CSA E0 312: Secure Computation September 4, 2015

[Lecture 7-8]

Instructor: Arpita Patra *Submitted by: Divya Ravi*

1 Introduction

In the previous lecture, we looked at an information-theoretic MPC protocol with honest majority for secure evaluation of arithmetic circuits. In this lecture we analyze the efficiency of the protocol and see how it can be improved using the *offline-online paradigm*. We will also look at the feasibility of information-theoretic MPC protocols in dishonest majority setting.

2 Efficiency of the MPC protocol for secure circuit evaluation

We analyze the efficiency of the MPC protocol of secure circuit evaluation in terms of the communication complexity i.e the number of bits communicated during the execution of the protocol. Let us look at the complexity of each step of circuit evaluation separately :

1. **Input Sharing :** Suppose c_I is the number of input gates in the circuit. For every input gate, the party with the secret shares it using LSSS (Linear Shamir-Secret Sharing) scheme. As we have seen, in this scheme, each of the n parties receives a share; this incurs a communication complexity of $\mathsf{O(n)|F_p|}$ for every input gate, where $\mathsf{|F_p|}$ is the number of bits needed to represent an element of the field F_p over which the random t-degree polynomial is chosen in the protocol.

2. **Evaluation of addition gates :** Suppose c_A is the number of addition gates in the circuit. We have seen that evaluation of the addition gate involves only local computation due to the linear property of Shamir-Sharing. Therefore the complexity of this step is NIL.

3. **Evaluation of the multiplication gate :** Suppose c_M is the number of multiplication gates in the circuit. We have seen that for evaluation of multiplication gate, a degree reduction technique is required where *each of the n parties shamir-shares a value* on the product polynomial. Since each instance of sharing a value among n parties incurs $\mathsf{O(n)|F_p|}$ complexity, the total complexity incurred at this step becomes $\mathsf{O(n^2)|F_p|}$ for each multiplication gate.

4. **Output reconstruction :** Suppose c_O is the number of output gates in the circuit. To reconstruct the output, *each* of the n parties send its value of the share to *every other* party. Thus, this incurs a communication complexity of $\mathsf{O(n^2)|F_p|}$ for each output gate.

Thus, the communication complexity for the circuit with c_I input gates, c_A addition gates, c_M multiplication gates and c_O output gates is:

$$O(c_I n + c_M n^2 + c_O n^2)|\mathsf{F_p}|$$

Can we optimize this further? In this lecture, we will see how to achieve the goal of bringing down this complexity to $O(c_I n + c_M n + c_O n)|\mathsf{F_p}|$ bits.[1]Complexity can also be viewed in terms of the *number of rounds* of interaction between the parties. In this protocol of circuit evaluation, the *round complexity* is $O(d)$ where *d is the multiplicative depth* of the circuit. Computation at multiplication gates of the same level of the circuit can be done in parallel.

3 Offline-Online Paradigm

The **'offline-online'** paradigm is considered to be synonymous with 'efficiency'and is extensively used in most of the recent MPC protocols. A protocol can be viewed as being divided into two phases - **offline phase** and the **online phase**

- **Offline Phase :** In the offline phase, neither the input nor the function to be evaluated needs to be specified. Raw data is generated and some precomputations are done so that the online phase can be made faster. This phase is generally not expected to be very efficient as it occurs before the actual interaction between the parties.
- **Online Phase :** The online phase uses the precomputed material created in the offline phase to evaluate the actual circuit. This is expected to be blazing fast.

The *motivation* of this offline-online approach is that the online phase is expected to be made fast by using only *inexpensive computation*. This is particularly useful in settings where parties know in advance that some computation has to be performed, and low online latency is desired. A classical example is the one of an airline company that wants to check the list of passengers on a flight against a database of blacklisted passengers (and neither the list of passenger nor the blacklist should be publicly disclosed). Here the final list of passengers might be ready only few minutes before take-off, while the flight has been scheduled way in advance. We will now see how the offline-online approach can be adopted for secure circuit evaluation. Sharing occurs in the offline phase and the online phase involves only reconstruction.

Remark : In many secret-sharing protocols, the reconstruction phase is usually fast compared to the sharing phase. For such protocols, this offline-online approach will be particularly useful since sharing happens in the offline phase.

