Lecture 38: Secure Multi-party Computation

Problem Statement I

- Suppose Alice has private input x, and Bob has private input y
- Alice and Bob are interested in computing $z = f(x, y)$ such that each party obtains *no additional information than* z
  - For the current presentation, we shall assume that $f(\cdot, \cdot)$ is a deterministic function of the inputs. It is possible to consider randomized functions as well, but we shall not cover it in this course.
  - In this course we shall restrict to two-parties. In general, there can be n-parties where the private-inputs of the parties, respectively, are $x_1, x_2, \ldots, x_n$.
  - All parties know the function "f" that they are evaluating.
  - In this course we shall consider adversaries that are *semi-honest*, i.e., the parties follow the protocol but are curious to know more information about the private input of the other parties

Problem Statement II

Example.

- Suppose countries A and B have two military satellites orbiting the earth
- The input for party 1 is the location and trajectory of their satellite
- The input for party 2 is the location and trajectory of their satellite
- The function "f" computes whether the two satellites will collide in the next 10 minutes (the output is true or false)

Problem Statement III

Correctly Evaluating Functions.

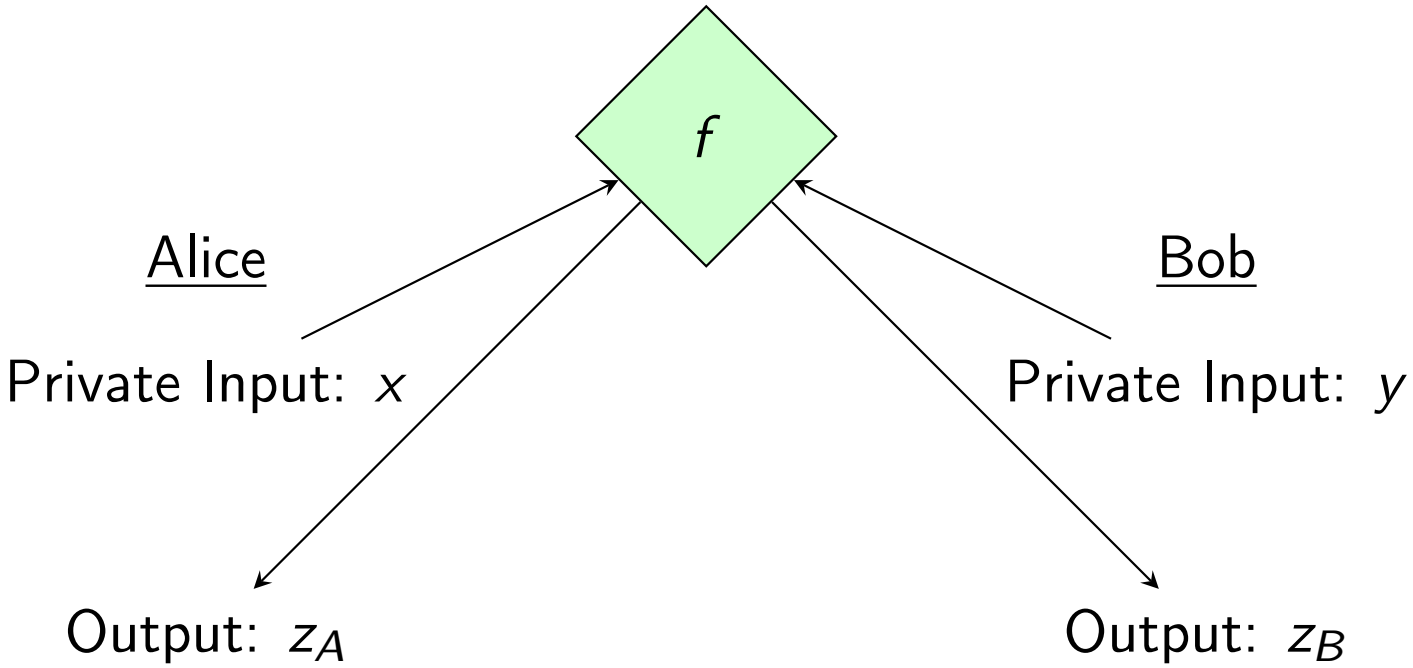
- Note that it is trivial to correctly evaluate a function. First, each party declares its private input to all other parties. Next, each party computes the function.
- However, this is not secure. Because, the parties (potentially) learn more than the output z.
- For example, in the example we consider, parties shall learn the location and trajectory of each other's military satellites, which is definitely not secure!
- The difficulty of the problem arises from the security constraint (and not the correctness constraint)

