Introduction to Cryptography 01 May 2018

Lecture 24: Secure Computation III

Instructor: Vipul Goyal *Scribe: Brad Denby*

1 Introduction

This lecture focuses on general secure two-party computation for circuits of any polynomial size. Specifically, it considers Yao's garbled circuits. At the time of publication, this scheme was considered largely impractical due to its inefficiency. Today, however, the scheme is usable in practice for some circuits.

A key prerequisite to constructing Yao's garbled circuit scheme is the oblivious transfer primitive. Oblivious transfer was covered in a previous lecture. As a reminder, oblivious transfer is summarized as follows.

Oblivious Transfer Consider two parties S and R. Party S has two values s_0 and s_1 that are initially secret. Party R has a single bit b indicating which value s_b it would like to receive. After executing the oblivious transfer protocol, party R learns s_b but learns nothing about $s_{\overline{b}}$. Party S learns nothing about b.

Before presenting a formal construction of Yao's garbled circuit scheme, consider the following general overview. Let a sender S and a receiver R have inputs x_S and x_R, respectively. The parties would like to compute a public function f with these inputs, i.e. $f(x_S, x_R)$, without revealing the input to the other party.

To accomplish this secure computation, the sender S first converts the function f into an equivalent circuit C. The sender S then exercises a KeyGen algorithm from a symmetric encryption scheme in order to select two random keys for every wire in the circuit C. One of these keys is associated with a wire value of 0, and the other key is associated with a wire value of 1. The idea is that the receiver R learns exactly one of the two key values for each wire.

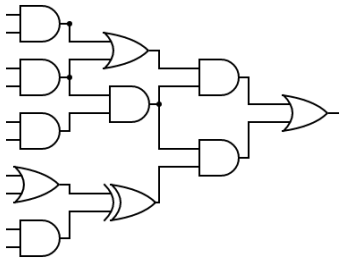


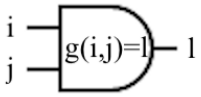
Figure 1: An example of a circuit C

At the time of computation, the receiver R has a key for each input wire to the circuit. Party R uses these inputs to compute the output at each gate in the circuit. These outputs are determined by garbled gate tables provided by the sender S. Eventually, R determines the garbled values for all output wires. These values are then decoded and the evaluation of the function $f(x_S, x_R)$ is revealed.

2 Construction

A more detailed construction is outlined as follows. Consider two parties S and R. These parties have agreed on a public function f that takes two inputs, x_S and x_R. An example of such a function is the millionaires' problem, in which two millionaires would like to learn who has more money without revealing their net worths. In this case, the inputs are their net worths and the output indicates the more wealthy party.

To begin the protocol, party S converts function f into an equivalent circuit C. Here, a circuit is considered to be a collection of gates connected by wires. A gate may be represented with a logic table which dictates the values of the output wires for every combination of input wire values. Let each wire be labeled with an index i, and require that each wire take on a value of either 0 or 1. An example (ungarbled) AND gate and corresponding logic table is pictured below.



Wire i	Wire j	Wire ℓ
0	0	0
0	1	0
1	0	0
1	1	1

After constructing the circuit C, party S selects two keys $k_{i,0}$, $k_{i,1}$ for each wire $i \in C$. Key $k_{i,0}$ corresponds to wire i with value 0, and key $k_{i,1}$ corresponds to wire i with value 1. For every gate $g \in C$, party S prepares a logic table having four encrypted entries. Specifically, these entries are $\mathsf{Enc}_{k_{i,0}}(\mathsf{Enc}_{k_{j,0}}(\cdot))$, $\mathsf{Enc}_{k_{i,0}}(\mathsf{Enc}_{k_{j,1}}(\cdot))$, $\mathsf{Enc}_{k_{i,1}}(\mathsf{Enc}_{k_{j,0}}(\cdot))$, and $\mathsf{Enc}_{k_{i,1}}(\mathsf{Enc}_{k_{j,1}}(\cdot))$ where i and j are the input wires into gate g.

Example 1 *Let gate g be an AND gate with input wires i and j and output wire ℓ. Now input wire i has keys $k_{i,0}$, $k_{i,1}$; input wire j has keys $k_{j,0}$, $k_{j,1}$; and output wire ℓ has keys $k_{\ell,0}$, $k_{\ell,1}$.*

The entries in the garbled logic table include the following values.