3.1 Online Phase Complexity

We have seen that input sharing, computation of multiplication and reconstruction of output are the steps that involve interaction and incur high complexity. Let us see how these steps

[1]Is it possible to attain complexity of constant order? Yes, in computational setting it has been proven that it is possible to evaluate any function with poly power in constant time

can be *reduced to reconstructions alone* and how precomputed data can be used to make the online phase faster and more efficient.

3.1.1 Input Sharing using one reconstruction

Suppose in the offline phase, the raw data generated is (n, t) *Shamir-sharing of a random, secret value* r. We will see how this can be generated later, let's focus now on how the online phase can use this data for input sharing assuming it is available. In the online phase, suppose the party P_i wishes to initiate (n, t) Shamir-sharing of the actual secret X, it can be done using *single reconstruction* as follows -

- Suppose (n, t) Shamir-Sharing of a secret, random value r had been done in the offline phase and r_1, r_2 .... r_n are the respective shares of the n parties.
- The party P_i who wishes to share actual secret X will alone reconstruct r, this is the *only reconstruction that is needed*. The value r still remains random and secret for the other parties.
- Now P_i sends $(X + r)$ to each of the parties as shown in figure 1. The secrecy of the input X is maintained as r is a random, secret value to the other parties and therefore $(X + r)$ acts as a one-time pad for X.
- Each party P_j can now locally compute $(X + r) - r_j$. Due to the linearity of Shamir-Sharing, this can be viewed as if each party contains a point on a polynomial of degree atmost t whose constant term is $(X + r - r)$ which is nothing but the *(n, t) shamir-sharing of X.*

Thus input sharing can be reduced to a single reconstruction in the online phase.

3.1.2 Beaver's circuit-randomization technique for multiplication

We have seen that one of the major bottleneck in the shared evaluation of the circuit is to evaluate the multiplication gates. Beaver's technique for multiplication forms the core of the offline-online paradigm. Beaver's circuit-randomization technique is commmonly used in the online phase for the evaluation of multiplication gates. Here, the gates are evaluated using *pre-computed, t-shared random multiplication triples.* Consider, for now that we have an oracle in the offline phase that generates such triples (a, b, c) *where* a, b *are random and private values independent of the actual inputs to the multiplication gates and* $c = ab$. We will now see how, given such triples, we can evaluate the multiplication gate in the online phase using just *two reconstructions.*

Let x, y be the actual inputs to the multiplication gate. If the product xy can be written as a linear combination of a, b and $c(= ab)$ where the combiners will be publicly known and will not leak any information about x and y then (n, t) Shamir-sharing of a, b and c beforehand will enable each party to easily obtain (n, t) Shamir-sharing of xy. This is done as follows -

$$\begin{aligned} xy &= ((x - a) + a)((x - b) + b) \\ &= (\alpha + a)(\beta + b) \\ &= ab + \alpha b + \beta a + \alpha\beta \end{aligned} \tag{1}$$

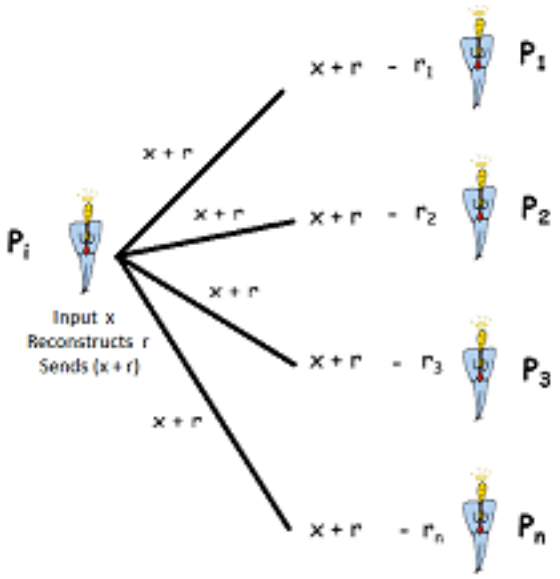

Figure 1: Input Sharing in online phase using precomputed sharing of a secret, random value r

where $\alpha = (x - a)$ and $\beta = (x - b)$

Using the above equation, we can now combine sharing of a, b and ab using the combiners to get sharing of xy as follows -