Trusted Third-party

- Suppose Alice and Bob have a trusted third-party (TTP)
- Then, Alice and Bob can send their respective private inputs x and y to the TTP, and the TTP computes $z = f(x, y)$ and sends z to both Alice and Bob
- In our example, suppose, both countries trust Switzerland. Countries A and B share the location and trajectory of their military satellites, and Switzerland computes whether they will collide in the next 10 minutes and tells the answer to both the countries. This is a secure solution!
- **Lesson Learned.** If there is a trusted third-party, then secure computation is easy.
- The non-triviality of secure multi-party computation arises because there is no trusted third-party in the real world

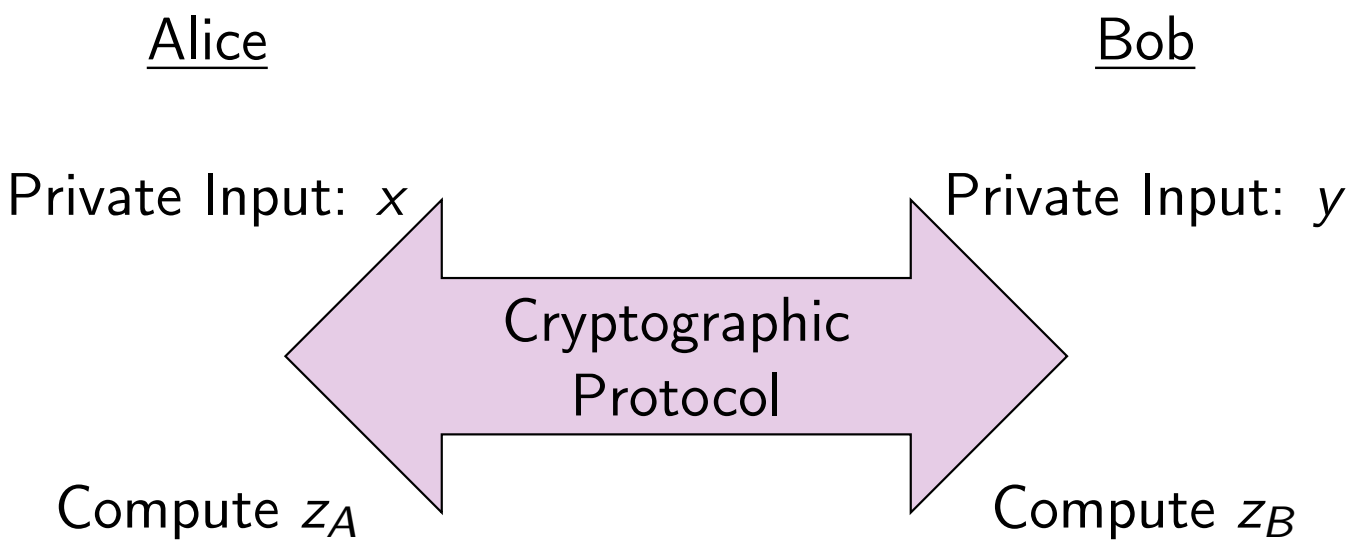
Ideal World

- In the ideal world, parties have a trusted third-party
- In this world, parties send their respective input to the TTP and the TTP performs the computation on behalf of the parties, and reports back their respective outputs z_A and z_B



Real World

- In the real world, there is no trusted third-party
- Our goal is to algorithmically implement the TTP using cryptographic protocols
- Security implies that "Alice learns no more than z_A" and "Bob learns no more than z_B"



