CSA E0 312: Secure Computation	9th November 2015
[Lecture 18]	
Instructor: Arpita Patra	*Submitted by: Marilyn George*

In the previous lectures we have been seeing the building blocks for an information-theoretically secure BGW protocol in the malicious adversarial model. We have seen Verifiable Secret Sharing (VSS) and the use of Reed-Solomon codes for distributed error correction during reconstruction. In this lecture, we will study the multiplication sub-protocol.

1 Recap: Multiplication in Semi-Honest Adversarial Model

Given the two secrets a and b that are (n,t) Shamir-shared among the n parties, we interactively compute the (n,t) Shamir-sharing of $c = a \cdot b$ as follows. We denote the shares of a and b given to i^{th} player P_i as a_i and b_i respectively:

1. P_i multiplies a_i and b_i to get a share c_i^{2t}, which is a share of $c = a \cdot b$ using a $2t$ degree polynomial $c(x)$ which is the product of the t-degree polynomials $a(x)$ and $b(x)$ which are used to share a and b respectively.

2. This share that P_i holds is a point on $c(x)$ with c as the constant term. P_i now (n,t) shares c_i^{2t} with the other players.

3. Now each player has (n,t) shares of $2t+1$ points on $c(x)$ (since honest majority holds) and they each compute a share of $c(0) = c$ as a linear combination of these shares using Lagrange's Interpolation. By the linearity of Shamir sharing, performing the linear operations on the shares of secrets is equivalent to sharing the output of the linear operations on the secrets. Additionally linear operations on (n,t) shared secrets output (n,t) shared secrets.

4. Hence at the end of this computation each player holds a share of c i.e. P_i holds c_i as required.

2 Issues in the Malicious Adversarial Model & Solutions

In the malicious world, there are several issues that arise in the above protocol. Again, we are given two (n,t) Shamir-shared secrets a and b (which are shared using VSS since this is the malicious world), and $3t+1$ parties, with at most t maliciously corrupted parties among them.

- Step 1 is a local computation and hence remains the same.

- Step 2 requires a player to share his c_i^{2t} value. Here, we use VSS to ensure that the shares he provides are consistent and correspond to the same secret. However, this secret could be some $\overline{c_i^{2t}} \neq c_i^{2t}$. We have to find a method to force players to share the correct product of a_i and b_i

- Step 3 is again a local computation and does not change.
- Provided all the above steps have executed correctly, at the end of this computation each player holds a share of c.

2.1 Forcing Adversaries to Share Correct Values

We note first that, we require only the participating adversaries who have not been eliminated to share correct values. A player can always refuse to send out any messages, and as a result, he is not considered in the computation any more.

Let us first recall some properties of VSS that we will use in our protocol:

- The VSS can be viewed as a two-level sharing i.e. each share received by player P_i is already shared among the other players. To elaborate, P_i receives the polynomials $F(x, i)$ and $F(i, y)$ as his shares. Consider $F(x, i)$ - since the other players P_1, P_2.. P_n have $F(1, y)$, $F(2, y)$.. $F(n, y)$, it follows that they can compute $F(1, i)$, $F(2, i)$.. $F(n, i)$, which are the shares of the polynomial $F(x, i)$. Hence this share of P_i is already shared among the n players. Similarly, every share in the VSS is double-shared.
- We can start VSS with a polynomial p as input instead of a secret. Then if this polynomial is a function of x, it is equivalent to setting $F(x, 0) = p(x)$. If it is a function of y, we set $F(0, y) = p(y)$.

Also as we saw in the previous lecture, given n points of a polynomial, Reed-Solomon codes can correct upto t errors, when $n = 3t + 1$.

As mentioned before, we are given players who already have the shares of a and b, shared using VSS. Then our protocol proceeds as per the following 2 sections:

2.1.1 Distributed Error Correction

First, the players broadcast their shares of a and b. These are points on the polynomials $a(x)$ and $b(x)$ used to share a and b. From the property of Reed-Solomon codes we can detect upto t errors, and identify the faulty players. We them leave them out of the computation in the following phases. Two points to be noted here are:

- The maximum number of faults that will occur is t, since only that many malicious players exist.
- The players broadcast shares of $a(x)$ and $b(x)$ and not the product polynomial, since the Reed-Solomon code with $3t + 1$ elements (players) can only correct for t errors in a t degree polynomial. The product polynomial is of degree $2t$. To correct t errors, we would need distance $> 2t$. For a distance of $2t + 1$, $d = n - (2t + 1) + 1 \Rightarrow$ We require $n = 4t + 1$ players.

2.1.2 Enforcing Correct Shares

This is a more involved procedure which assumes that the above step is concluded successfully i.e. all the players who were not eliminated shared their shares of a and b correctly. We now focus on the i^{th} player P who has shared his shares of a and b correctly. We shall abuse notation for this section and use a and b to denote a_i and b_i, the shares of a and b held by P. We then use $a_1, a_2, ..., a_n$ to denote the shares of a_i that P sent out to $P_1, P_2, ..., P_n$ in Step 2 of the protocol. Similar notation is used for the shares of b_i.