- Assume that in the offline phase (n, t) Shamir-sharing of a, b and $c(= ab)$ was done.
- Suppose x_i, y_i are the shares received by party P_i by (n, t) Shamir-sharing of x and y respectively which are inputs to the multiplication gate.
- Then each party can now locally compute (x_i - a_i) and (y_i - b_i) i.e their shares for (n, t) Shamir sharing of α and β respectively by the linearity property of sharing.
- Now these shares can be used to reconstruct α and β. These are the *two reconstructions* needed in the online phase. Once this is done, it will easily enable each party to use the equation (1) to get the (n, t) Shamir-sharing of xy as shown in figure 2.

An important aspect which should not be missed is the fact that α and β do not reveal any information about the inputs x and y since a, b remain *random secret values*. In order to ensure this, one should *not use the same multiplication triple* for more than one evaluation. Thus the oracle in the offline phase should ideally give c_M multiplication triples where c_M is the number of multiplication gates in the circuit. Since the triples are independent of the input, Beaver's trick is to generate many triples in parallel in the offline phase so that the efficiency will be amortized in large-sized circuits.[2] Thus, evaluation of the multiplication gate is reduced to two reconstructions in the online phase.

[2]Note that parallelization is possible in the offline phase since the triples are independent of the actual input.

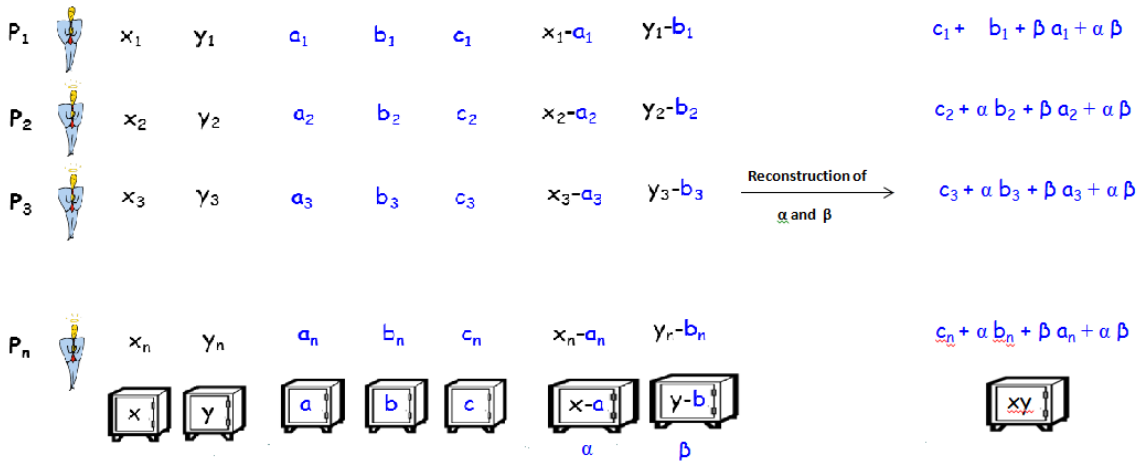

Figure 2: Evaluation of multiplication gate using Beaver's randomization technique

3.1.3 Reconstruction of output of Shamir-Sharing for semi-honest adversary

We have seen in the previous sections how input sharing and multiplication gate evaluation can be done with one and two reconstructions respectively. This invokes the requirement to try to nail the reconstruction step and make it more efficient. Let us see how we can reduce the complexity of output reconstruction step which is $\mathsf{O}(\mathsf{n}^2)$ according to what we analyzed previously.

Recall that in the output reconstruction that we have seen, each party sends its share to every other party, then uses Lagrange's Interpolation to get the t-degree polynomial and recover the secret which is the constant term of the polynomial. This incurs $\mathsf{O}(\mathsf{n}^2)$ communication complexity. There is a simple way to reduce this communication complexity-

- Suppose all the parties send their shares to only one particular party, say P_1 for instance.
- Now P_1 can reconstruct the t-degree polynomial and recover the secret X.
- P_1 sends the secret X to all other parties.

