Secure Multi-Party Computation (Cont.)

Notes by Yael Kalai

MIT - 6.5610
Lecture 13 (March 17, 2025)

> ***Warning:*** This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Outline

- The BGW construction in the semi-honest setting (secure against $t < n/2$ corruptions)
- The BGW construction in the malicious setting (secure against $t < n/3$ corruptions)

The BGW Construction in the semi-honest setting

Last class we started to present the BGW multi-party computation (MPC) construction due to Ben-Or, Goldwasser and Wigderson [1], in the honest-but-curious. In this setting the adversary controls $t < n/2$ parties, but does not modify the behaviour of the parties it controls. Rather it only observes their internal states and tries to learn information about the secret inputs of the honest parties. We note that their protocol is information theoretically secure! It does not rely on any cryptography, and is secure even if the adversary is all powerful.

We note that if we rely on cryptography we can get security against any number of corruptions!

Construction. The BGW protocol consists of three phases:

1. Secret sharing of inputs phase
2. Computation on shares phase
3. Reconstruction phase

We elaborate on each of these phases.

Phase 1: Secret-sharing of the inputs. Fix a prime $p > n$. Each party secret-shares her input among the n parties using the $(t+1)$-out-of-n Shamir secret sharing scheme over $\mathsf{GF}[p]$. Formally, each party P_i, who wants to share a bit $b \in \{0,1\}$ does the following:

Recall that an adversary that controls at most t parties learns nothing (information theoretically).

1. Select uniformly at random t coefficients $a_1, \ldots, a_t$ from $\mathsf{GF}[p]$, and let $g_b(x) = b + a_1 x \cdots + a_t x^t$.

2. Send $g_b(j)$ to party P_j, for all $j \in [n]$.

Phase 2: Computation on shares. The parties jointly compute the function f "on their shares," as follows. Think of f as being represented as an arithmetic circuit with addition gates and multiplication gates modulo p.

The parties jointly compute each gate in a way that satisfies the following invariant: If the parties start with random shares of the input wires to the gate they end up with random shares of the output wire to the gate. Suppose the input wires have values a and b and were shared via random degree t polynomials g_a and g_b, respectively, where $g_a(0) = a$ and $g_b(0) = b$, and party P_i holds the shares $g_a(i)$ and $g_b(i)$.

As we mentioned last class addition and multiplication modulo p is complete. This is the case since we know that addition and multiplication modulo 2 is complete, and for any bits $a, b \in \{0,1\}$, it holds that $a \cdot b \mod 2 = a \cdot b \mod p$ and $a + b \mod 2 = a + b - 2ab \mod p$.

By random shares we mean shares that were chosen via a *random* degree t polynomial that respects the value of the gate.

Addition gates modulo p: If the gate is an addition gate then the value of the output wire is $a + b$. Note that $g_{a+b} := g_a + g_b$ is a random degree t polynomial that satisfies

$$g_{a+b}(0) = g_a(0) + g_b(0) = a + b.$$

Each party P_i can compute the i'th share of g_{a+b} locally given the shares $g_a(i)$ and $g_b(i)$ by

$$g_{a+b}(i) = g_a(i) + g_b(i)$$

Note that no interaction is required here! The parties can do this computation locally.

multiplication by a scalar modulo p: If the gate is a multiplication by a non-zero scalar $c \in \mathsf{GF}[p]$ of the wire with value a, shared via the random degree t polynomial g_a, then the value of the output wire is $c \cdot a$. Note that $g_{c \cdot a} = c \cdot g_a$ is a random degree t polynomial such that

$$g_{c \cdot a}(0) = c \cdot g_a(0) = c \cdot a.$$

Therefore, each party P_i can compute the i'th share of this polynomial locally by

$$g_{c \cdot a}(i) = c \cdot g_a(i)$$

We defer the computation of multiplication gates for later. For now, simply assume there is a way to securely compute the shares of multiplication gates as well, in a way that satisfies the invariant.

