 1>, 1, Binding<SMCvalue<int, 2>, 2, EmptyEnv> >; bool
    IsReturnLegal = true]':
SMC_abstract.hpp:146:43:   required from 'std::string wrapper(Args ...) [with
    Expr = If<Greater<SMCvalue<int, 1>, SMCvalue<int, 2> >, Ret<Id<Greater<
    SMCvalue<int, 1>, SMCvalue<int, 1> > > >, Ret<Id<SMCvalue<int, 2> > > >;
    Args = {SMCvalue<int, 1>, SMCvalue<int, 2>}; std::string = std::
    basic_string<char>]':
main.cpp:44:28:   required from here
SMC_lang.hpp:214:9: error: static assertion failed: After Id there must be an
    SMVvalue!
    static_assert(!std::is_same<Expr1, Expr1>::value, "After Id there must
        be an SMVvalue!");

```



Presentation Outline

- 1 Problem
- 2 C++ Template Meta-programming
- 3 Our approach
- 4 Conclusion**



Conclusion





- C++ meta-programming is very strange and difficult to learn
- You can use meta-programming for anything
- Maybe not the best approach
- Future work
 - Add the boolean translation
 - Find a way to create better errors



Questions

Questions?



-  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.
-  B. Kreuter, ahbi shelat, B. Mood, and K. Butler, “PCF: A
portable circuit format for scalable two-party secure
computation.” in USENIX, 2013
-  Rastogi, Aseem, Matthew A. Hammer, and Michael Hicks.
“Wysteria: A programming language for generic, mixed-mode
multiparty computations.” (2014).
-  S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, and D.
Rubenstein, Secure multi-party computation of boolean
circuits with applications to privacy in on-line marketplaces,
2011, <http://eprint.iacr.org/>