This will incur a complexity of only $\mathsf{O}(\mathsf{n})$ since each of the n parties communicates with P_1 alone. This protocol works correctly as long as we trust that P_1 reconstructs and sends the secret X to all parties correctly as he is supposed to. Since we are in the *semi-honest* setting we can make this assumption that P_1 will not deviate from the protocol. Thus output reconstruction in the semi-honest setting can be done with $\mathsf{O}(\mathsf{n})$ complexity.

In this section, we have seen how the online phase can be made more efficient - By reduction of input sharing to one reconstruction, multiplication gate evaluation to two reconstructions and a way to reconstruct output with $\mathsf{O}(\mathsf{n})$ complexity. Therefore

Online-Phase Complexity : $O(c_I n + c_M n + c_O n)|\mathsf{F_p}|$

3.2 Offline Phase Complexity

In the previous section, we assumed that the precomputed material was available in the online phase. Now lets see how that can be generated. We have seen that for input sharing and multiplication gate evaluation we need random secret values. Let us look at a general *way to produce* $(c_M + c_I)$ *shared, random, secret multiplication triples.*[3] As the name multiplication triples suggests, the goal is to generate $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ which are secret shared using Linear Shamir-secret sharing where *a,b,c are random and secret and* $c = ab$. We can divide this goal into **three tasks** as follows :

1. **Generation of Secret Sharing :** We have seen how to do secret sharing using Shamir-secret sharing scheme which incurs $\mathsf{O}(\mathsf{n})$ complexity.

2. **Generation of Secret Sharing where the secret is random and secret :** This is different from the previous task. In the previous task, one party shares the secret which is known to him and the adversary as well if this party is corrupted. In this case the secret should not be known to the adversary either. We will look at how this is done. **a**, **b** of the multiplication triple is generated by this task.

3. **Generation of sharing of random, secret multiplication triple :** Using **a** and **b** generated by task 2, the third component of the triple i.e **c** is generated by this task.

We already know how to achieve task 1. Let us look at how task 2 can be achieved.

3.2.1 Generation of Secret Sharing where the secret is random and secret :

Suppose each party P_i picks a random value and uses (n, t) Shamir-sharing of the random value. If we pick any of these sharings, that may not be equivalent to the secret sharing of a random, secret value since t out of n parties are corrupted. The sharing selected may be random since we are in the semi-honest setting and each party would have picked a random value; but the sharing selected is not really the sharing of a *secret* value in case the party is corrupted and thereby the value is known to the adversary. We need the *sharing of a random as well as secret value.* We can use a *randomness extractor* for this purpose.

Randomness Extractor :
Randomness extractor is simply an algorithm which takes an input and outputs a random value. To get a (n, t) sharing of a secret random value from the (n, t) sharing of random values by n parties, we can apply a randomness extractor on $(a_1, a_2....a_n)$ where a_i is the share P_i. Here we can use the *simplest randomness extractor which is addition.* Since only t out of n inputs $(a_1, a_2....a_n)$ will be known to the adversary, the sum of the (n, t) Shamir-sharings will be the (n, t) Shamir-sharing of a random secret value and hence our goal has been achieved. However this is *not efficient* since we started with $(n - t)$ random secret value sharings i.e the shares of the honest parties but we extracted *just one random value.* We can achieve a more efficient randomness extractor as shall see further.

[3] For the sake of simplicity, we can assume that the random secret values for input sharing is nothing but the first component of the triple generated.

Efficient Randomness Extractor :
We can view the individual shares of the parties i.e a_1, a_2... a_n to be n points on a $(n-1)$ degree polynomial $f(x)$ such that $f(1) = a_1, f(2) = a_2.....f(n) = a_n$. Among these n points, $(n-t)$ are randomly chosen and t points may be non-random and known to the adversary. We make the following claim.

Claim 1 *If we consider any $(n-t)$ points on $f(x)$ at points other that $x = \{1, 2, ...n\}$, say $f(n+1).....f(n+n-t)$, then these points will be random.*

Proof Without loss of generality say parties $1, 2....t$ are corrupted and thus a_1,.... a_t are known to the adversary. We can define a mapping $M_{a_1...a_t} : F^{n-t} \rightarrow F^{n-t}$ as follows -

- The function takes as input $(n-t)$ values from the domain at random. These values along with the t values known to the adversary will form a set of n points.
- These n points are used to define a unique polynomial $f(x)$ of degree atmost $(n-1)$. In other words $f(1) = a_1, f(2) = a_2.....f(n) = a_n$.
- The output of this function $M_{a_1...a_t}$ is the value of the polynomial $f(x)$ evaluated at $(n+1), ...(n+n-t)$.

