Quick introduction to Secure multiparty computation

Seminario en Zacatecas [Q3 2015]

Luis J. Dominguez Perez

CIMAT-Zac, Septiembre 11 de 2015

Introduction

MPC

Contenido de la sección 1

Introduction

MPC

What if?

- Take two drug companies
- Each has a database of molecules and toxycology results
- They want to combine their results
- Without revealing what molecules are in the databases

What if?

- A government wants to search network traffic for a specific anomalous behavoir
- But the network operator does not want to give access to the netowrk to the government
- And the government does not want to reveal exactly what behavoir it is searching for

Two millionaires problem

- Two millionaires need to know which one is richer, without revealing their wealth amount.

Two millionaires problem

- Two millionaires need to know which one is richer, without revealing their wealth amount.

How?

Two millionaires problem

There are two numbers, n_1, and n_2, and the goal is to solve the inequality $n_1 \geq n_2$? without revealing the actual values of n_1, and n_2.

- Alice's number is n_1
- Bob's number is n_2
- Alice split n_1 as $n_1 = n_1^+ - n_1^-$. Sends n_1^- to Bob
- Bob split n_2 as $n_2 = n_2^+ - n_2^-$. Sends n_2^- to Alice
- Alice sends $n_1^+ + n_2^-$ to a trusted third party Dan
- Bob sends $n_2^+ + n_1^-$ to a trusted third party Dan
- Dan subtracts $(n_2^+ + n_1^-)$ from $(n_1^+ + n_2^-)$ to get $n_1 - n_2$, and announces whether this result is positive or negative.

Two millionaires problem

There are two numbers, n_1, and n_2, and the goal is to solve the inequality $n_1 \geq n_2$? without revealing the actual values of n_1, and n_2.

- Alice's number is n_1
- Bob's number is n_2
- Alice split n_1 as $n_1 = n_1^+ - n_1^-$. Sends n_1^- to Bob
- Bob split n_2 as $n_2 = n_2^+ - n_2^-$. Sends n_2^- to Alice
- Alice sends $n_1^+ + n_2^-$ to a trusted third party Dan
- Bob sends $n_2^+ + n_1^-$ to a trusted third party Dan
- Dan subtracts $(n_2^+ + n_1^-)$ from $(n_1^+ + n_2^-)$ to get $n_1 - n_2$, and announces whether this result is positive or negative.

Why?

Two millionaires problem

Remark

Dan ends up knowing the actual difference $n_1 - n_2$. If there is a leakage of information, any party could calculate the other party's wealth.

Other examples

Cases when we need the computing party not to know our data

- democratic elections
- authentication systems that requires the visitor to be over 18 y.o.
- buying something without transmiting our credit card PIN
- general sql queries on private data over the cloud
- sharing a secret
- drug prices to country-wide health systems
- bidding for public works

*These examples does not necesarily requires multiparty computation.

Two approaches

- Fully Homomorphic Encryption
  - Developed initially in 2009 (only feasible in partially homomorphic scenarios)
  - Party A sends encrypted data to Party B
  - Party B does some computation and returns the encrypted result to Party A
  - Party A decrypts, and find out the answer

- Multi-party computation
  - First scheme developed in 1980's
  - Parties jointly compute function on their inputs using a protocol
  - No information is revealed about the parties inputs.

Two approaches

- Fully Homomorphic Encryption
  - Developed initially in 2009 (only feasible in partially homomorphic scenarios)
  - Party A sends encrypted data to Party B
  - Party B does some computation and returns the encrypted result to Party A
  - Party A decrypts, and find out the answer

- Multi-party computation
  - First scheme developed in 1980's
  - Parties jointly compute function on their inputs using a protocol
  - No information is revealed about the parties inputs.
- there are others in some cases

Differences

In theory both works, but. . .

Differences

In theory both works, but. . .

- FHE: huge computational cost, zero communication
- MPC: virtually no computation cost, huge communication

Practicallity

- FHE - only practical on simplest functions

- MPC - only with semi-honest adversaries (1-out-of-3)

Practicallity

- FHE - only practical on simplest functions

- MPC - only with semi-honest adversaries (1-out-of-3)

We can mix them to make them practical, but this will be out of the scope of this talk :)

Small example - I

Setup

- Assume n parties of which $n-1$ can be malicious
- Assume global (secret) key $\alpha \in \mathbb{F}_p$ is determined
- Each party i holds α_i with

$$\alpha = \alpha_1 + \cdots + \alpha_n$$

Small example - I

- All data is represented by elements in $\mathbb{F}_p$
- A secret value $x \in \mathbb{F}_p$ is shared between the parties as follows:
  - Party i holds a data share x_i
  - Party i holds a "MAC"share $\gamma_i(x)$

  such that

$$x = x_1 + \cdots + x_n \quad \alpha \cdot x = \gamma_1(x) + \cdots + \gamma_n(x)$$

- Note we can share a public constant v by:
  - Party I sets $x_1 = v$
  - Party $i \neq 1$ sets $x_i = 0$
  - Party i sets $\gamma_i(v) = \alpha_i \cdot v$.