Reconstruction phase: Once the parties hold random shares of the output gate, the parties simply send their share of the output gate to all the parties that should receive the output. Each parties locally reconstruct the value of the output gate using the reconstruction algorithm of Shamir's secret-sharing scheme.

In some protocols we want all the parties to receive the output, whereas in others we may want only one party (or a few parties) to receive the output. This will come up when we compute the shares of a multiplication gate.

Security

We next give an intuitive argument why this protocol is secure, assuming there is a way to securely compute random shares of multiplication gates from random shares of the input gates. Intuitively, security follows from the fact that the adversary only sees t shares for each gate, which we argued that are always random shares (i.e., shares of a random degree t polynomial that respects the value of the gate). Thus, security follows from the security of Shamir's secret sharing scheme.

Remark: So far we have seen how the parties can compute *linear* functions securely! It remains to show how to securely compute shares of multiplication gates. For this we will use the fact that we know how to compute linear functions securely.

Multiplication gates: Suppose the gate g is a multiplication gate with input gates with values a and b that are shared via random polynomials g_a and g_b, respectively.

It is tempting to set the secret sharing polynomial corresponding to this gate to be the polynomial $g_{a\cdot b} = g_a \cdot g_b$. Note that indeed

$$g_{a\cdot b}(0) := g_a(0) \cdot g_b(0) = a \cdot b,$$

as desired, and each party P_i can locally compute the shares of this polynomial by setting

$$g_{a\cdot b}(i) = g_a(i) \cdot g_b(i).$$

The problem is that $g_{a\cdot b}$ is of degree $2t$. So, now we will need $2t+1$ parties to reconstruct, and this number will grow with every multiplication gate. So this seems like a bad idea. Moreover, this polynomial is not random. Nevertheless, this serves as a first step. After this step the parties will carry out two additional steps, to fix the two problems mentioned above: a *degree reduction* step and a *rerandomization* step.

Degree reduction In the degree reduction step each party converts its share of $g(x) = h_0 + h_1 \cdot x + \ldots + h_{2t}x^{2t}$ to a share of the truncated degree t polynomial, denoted by $\hat{g}(x) = h_0 + h_1 \cdot x + \ldots + h_t x^t$. To this end we use the following theorem.

Theorem 1. *There exists a constant matrix $A \in \mathsf{GF}[p]^{n\times n}$ such that for every degree $2t$ polynomial g, denoting by $\hat{g}$ the corresponding degree t truncated polynomial, it holds that:*

$$(\hat{g}(1), \ldots, \hat{g}(n))^T = A \cdot (g(1), \ldots, g(n))^T$$

We will later to see what this matrix A is. But before that, note that assuming we know what A is, it seems like to compute the shares of the multiplication gate we simply need to compute this linear function securely! Recall, this is done in three phases:

1. **Secret sharing phase:** Each party P_i secret shares her input $g(i)$ using Shamir's $(t+1)$-out-of-n secret sharing scheme.
2. **Computation phase:** Each party locally applies the linear function A on all the shares.
3. **Reconstruction phase:** Finally, all parties send P_i the shares corresponding to $\hat{g}(i)$.

Indeed, if we compute each multiplication gate by running this three phase protocol, the parties will end up with shares of a a degree t polynomial corresponding to the output gate . However, this polynomial is not necessarily random, and this may be a problem for security! To remedy this, before the parties run this degree reduction step, they first rerandomize this degree $2t$ polynomial and then run the degree reduction, to ensure that they are truncating a random degree $2t$ polynomial.