Wire i	Wire j	Encrypted wire ℓ values
$k_{i,1}$	$k_{j,0}$	$\mathsf{Enc}_{k_{i,1}}(\mathsf{Enc}_{k_{j,0}}(k_{\ell,0}\|\|0^m))$
$k_{i,0}$	$k_{j,0}$	$\mathsf{Enc}_{k_{i,0}}(\mathsf{Enc}_{k_{j,0}}(k_{\ell,0}\|\|0^m))$
$k_{i,1}$	$k_{j,1}$	$\mathsf{Enc}_{k_{i,1}}(\mathsf{Enc}_{k_{j,1}}(k_{\ell,1}\|\|0^m))$
$k_{i,0}$	$k_{j,1}$	$\mathsf{Enc}_{k_{i,0}}(\mathsf{Enc}_{k_{j,1}}(k_{\ell,0}\|\|0^m))$

Under this construction, party R is able to correctly decrypt exactly one of the table entries. The decrypted entry provides R with the garbled wire value for the gate output. Specifically, when party S has keys k_{i,b_i} and k_{j,b_j} then value $k_{\ell,g(b_i,b_j)}$ may be recovered. Here, $g(b_i, b_j)$ represents the gate g output with input bits b_i and b_j.

Of course, if the garbled tables generated by party S are in a predictable order, then party R can deduce the ungarbled input values based on which table entry it was able to successfully decrypt. In practice, the garbled table entries are permuted randomly. Party R attempts to decrypt all table entries, and the entry that successfully decrypts is used as the garbled value for the output wire.

Example 2 *Let $k_{\ell,0}$ and $k_{\ell,1}$ be the garbled outputs for gate g. Before encryption, each key is appended with m zeros, i.e. $k_{\ell,0}||0^m$ and $k_{\ell,1}||0^m$. When evaluating a gate, party R attempts to decrypt all entries of the logic table. The entry that decrypts with m zeros in the least significant bits is taken to be the output value.*

Finally, party S prepares tables that decode the output wires of the circuit. For an output wire w, party S provides pairs $k_{w,0} : 0$ and $k_{w,1} : 1$. After party R evaluates all wires in the circuit, it collects the values of the output keys and converts them into ungarbled output values.

3 A Secure Two-Party Computation Protocol

Given the previously described construction, a secure two-party computation protocol may be established. The first version of this protocol involves two parties S and R who are assumed to be semi-honest, i.e. honest-but-curious. Both parties know the function f and the expected corresponding circuit C. Let party S have secret input x_S, and let party R have secret input x_R. The goal of these parties is to compute $f(x_S, x_R)$ without revealing additional information about the inputs to the other parties.

To accomplish this task, party S constructs the garbled circuit (i.e. obfuscated logic gates) and party R computes with these obfuscated values. Thus, party R must somehow obtain the keys for all input wires of circuit C. The input keys corresponding the secret value x_S of party S may be sent to party R directly. In order for party R to obtain the keys for its input x_R, the oblivious transfer primitive must be exercised.

This portion of the protocol may be summarized with the following steps.

1. $S \rightarrow R$: Party S sends $|x_S|$ keys to R, where $|x_S|$ represents the number of bits in x_S. Each key represents a single bit of x_S, which is input into the circuit over a wire.

2. $R \leftrightarrow S$: Parties S and R execute a 1-out-of-2 oblivious transfer interactive protocol for each bit of x_R. Specifically, for each bit of x_R party R receives the corresponding key $k_{i,b}$. Party S does not know the value of the bit b, and party R cannot determined the value of the other key $k_{i,\bar{b}}$.

3. $S \rightarrow R$: Party S sends the garbled gate tables along with the output decoding tables to party R, i.e. party S sends the garbled circuit C.

After executing this portion of the protocol, party R has one key for each input wire in addition to the gate tables. As a result, party R can determine the output for each gate. Once the outputs for the final gates have been determined, party R can utilize the output decoding tables in order to find the function result.

4 Security Questions

At a high level, why should this scheme be secure for honest-but-curious adversaries? First, note the following fact:

- Given key $k_{i,b}$, party R cannot determined the corresponding bit value b (unless, of course, i is an input wire for a bit of x_R).

This fact informs the notion of security for party S. Since party R only ever learns one key for the input wires of x_S, then party R cannot determine the bits of x_S. The notion of security for party R follows directly from the receiver security of oblivious transfer.