Small example - I

Preprocessing model

- Such a sharing of x is denoted by $[x]$
- Our protocol works in the preprocessing model
- We (overnight say) generate a lot of data which is independent of the function to be computed, or its inputs
- In its basic form the data consists of the triples of shared values:

$$[a], [b], [c]$$

such that

$$c = a \cdot b$$

Small example - I

- To perform the computation we utilize the following idea
- Any computation can be represented by a series of additions and multiplications of elements in $\mathbb{F}_p$
- In other words $+$, and $\times$ are a set of Universal Gates over $\mathbb{F}_p$
- We assume trhe players inputs are shared first using the above sharing
- So we all need to know if how to do add and multiply shared values
- (addition will be easy, multiplication will be hard)

Small example - I

Suppose we have two shared values $[x], [y]$

- To compute the result $[z]$ of an addition gate the parties individually execute:
  - $z_i = x_i + y_i$
  - $\gamma_i(z) = \gamma_i(x) + \gamma_i(y)$
- Note this is a local operation and that we end up with:

$$\begin{aligned} z &= \sum z_i = \sum (x_i + y_i) = (\sum x_i) + (\sum y_i) \\ &= x + y, \\ \alpha \cdot z &= \sum \gamma_i(z) = \sum (\gamma_i(x) + \gamma_i(y)) = \alpha \cdot x + \alpha \cdot y \\ &= \alpha \cdot (x + y) \end{aligned}$$

Small example - I

Linear Secret Sharing

- The addiion trick works because we have a Linear Secret Sharing Scheme
- We can locally compute any linear function of shared values
- i.e. given constants v_1, v_2, and v_3 and shared values $[x]$, $[y]$ we can compute:

$$v_1 \cdot [x] + v_2 \cdot [y] + v_3 = [v_1 \cdot x + v_2 \cdot y + v_3]$$

Contenido de la sección 2

Introduction

MPC

Secure multiparty computation

- Secure multi-party computation is a problem that was originally suggested by Yao in 1982.

- Refers to computational systems in which several parties which to jointly compute some value based on individually held secret bits of information, but do not wish to reveal their secrets to anybody in the process.

Secure multiparty computation

- A set of parties with **private** inputs wish to compute some **joint function** of their inputs
- Parties wish to preserve some security properties; i.e., **privacy** and **correctness**
  - Example: secure election protocol
- Security must be preserved in the face of **adversarial behavior** by some of the participants, or by an external party.

Properties

- **Privacy**: No party should learn anything more than the prescribed output
- **Correctness**: Each party is guaranteed that the output it receives is correct.
- **Independence of inputs**: Corrupted parties must choose their inputs independently of the honest parties' inputs.
- **Guaranteed Output Delivery**: Corrupted parties should not be able to prevent honest parties from receiving their output. (no denial-of-service)
- **Fairness**: Corrupted parties should receive their outputs if and only the honest parties also receive their outputs

The adversary

The power of the adversary that attacks a protocol execution most be defined:

- **1. Corruption Strategy**:
  - Static corruption model: the adversary has a fixed set of parties whom it controls
  - Adaptive corruption model: adaptive adversaries are given the capability of corrupting p arties during the computation

The adversary

- **2. Allowed adversarial behavoir**:
  - Semi-honest adversaries: corrupted parties correctly follow the protocol specification
  - Malicious adversaries: the corrupted parties can arbitrarily deviate from the protocol specification

The adversary

- **3. Complexity**:
  - Polynomial-time: the adversary is allowed to run in (probabilistic) polynomial-time
  - Computationally unbounded: The adversary has no computational limits whatsover

Feasibility of the secure multiparty computation

Let m denote the number of participating parties, and let t denote a bound on the number of arties that may be corrupted:

- $t < m/3$: (when less than a third of the parties can be corrupted) secure multiparty protocols with fairness and guaranteed output delivery can be achieved for any function in a point-to-point network.
- $t < m/2$: secure multiparty computation with fairness and guaranteed output delivery can be achieved for any function assuming the parties have access to a broadcast channel.
- $t \geq m/2$: (unlimited number of corrupted parties) secure multiparty protocols can be achieved assuming that the parties have access to a broadcast channel, and that enhanced trapdoor permutations exist.

Remark

- For secure multiparty computation, we need secure protocols. Let's have a quick look at the concerns around them. . .

Protocols and functions

- Cryptography aims for the following (regarding privacy):
  - A secure protocol must reveal **no more information than the output of the function itself**
  - That is, the **process of protocol computation** reveals nothing.
- Cryptography does **not** deal with the question of whether or not the function reveals much information
  - Example: mean of two parties' salaries
- Deciding **which functions to compute** is a different challenge that must also be addressed in the context of privacy preserving data mining.