Rerandomization: To rerandomize the degree $2t$ polynomial $g_{a\cdot b}$. Each party P_i secret shares the value 0 using a Shamir's $(2t+1)$-out-of-n secret sharing scheme. Namely, each party P_i chooses a random degree $2t$ polynomial g_i such that $g_i(0) = 0$ and sends $g_i(j)$ to each party P_j. Then each party P_i "rerandomizes" its share $g_{a\cdot b}(i)$ by adding to it the value $\sum_{j=1}^{n} g_j(i)$. Note that this is the i'th share of the polynomial $g_{a\cdot b} + g_1 + \ldots + g_n$, which is a random degree $2t$ polynomial that on 0 evaluates to $a \cdot b$. On this random polynomial the parties run the degree reduction.

The proof of Theorem 1. Denote by V the n-by-n Vandermonde matrix, where the i'th row is $(1, i, i^2, \ldots, i^{n-1})$. Let

$$g(x) = \sum_{i=0}^{2t} h_i x^i \quad \text{and} \quad \hat{g} = \sum_{i=1}^{t} h_i x^i$$

where g is a degree $2t$ polynomial, and $\hat{g}$ is its degree t truncated polynomial. Let

$$h = (h_0, h_1, \ldots, h_{2t}, 0, \ldots, 0) \in \mathsf{GF}[p]^n$$

be the coefficients of g and let

$$\hat{h} = (h_0, h_1, \ldots, h_t, 0, \ldots, 0) \in \mathsf{GF}[p]^n.$$

be the coefficients of $\hat{g}$. Note that

$$(g(1),\ldots,g(n)) = V \cdot h \quad \text{and} \quad (\hat{g}(1),\ldots,\hat{g}(n)) = V \cdot \hat{h}.$$

Also note that there is a linear projection function P that converts the vector h to the vector $\hat{h}$. Namely, consider the n-by-n matrix P such that for every $i, j \in [n]$, $P(i,j) = 1$ if and only if $i = j$ and they are both in $\{1,\ldots,t\}$, and otherwise $P(i,j) = 0$. Note that

$$P \cdot h = \hat{h}.$$

Combining all of the above, we have that

$$(\hat{g}(1),\ldots,\hat{g}(n))^T = V \cdot \hat{h} = V \cdot P \cdot h = V \cdot P \cdot V^{-1}(g(1),\ldots,g(n))$$

as desired. Thus, set

$$A = V \cdot P \cdot V^{-1}.$$

□

The Malicious Setting

We next consider the setting where the adversary is malicious and arbitrarily modifies the messages sent by the parties it controls. The first question that we should ask is whether the protocol described above is secure *as is*, against a malicious adversary. Recall that in the last lecture we mentioned that Shamir's secret-sharing scheme has the property that we can reconstruct the secret even if some of the parties send malicious shares. Specifically, we mentioned that Shamir's secret sharing scheme is the Read-Solomon code of the message $(f(0), f(1),\ldots,f(t))$, of length $k = t+1$, which we know can be decoded if at most $\frac{n-k}{2}$ of the coordinates are maliciously corrupted. Note that

$$\frac{n-k}{2} = \frac{n-t-1}{2}.$$

Thus as long as the number of corrupted parties, t, is at most $\frac{n-t-1}{2}$, which holds as long as $t < n/3$, we can reconstruct successfully!

This begs the following question: *is the BGW protocol secure assuming the malicious adversary controls less than $n/3$ of the parties?* The answer is yes, if the adversary behaves maliciously only in the reconstruction steps (either corresponding to the output gate or corresponding to a multiplication gate), and shares malicious shares on behalf of the corrupted parties. However, if the adversary behaves maliciously in the sharing phases, and sends shares that do not correspond to a degree t function (say, it simply sends random shares to all the parties), then the scheme is no longer secure! The issue is that

shares will no longer correspond to low-degree polynomials and as a result the scheme will lose its error correction property.

This is fixed by using a *verifiable secret sharing* scheme, where the dealer "proves" that it sent shares corresponding to a degree t polynomial (or a degree $2t$ in the case of a rerandomization step). This notion was introduced in [1], and they showed that if we replace each secret sharing phase with a verifiable one then the protocol is indeed secure against $t < n/3$ corruptions.

References

[1] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.