Example of Secure Computation: XOR I

- Suppose Alice has private input $x \in \{0, 1\}$
- Suppose Bob has private input $y \in \{0, 1\}$
- They are interested in computing $z_A = z_B = f(x, y) = x \oplus y$
- The function is equivalently represented by the following matrix

Alice Input \ Bob Input	0	1
0	0	1
1	1	0

Example of Secure Computation: XOR II

- Note that if the private input of Alice is $x = 0$ then z_A reveals the private input of Bob (Think about this)
- Similarly, if the private input of Alice is $x = 1$ then z_A reveals the private input of Bob (Think)
- So, it is secure for Bob to just reveal his input to Alice (because Alice finds out Bob's input from her output anyways)
- Analogously, Bob also finds out the private input of Alice always. So, it is secure for Alice to reveal her private input to Bob (because Bob finds out Alice's input from his output anyways)
- So, the following protocol is a secure protocol to compute $z_A = z_B = x \oplus y$

1. Alice sends her private input x to Bob
2. Bob sends his private input y to Alice
3. Both parties compute $z_A = z_B = x \oplus y$

Example of Secure Computation: OR I

- Suppose Alice has private input $x \in \{0, 1\}$
- Suppose Bob has private input $y \in \{0, 1\}$
- They are interested in computing $z_A = z_B = f(x, y) = x \vee y$
- The function is equivalently represented by the following matrix

Alice Input \ Bob Input	0	1
0	0	1
1	1	1

Example of Secure Computation: OR II

- Note that if the private input of Alice is $x = 0$ then z_A reveals the private input of Bob
- However, if the private input of Alice is $x = 1$ then z_A does not reveal anything about Bob's input (Think)
- So, Bob can reveal his private input to Alice only when $x = 0$; otherwise, not
- Analogously, Alice can reveal her private input to Bob only when $y = 0$; otherwise, not
- **The Conclusion.** Alice cannot reveal her private input unless she is sure that $y = 0$. Moreover, Bob cannot reveal his private input unless he is sure that $x = 0$. So, neither party can risk revealing their respective private inputs first. There is a deadlock. This argument can be made more formal mathematically to argue that there is no secure protocol to compute $z_A = z_B = x \vee y$. (This mathematical argument is beyond the scope of this course)

Example of Secure Computation: OR III

Additional Pointer. In fact, one can show that computing OR is complete. That is, if we can securely compute $z_A = z_B = x \vee y$ then we can securely compute any function! (Again, this is beyond the scope of this course)

Example of Secure Computation: MAX I

- Suppose Alice has private input $x \in \{0, 2\}$
- Suppose Bob has private input $y \in \{1, 3\}$
- They are interested in computing
  $z_A = z_B = f(x, y) = \max\{x, y\}$
- The function is equivalently represented by the following matrix

Alice Input \ Bob Input	1	3
0	1	3
2	2	3

Example of Secure Computation: MAX II

- Note that z_A always reveals Bob's private input to Alice (Think)
- Moreover, z_B reveals Alice's private input to Bob if $y = 1$; otherwise, it reveals no information (Think)
- So, it is secure for Bob to reveal his private input to Alice. If $y \neq 3$, then Alice can reveal her private input to Bob.
- So, the following is a secure protocol to compute $z_A = z_B = \max\{x, y\}$, where $x \in \{0, 2\}$ and $y \in \{1, 3\}$.

1. Bob sends y to Alice
2. Alice sends $z = \max\{x, y\}$ to Bob

Example of Secure Computation: The Dutch Flower Auction I

Let us generalize the previous example.

- Suppose Alice and Bob are bidding for an item
- Assume that Alice only bids even amount of money $x \in \{0, 2, \ldots, 2k-2\}$, and Bob bids odd amount of money $y \in \{1, 3, \ldots, 2k-1\}$
- We are interested in computing the winning bid $z_A = z_B = \max\{x, y\}$

Example of Secure Computation: The Dutch Flower Auction II

Crucial Observation.