Let the polynomials used by P to share a and b be $A(x)$ and $B(x)$ respectively. The product polynomial $C(x) = A(x) \cdot B(x)$ has constant term $= c$, i.e. the required product of a and b (Note that a and b are shares of the underlying secrets). However, $C(x)$ is of degree $2t$ and non-random. We have to reduce the degree and randomize the polynomial. We do this by choosing t random polynomials $D_1(x), D_2(x), ..., D_t(x)$ such that the following polynomial is random, has degree at most t and constant term $c = a \cdot b$.

$$D(x) = C(x) - x \cdot D_1(x) - x^2 \cdot D_2(x) - ... - x^t \cdot D_t(x)$$

Then the polynomials are picked as follows:

$$\begin{aligned}
C(x) &= c + c_1 \cdot x + ... + c_{2t-1} \cdot x^{2t-1} + c_{2t} \cdot x^{2t} \\
\text{Then, } D_t(x) &= r_{t,1} + r_{t,2} \cdot x + ... + r_{t,t} \cdot x^{t-1} + c_{2t} x^t \\
x^t \cdot D_t(x) &= r_{t,1} \cdot x^t + r_{t,2} \cdot x^{t+1} + ... + r_{t,t} \cdot x^{2t-1} + c_{2t} x^{2t} \\
\text{Then, } D_{t-1}(x) &= r_{t-1,1} + r_{t-1,2} \cdot x + ... + r_{t-1,t} \cdot x^{t-1} + (c_{2t-1} - r_{t,t}) x^t \\
x^{t-1} \cdot D_{t-1}(x) &= r_{t-1,1} \cdot x^{t-1} + r_{t-1,2} \cdot x^t + ... + r_{t-1,t} \cdot x^{2t-2} + (c_{2t-1} - r_{t,t}) x^{2t-1}
\end{aligned}$$

We continue the choice of the coefficients similarly, as shown in the table below, where $r_{i,d}$ is the (randomly-chosen) coefficient of x^{d-1} in $D_i(x)$. Recall that we want all the terms of x with degree greater than t to cancel out in $D(x)$.

$C(x)$	c	c_1	..	c_{t-1}	c_t	c_{t+1}	..	c_{2t-2}	c_{2t-1}	c_{2t}
$x^t \cdot D_t(x)$			..		$r_{t,1}$	$r_{t,2}$	..	$r_{t,t-2}$	$r_{t,t-1}$	c_{2t}
$x^{t-1} \cdot D_{t-1}(x)$			..	$r_{t-1,1}$	$r_{t-1,2}$	$r_{t-1,3}$	..	$r_{t-1,t-1}$	$c_{2t-1} - r_{t,t-1}$	
...			..	...	..	..		..	..	
$x \cdot D_1(x)$		$r_{1,1}$		$r_{1,t-2}$	$r_{1,t-1}$	$c_{t+1} - r_{t,2} - ... - r_{2,t}$				

It is now easy to see that

$$D(x) = C(x) - x \cdot D_1(x) - x^2 \cdot D_2(x) - ... - x^t \cdot D_t(x)$$

will be a random polynomial of degree t, with constant term c. Then $D(x)$ is the ideal polynomial to share c.

After obtaining $D(x)$ by local computation, P uses VSS to share c, setting $F(x, 0) = D(x)$. This is equivalent to sharing with a polynomial as input, as we have seen in the properties of VSS. If P is an honest player, our protocol is complete and c is shared. However, if P is malicious, we have to ensure the correctness of $D(x)$ as follows:

- P also shares $D_1(x)$ among the other players as $d_{11}, d_{12}, ...d_{1n}$. Similarly he also shares $D_2(x), D_3(x), ..., D_t(x)$.
- Now, we consider another player say P_j having his (correctly-shared) shares of $a \to a_j$ and b_j. He locally computes $c_j = a_j \cdot b_j$, which is a point on the product polynomial $C(x) = A(x) \cdot B(x)$.
- Given the relation $D(x) = C(x) - x \cdot D_1(x) - x^2 \cdot D_2(x) - ... - x^t \cdot D_t(x)$, and his shares $d_j, d_{1j}, d_{2j}, ..d_{tj}$, he can verify if the relation holds at his public point α_j, taken as j for simplicity. Hence, he checks if $d_j = a_j \cdot b_j - j \cdot d_{1j} - j^2 \cdot d_{2j} - ... - j^t \cdot d_{tj}$. If it does not hold, P_j makes a public complaint.
- By the double-sharing property of VSS, this complaint can be verified by the other players. Since they hold shares of the shares used to compute the equation, they can check the complaint. If the complaint holds, then P is corrupt and is eliminated from the computation. Else, the complaint is ignored.

We conclude by saying that if all the honest parties accept the sharing without any valid complaints, then $D(x)$ does indeed share c. To see this clearly we look at the relation $D(x) = C(x) - x \cdot D_1(x) - x^2 \cdot D_2(x) - ... - x^t \cdot D_t(x)$.

- The LHS i.e. $D(x)$ is of degree t, since we used VSS. However, it need not share c.
- The RHS would share c, since the constant term of $C(x)$ is never modified, but it need not be of degree t. However, the maximum degree of the RHS is $2t$, since $D_t(x)$ is shared using VSS.
- If all the honest parties accept $\Rightarrow$ The LHS and RHS coincide at $2t + 1$ points, when the maximum degree either of them could have is $2t$.
- This implies that LHS = RHS. From their properties, we have that $D(x)$ is of degree t, and it shares c.

Then Step 2 can be concluded successfully, and after Steps 3 and 4 (local computation of Lagrange's Interpolation) the multiplication protocol is complete.

3 Summary: Multiplication Protocol in Malicious Adversarial Model

In the malicious world, we have to ensure that all the players share the correct values during the multiplication protocol. For this purpose, VSS alone does not suffice. We have to construct a special polynomial which can be used to share the product in such a manner that it is not possible to cheat without being caught by the other players. Whenever a player is caught cheating, he is eliminated from the protocol.

4 Conclusion

We now have all the required sub-protocols (verifiable secret-sharing, multiplication and reconstruction) to construct a BGW protocol that is information-theoretically secure against a malicious adversary.