We can check that this mapping will be a bijection. This can be inferred from the following two observations :

1. Suppose the mapping $M_{a_1...a_t} : F^{n-t} \rightarrow F^{n-t}$ as defined above is not one-one. This means that there are two different $(n-t)$ tuples in the domain which have the same output. Let us consider these two tuples. These $(n-t)$ points combined with a fixed set of t points will form two different sets of n points, and thereby define *two different polynomials* of degree atmost $(n-1)$ say $f_1(x)$ and $f_2(x)$.[4] By our assumption, the output of both the $(n-t)$ tuples is the same. This means that

$$\begin{aligned} f_1(n+1) &= f_2(n+1); \\ f_1(n+2) &= f_2(n+2); ..... \\ f_1(n+n-t) &= f_2(n+n-t) \end{aligned} \tag{2}$$

   We already know that f_1 and f_2 coincide at the t fixed points $\{1, 2....t\}$. If they coincide on another $(n-t)$ points i.e at $\{(n+1), ...(n+n-t)\}$ as well, this would imply that f_1 and f_2 are two different polynomials of $(n-1)$ degree which have the same value at n points. This is a contradiction since two different $(n-1)$ degree polynomials cannot coincide at n or more points. Thus our assumption that the mapping is not one-one is false. Hence, we can conclude that the mapping M is *one-one*.

2. The *onto property* of the mapping M also holds. Consider a $(n-t)$ set of values from the range. Value at the first t points is fixed. These n points will define a unique polynomial $f(x)$ of degree atmost $(n-1)$. The preimage will be the value of this polynomial at $\{(t+1)....n\}$. Since every set of values from the range will have a preimage, the mapping is onto.

[4] This is due to the fact that a set of n points uniquely defines a polynomial of degree $(n-1)$

We have proved that the mapping is bijective in nature. Since we have a uniform distribution on the domain (uniform over F^{n-t}) we get the same distribution on the range as well. Thus the claim that the distribution of the value of the polynomial $f(x)$ at $(n-t)$ points will be random holds true. ■

Thus we started with $(n-t)$ random values and have obtained $(n-t)$ random values in the output as well. This is the optimal efficiency since randomness cannot be expanded. Shamir-sharing of these $(n-t)$ values that is of $f(n+1).....f(2n-t)$ can be obtained as usual by Lagrange's interpolation formula -

$$\begin{aligned} f(x) &= \sum_{i \in \{i...n\}} a_i \delta_i(x) \\ a_(n+i) = f(n+i) &= \sum_{i \in \{i...n\}} a_i \delta_i(n+i) \end{aligned} \tag{3}$$

Shamir-sharing of n values took $\mathsf{O(n^2)}$ complexity but we have obtained $(n-t)$ sharings of random secret values together. Thus the *amortized cost for one sharing* of a random secret value i.e for achieving one instance of task 2 is $\mathsf{O(n)}$.

We have seen how to achieve the first two tasks so far. Once $\mathbf{a}$ and $\mathbf{b}$ are generated using task 2, the full multiplication triple can be generated by simply using the *secure multiplication protocol* to evaluate the third component of the triple i.e $\mathbf{c} = \mathbf{a}.\mathbf{b}$. This will incur a communication complexity of $\mathsf{O(n^2)}$ as we have seen.[5] Therefore, we have seen how to achieve the following efficiency -

Offline Complexity : $O(c_I n + c_M n^2)|\mathsf{F_p}|$
Online Complexity : $O(c_I n + c_M n + c_O n)|\mathsf{F_p}|$
Total Complexity : $O(c_I n + c_M n^2 + c_O n)|\mathsf{F_p}|$

3.3 Alternate way to evaluate multiplication using single reconstruction

We have seen how to use precomputed multiplication triples to evaluate multiplication gates in the circuit. Another type of raw material that can be used is $(n, 2t)$ *and* (n, t) *sharing of a random value*. How can we use this for evaluation of multiplication gate?