- Let M be the maximum possible bid. (In this case, we have $M = 2k - 1$.)
- Note that both parties find out whether the bid is M or not. (Think very carefully about this)
- So, the party who can make the bid M declares "whether his bid is M or not" (In this case, Bob can declare whether $y = 2k - 1$ or not)
- If the maximum bid is made then we are done!
- Otherwise, we recurse. (In this case, we recursively compute the maximum where $x \in \{0, 2, \ldots, 2k - 2\}$ and $y \in \{1, 3, \ldots, 2k - 3\}$)

Example of Secure Computation: The Dutch Flower Auction III

The secure protocol, therefore, is

1. If $y = 2k - 1$, then Bob declares $z = 2k - 1$; otherwise he asks Alice to continue
2. If $x = 2k - 2$, then Alice declares $z = 2k - 2$; otherwise she asks Bob to continue
3. If $y = 2k - 3$, then Bob declares $z = 2k - 3$; otherwise he asks Alice to continue
4. And so on...

Example of Secure Computation: The Dutch Flower Auction IV

- However, this protocol is not efficient. The round complexity of the protocol is exponential in the representation size of the parties' inputs.

Oblivious Transfer I

m-choose-1-Oblivious Transfer (OT).

- Alice has private input $(\theta_1, \theta_2, \ldots, \theta_m)$ such that each $\theta_i \in S$ (an arbitrary set S)
- Bob has private input $i \in \{1, 2, \ldots, m\}$
- And, we have $z_A = \text{NULL}$ (the empty string), and $z_B = \theta_i$

Oblivious Transfer II

Security Requirement.

- Note that Alice obtains no additional information about Bob's private input i
- And, Bob obtains no additional information about θ_j, where $j \neq i$

Oblivious Transfer III

Using the RSA assumption, we can construct a secure protocol for m-choose-1 OT.

OT$(\theta_1, \theta_2, \ldots, \theta_n, i)$:

1. Alice picks (u, trap) and sends the trapdoor permutation $f_u : \mathcal{D} \to \mathcal{R}$ to Bob
2. Bob picks $\alpha_i \xleftarrow{\$} \mathcal{D}$ and computes $\beta_i = f_u(\alpha_i)$. For all $j \neq i$, Bob picks $\beta_j \xleftarrow{\$} \mathcal{R}$. Bob sends $(\beta_1, \beta_2, \ldots, \beta_m)$ to Alice.
3. For each j, Alice computes $\alpha_j = f_u^{-1}(\beta_j)$, using the trapdoor trap. For each j, Alice computes the encryption $c_j = \theta_j \cdot \alpha_j$ (i.e., the encryption of θ_j using one-time pad, where the secret-key is α_j). Alice sends $(c_1, c_2, \ldots, c_m)$ to Bob.
4. Bob recovers θ_i by computing c_i/α_i.

Oblivious Transfer IV

Note that Bob cannot recover any other message θ_j, where $j \neq i$, because he does not have the associated secret-key α_j (he only has β_j and it is computationally hard for him to invert β_j because he does not have the trapdoor trap).
Alice's view in this protocol is independent of Bob's input i.

Securely Computing Any Function using Oblivious Transfer I

- Suppose Alice and Bob have private inputs $x \in S_A$ and $y \in S_B$, respectively
- They are interested in securely computing $z_A = z_B = f(x, y)$
- Let us use the m-choose-1 OT protocol to securely compute f

1. Suppose $S_B = \{y_1, y_2, \ldots, y_m\}$. Alice computes $\theta_j = f(x, y_j)$, where $j \in \{1, 2, \ldots, m\}$. Let $y = y_i$. Alice and Bob run the m-choose-1 OT protocol with Alice private input $(\theta_1, \theta_2, \ldots, \theta_m)$ and Bob private input i. At the end of this OT protocol, Bob obtains the output $z = \theta_i = f(x, y_i)$.
2. Bob sends z to Alice

Securely Computing Any Function using Oblivious Transfer II

- Note that this is an efficient protocol only when m is a constant. So, for a constant value of m we have constructed a secure protocol for f that is also efficient.
- Otherwise, the value m is exponentially large in the size of the representation of Bob's input. In this case, the protocol is secure but not efficient. In the next lecture, we shall construct efficient secure computation protocols for any efficient f.