Also note that security is defined in terms of how much *more* a party may learn due to the scheme. If the circuit is simply the XOR of the two party's inputs, then a trivially secure scheme is defined to be one in which party S sends x_S to party R. No matter the scheme, party R is able to recover x_S.

What happens to the security of this scheme when the parties are more malicious than under the honest-but-curious adversary model? If party S is malicious, it could construct a circuit C that does not properly reflect the reference function f. Are there ways to modify the protocol in order to avoid this vulnerability?

Multiple solutions to this problem have been considered. Some of these solutions are presented as follows.

- One solution is to convert the circuit C into a graph three-coloring problem and perform a zero knowledge proof that the circuit has been constructed properly. However, in practice this approach is not efficient (despite being polynomial time). Thus, this solution is not currently practical.
- In practice, this attack is mitigated by requiring party S to prepare n independently constructed garbled circuits. All of these garbled circuits are sent to party R, and party R selects one of the n choices. After party R selects one of the garbled circuits, party S reveals the remaining garbled circuits. Party R verifies that the revealed circuits were properly constructed. Thus, the chosen circuit is incorrect with probability no greater than $1/n$.
- An alternative to this approach is for half, i.e. $n/2$, of the garbled circuits to be revealed. Party S executes the remaining garbled circuits in parallel. If all of the evaluated circuits give the same result, then party R can be confident that party S did not cheat when constructing the circuits. Specifically, it is unlikely that all of the unopened circuits were incorrect and all of the opened circuits were correct.

Many variations of these techniques exist. In the above schemes, party R terminates when an invalid circuit construction is detected. However, terminating execution could reveal information about x_R. As a result, it can be more secure to avoid early termination even if an incorrectly constructed circuit is detected. Other variations, such as accepting the majority vote when evaluating $n/2$ circuits, have also been proposed.

5 Secret Sharing Schemes

Broadly speaking, secret sharing schemes are used to split a secret s into n shares $s_1, s_2, \ldots, s_n$. In general, a certain number of these shares are required in order to reconstruct the secret s. Various kinds of secret sharing schemes exist.

n-out-of-n secret sharing scheme properties

- Given any $n-1$ or fewer shares, a party can learn no additional information about the original secret s.
- Given all n shares $s_1, s_2, \ldots, s_n$, a party can fully recover the original secret s.

Shamir Secret Sharing: a t-out-of-n secret sharing scheme

- Given any $t-1$ or fewer shares, a party can learn no additional information about the original secret s.
- Given any t out of n shares, a party can fully recover the original secret s.

Construction 1 (Shamir secret sharing scheme) *Let the secret s be an element of a finite field $\mathbb{F}$. Let the total number of shares be n, and let the threshold number of shares be t. In order to generate shares $s_1, s_2, \ldots, s_n$, select $t-1$ elements from field $\mathbb{F}$ uniformly at random. Use these elements to construct a polynomial of the form $p(x) = s + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$, where the coefficients are the selected elements. Share s_1 is defined to be $p(1)$, share s_2 is defined to be $p(2)$, and so on. Formally, share $s_i = (i, p(i))$.*

The two properties of a secret sharing scheme may be verified as follows. First, consider the first property. Given any $t-1$ or fewer shares, a party cannot recover secret s. To see why, note that with $t-1$ shares an adversary has $t-1$ points of a degree $t-1$ polynomial. For every additional point, there exists a unique, valid polynomial through all t points. Thus, an adversary cannot recover the secret s with less than t points.

Next, consider the second property. Given t points, there exists a unique polynomial of degree $t-1$ through all t points. Such a polynomial may be determined through various schemes, such as Lagrange interpolation. Once the polynomial has been constructed from the points, its evaluation at zero gives the original secret s.

Secret sharing schemes serve as the building blocks for threshold cryptography. Threshold cryptography consists of a broad collection of schemes that require some threshold of participants in order to recover a secret. These schemes can become quite intricate. For example, there are applications in which a group consists of a core committee and a general committee. It may be desireable for any single member of the core committee to recover the secret, or a quorum of members from the general committee. Different members may be given different weights, which affect their impact on the threshold for secret recovery. A general approach to this problem defines a circuit with n inputs, one for each member of the group. If the member is present, then the corresponding input wire is 1; otherwise, the corresponding input wire is 0. The output of the circuit is 0 or 1, indicating whether the threshold for secret recovering has been acheived. This problem statement connects secret sharing schemes to Yao's garbled circuits.