1 Summary

This is a guest lecture about knowledge inference for optimizing secure multiparty computation[1].

Given an Secure Multiparty Computation program $S$, we want to know for each party $P$, which program variables they “know”.

Motivation: Given this knowledge, we can optimize Secure Multiparty computation to be more efficient.

- Knowledge Inference Algorithm: For each Party, it outputs which variable they know.
- constructive knowledge Inference Algorithm: For each Party, it outputs which variable they know, and a program that can generate it using party’s input and output.

2 Example: 2 party 2 input median

We assume that party A and B owns $x_1, x_2, y_1, y_2$ respectively, s.t. $x_1 < x_2, y_1 < y_2$. The binary-search-like code is as follows:

\[
\text{Algorithm 1: } \text{median}(x_1, x_2, y_1, y_2)
\]

1. bool $a = x_1 \leq y_1$
2. int $x_3 = a ? x_2 : x_1$
3. int $y_3 = a ? y_1 : y_2$
4. bool $d = x_3 \leq y_3$
5. int $m = d ? x_3 : y_3$
6. return $m$

Traditional way of doing Secure Computation will transform the whole piece of code into Garbled-Circuits or GMW, which is quite large. However, in this particular case we do not need to do secure computation for all code.

Claim 1 Given $x_1, x_2, m$, Alice can always infer values of $a$ and $d$ independent of $y_1, y_2$. Similarly, given $y_1, y_2, m$, Bob can always infer values of $a$ and $d$ independent of $x_1, x_2$.

This can be verified in the following tree:
Each run of the program will take one of the path of the tree. we can see that for bob, 
\( d = (m \neq y_1) \land (m \neq y_2), a = m \leq y_1 \)

3 In general

Let \( S \) be a program, \( y \) be a variable. Party \( A \) knows \( y \) if value of \( y \) only depends on its inputs and outputs. In other word, for any two program runs \( R_1, R_2 \), the coincidence on \( A \)'s input and output implies the coincidence of \( y \).

We say \( a \) is knows if:

\[
(x_1 = x'_1) \land (x_2 = x'_2) \land (m = m') \rightarrow (a = a')
\]

We feed

\[
\Phi_{pre} \land (\bigvee_i \Phi_i) \land (x_1 = x'_1) \land (x_2 = x'_2) \land (m = m') \rightarrow (a = a')
\]

into SMT to see if this is valid, where \( \Phi_{pre} \) are some pre-conditions, \( \Phi_i \) is one of the path condition, which is a set of predicates relating the program variables.

References