- Suppose we have access to an oracle which gives us $(n, 2t)$ and (n, t) sharing of random values. As shown in the figure 3, suppose x_i and y_i are the shares corresponding to the inputs x and y respectively.
- Let A_i denote the $(n, 2t)$ share of party P_i corresponding to the $(n, 2t)$ Shamir sharing of random value $\mathbf{a}$.
- Let a_i denote the (n, t) share of party P_i corresponding to the (n, t) Shamir sharing of same random value $\mathbf{a}$.
- Each party locally computes $(x_i y_i - A_i)$. This will correspond to $(n, 2t)$ *sharing of* $(xy - a)$ due to the linearity of the sharing.

[5] It is possible to reduce the complexity of generation of triple sharing to $\mathsf{O(n)}$ with statistical security and with n = 3t + 1.

- Now a *single reconstruction* is used to recover $(xy-a)$ from the $(n, 2t)$ shamir sharing. This is feasible since we have $n = 2t+1$ parties which suffice to recover the polynomial of degree $2t$ corresponding to the $(n, 2t)$ sharing of $(xy - a)$.
- Now, using the linearity property of shamir-sharing each party can locally compute $a_i + (xy - a)$ to obtain (n, t) sharing of xy which is exactly what we need.

Using this approach, the online complexity becomes

Online Complexity : $O(c_I n + c_M n + c_O n)|\mathsf{F_p}|$

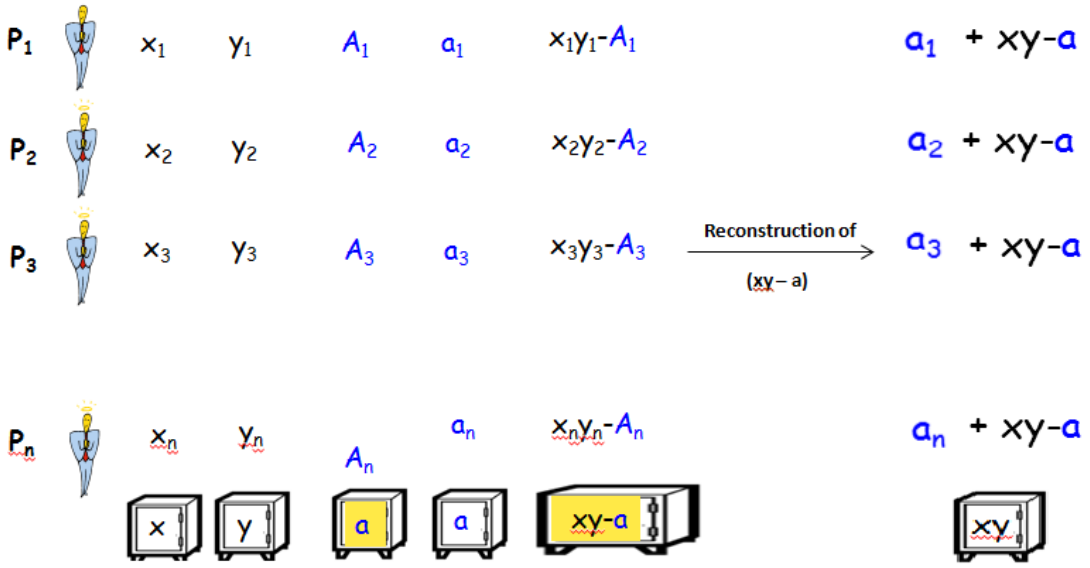

Figure 3: Evaluation of multiplication using (n,(2t,t)) sharing of secret random value a

Now the question that remains is how to generate this raw data i.e we need $(c_M + c_I)$ $(n, (2t, t))$[6] secret sharing of random secret values. An important aspect of secrecy is that **a** remains a random secret value throughout the online phase and thus the secrecy of xy is ensured by relying on this feature. Hence we need an instance of $(n, (2t, t))$ secret sharing of a random secret value for each multiplication and input gate. We can achieve this by the following two tasks.

1. **Generation of** $(n, 2t)$ **and** (n, t) **Secret sharing :** We know how this can be done using Shamir-secret sharing.
2. **Generation of secret sharing where the secret sharing is random and secret :** We have seen how this can be done with an amortized communication complexity of $\mathsf{O(n)}$ in section 3.2.1.

Thus, using $(n, (2t, t))$ precomputed secret sharing, we get an alternate efficient way to evaluate multiplication gates with an amortized complexity of $\mathsf{O(n)}$.

[6] (n,(2t,t)) refers to (n, t) and $(n, 2t)$ sharing of the same value

4 Impossibility of information-theoretic MPC protocol with $n \leq 2t$

Till now we have considered only honest majority setting, i.e where $n > 2t$. Will the protocols we have discussed so far for secure evaluation of functions fail if this condition is not satisfied? Where will we face a problem?

1. **Multiplication protocol :** In the evaluation of multiplication gates, we recovered the t-sharing of the product by t-sharing of the input values by making use of the fact that we know $(2t + 1)$ points on the product polynomial of degree $2t$. This was possible only because we had atleast $(n = 2t + 1)$ parties. These parties could each compute a point on the product polynomial locally and these $(2t + 1)$ points sufficed to define the product polynomial of degree atmost $2t$. In dishonest majority setting where $n \leq 2t$ this won't work.

2. **Generation of multiplication triple sharing :** Generating triple sharing involved using the multiplication protocol on pair (a, b) which are random, secret values to obtain triple sharing of (a, b, c) where $c = ab$. As we inferred above, since multiplication won't work in dishonest majority setting, we will be unable to generate triple sharing as well in this manner.

3. $(n, (2t, t))$ **sharing :** We have seen how to use precomputed $(n, (2t, t))$ shares to evaluate multiplication. In this protocol, we use the $(n, 2t)$ sharing of a random secret value **a** and the $(n, 2t)$ sharing of the product xy (where x and y are actual inputs to the multiplication gate) to reconstruct $(xy - a)$. This was further used to obtain (n, t) sharing of the product xy. However, the reconstruction of $(xy - a)$ from its $(n, 2t)$ sharing is possible only if we have atleast $(2t+1)$ shares i.e $n = (2t+1)$ parties. Thus, this will fail in dishonest majority setting.

From the above observations, it looks like secure evaluation of functions which involve multiplication *cannot be done information-theoretically in dishonest majority setting.* Functions involving only linear operations like addition will not pose a problem since they involve only local computation. However, all functions cannot be evaluated information-theoretically in dishonest majority setting. Let us look at a formal proof of this claim.

Claim 2 *Information-theoretic MPC protocols do not exist for all functions if $n \leq 2t$.*

Proof We adopt the following approach to prove the above claim - We show that there cannot exist an MPC protocol which can provide information-theoretic security for the scenario in which $(n = 2)$ and $(t = 1)$, (thereby satisfying $n \leq 2t$) and the *function of multiplication of bits.* This will suffice to make the claim that there exist functions which cannot be computed with information-xtheoretic security in case of dishonest majority.

Towards a contradiction, suppose that there exists an information-theoretic perfectly secure protocol π which enables two parties to evaluate the logical AND of their respective input bits. Suppose the protocol runs as follows -

- At the start of the protocol, each of the two parties P_i have a private input bit b_i and some private randomness r_i.

- The exchange of messages between the two parties start according to the protocol. Without loss of generality, say P_0 sends the first message m_0 followed by $P_1's$ message m_2 and so on till there is sufficient information for both parties to compute the output. Let $T(b_0, b_1)$ denote the transcript of the protocol run with the input of P_0 as b_0 and P_1 as b_1. This is a random variable over the random choice of the parties.

- Both the parties output the logical AND of their respective input bits i.e $(b_0.b_1)$.

Given a transcript $T(b_0, b_1)$ we say that it is *consistent* with input $b_1 = 1$ if there exists r_1 such that running P_1 with input $b_1 = 1$ and randomness r_1 might result in T being generated. Now we can make the following inferences -

1. Since the protocol is *correct* by our assumption, this means that the protocol always leads to the correct output, i.e the logical AND of the respective input bits of the parties.

2. Since the protocol is *perfectly secure* according to our assumption, this means that if the input bit of a party is 0, he should have no information at all about the other party's input bit. This is due to the property of the function of logical AND. Even in the ideal world, if a party's input is 1, he can infer the other party's input based on the output. But if the party's input is 0, then the other party's input bit should ideally remain private. In other words, if $b_0 = 0$, then $T(b_0, b_1)$ should leak nothing about b_1.

Consider the following argument - Say a transcript T' is generated when the protocol is run with $(b_0 = 0, r_0)$.

- By *inference* 2, if $b_1 = 0$ considering the privacy of P_0, we can conclude this transcript T' should be consistent with $b_0 = 1$ as well. This means that there should exist some randomness r'_0 such that the transcript T' is consistent with $(b_0 = 1, r'_0)$.

- If $b_1 = 1$, there cannot exist any randomness r'_0, so that T' is consistent with $(b_0 = 1, r'_0)$. Since T' had been generated when the protocol was run with $b_0 = 0$, due to the correctness of the protocol, *this transcript should result in output* 0. Therefore if the transcript T' is consistent with $(b_0 = 1, r'_0)$, it would lead to the output 0. However, this would violate *inference 1* i.e the correctness since the correct output in this case when $b_0 = 1$ and $b_1 = 1$ is 1, not 0.

Using this idea we can design the following algorithm for the adversary to breach the security of protocol π. We consider that P_0 is corrupted, has input $b_0 = 0$ and tries to break the privacy of input bit of P_1 by this approach -

1. The adversary tries all possible randomness[7] to find r'_0 so that $T(b_0, b_1)$ is consistent with $(b_0 = 1, r'_0)$

2. If found, output $b_1 = 0$ else output $b_1 = 1$.

[7] We consider an adversary with unbounded computing power here, so this is possible

Thus, we have shown a way for the adversary with $b_0 = 0$ to be able to infer the other party's input b_1. This breaches the security of the protocol and we can therefore conclude that there does not exist a information-theoretic secure protocol for the function of multiplication of bits with dishonest majority ($n = 2, t = 1; n \leq 2t$). We can infer from this result that information-theoretic MPC protocols do not exist for all functions if $n \leq 2t$. ■

5 Conclusion

In this lecture we have seen how the offline-online approach of secure evaluation of arithmetic circuits achieves the following complexity in semi-honest and information-theoretic setting.

Offline Complexity	$O(nc_M + nc_I)\|\mathsf{F_p}\|$
Online complexity	$O(c_I n + c_M n + c_O n)\|\mathsf{F_p}\|$
Total complexity	$O(c_I n + c_M n + c_O n)\|\mathsf{F_p}\|$

We then proved the impossibility of an information-theoretic MPC protocol in dishonest majority. Recall that we have seen in one of the previous lectures how one instance of OT can be used for secure multiplication of input bits of two parties (refer figure 4). Since we know that multiplication of two bits is not possible infomation-theoretically, we can conclude that *OT is impossible to achieve information theoretically.* This emphazises the *need to develop computationally secure MPC protocols* as they can achieve some additional properties like constant round complexity, dishonest majority etc which is not possible in information-theoretic MPC protocols. We will explore this in the upcoming lectures.

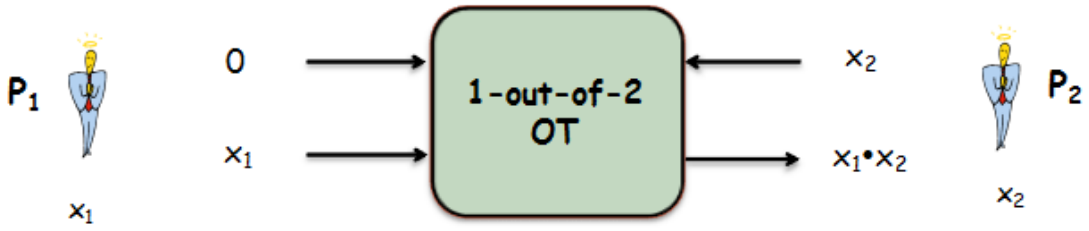

Figure 4: Multiplication of two input bits using an instance of OT

References

[1] Arpita Patra. *http://drona.csa.iisc.ernet.in/ arpita/SecureComputation15.html* . E0 312 - Secure Computation Course Lecture Slides

[2] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing - An Information Theoretic Approach*. Cambridge University Press, 2015.

[3] Claudio Orlandi. *Is Multiparty Computation Any Good In Practice?* ICASSP 2011.