Security in the protocols

- Consider a secure auction (with secret bids):
  - An **adversary** may wish to learn the bids of all parties – we need **privacy**
  - An adversary may wish to win with a lower bid than the highest – we need **correctness**
  - But, the adversary may also wish to ensure that it always gives the highest bid – we need **independence of inputs**

Analyzing the protocols

- The protocols can be analyzed in an ideal world, or a real world to determine if they are secure or not. . .

Real/Ideal model

- **Ideal model**: parties send inputs to a *trusted third party*, who computes the function and sends the outpus
- **Real model**: parties run a *real protocol* with no trusted third party help

- Informally, a protocol is secure if any attack on a **real protocol** can be carried out (or simulated) in the **ideal model**
- Since essentially **no** attacks can be carried out in the ideal model, then, the security is implied

What model to use?

- **Side A**: Security in the ideal model is **absolute**. Since *no attacks are possible* in the ideal model, we obtain that the same is also true of the real model.

- **Side B**: *Anything that the adversary could have learned or done* in the real model, it could have also learned or done in the ideal model

Example - 2

Computing the sum of powers: $\sum_{i=1}^{k} n_i^r$, where n_i are the p[arties's private numbers, k is the number of parties, and $r > 1$ is an arbitrary number

- P_1 initiates the process by sending a random element n_0 to P_2
- Each P_i, $2 \leq i \leq k-1$, does the following. upon receiving an element m from P_{i-1}, he ads his n_i^r to m, and sends the result to P_{i+1}
- P_k adds his n_i^r to whatever he has received from P_{k-1}, and sends the result to P_1
- P_1 substracts $(n_0 - n_1^r)$ from what he got from P_k; the result now is the sum of all n_i^r, $1 \leq i \leq k$.

Question

- What other functions can be securely computed without revealing intermediate results to any party?

Example - 3

Computing $f(n-1, n_2, n_3) = n_1 n_2 + n_2 n_3$, without computing $n_1 n_2$, or $n2n_3$

- P_1 initiates the process by sending a random element n_0 to P_3
- P_3 adds his n_3 to n_0 and sends $(n_3 + n_0)$ to P_1
- P_2 substracts n_0 from $n_0 + n_3 + n_1$ and multiplies the result by n_2. This is now $n_1 n_2 + n2n_3$

Example - 3

Computing $f(n-1, n_2, n_3) = n_1 n_2 + n_2 n_3$, without computing $n_1 n_2$, or $n2n_3$

- P_1 initiates the process by sending a random element n_0 to P_3
- P_3 adds his n_3 to n_0 and sends $(n_3 + n_0)$ to P_1
- P_2 substracts n_0 from $n_0 + n_3 + n_1$ and multiplies the result by n_2. This is now $n_1 n_2 + n2n_3$

Can we modify the protocol for not revealing the n_i?

Example - 4

Compute $f(n_1, n_2, n_3) = n_1 n_2 + g(n_3)$, where $g(\cdot)$ is some function

- P_1 initiates the process by sending a random element a_0 to P_2
- P_2 multiplies a_0 by his n_2, and sends the result to P_3
- P_3 multiplies $a_0 n_2$ by a random element c_0, and send the result to P_1
- P_1 multiplies $a_0 n_2 c_0$ by his n_1, divides by a_0, and sends the result, which is $n_1 n_2 c_0$, back to P_3
- P_3 divides $n_1 n_2 c_0$ by c_0, and adds $g(n_3)$, to end uo with $n_1 n_2 + g(n_3)$

Example - 5

Suppose a player P_i receives a number M that has to be split in a sum of k private numbers. (all operations $\mathbb{Z}_k$)

- P_i initiates the process by sending $M - m_i$ to P_{i+1}, where m_i is a random number (either positive, or negative)
- Each subsecuent P_j does the following: Upon receiving a number m from P_{j-1}, he substracts a random number m_j from m, and send the result to P_{j+1}. The number m_j is now P_j's secret summand.
- When this process gets back to P_i, he ads m_i to whatever he got from P_{i-1}; the result is the secret summand

Example - 6

Secret sharing (k, k)-threshold scheme. The dealer D wants to distribute shares of a secret number N to k players P_i so that, if P_i gets a number s_i, then $\sum_{i=1}^{k} s_i = N$

- D arbitrarly splits N in a sum of k integers: $N = \sum_{i=1}^{k} n_i$
- The loop: at Step i of the loop, D sends n_i to P_i, and P_i initiates the above Subroutine (Example 5) to distribute shares n_{ji} of n_i among the players, so that $\sum_{j=1}^{k} n_{ji} = n_i$
- After all k steps of the loop are completed, each player P_i ends up with k numbers n_{ji} that sum up to $s_i = \sum_{j=1}^{k} n_{ji}$. Then $\sum_{i=1}^{k} s_i = N$

What's next?

Constructing protocols in

- the semi-honest model
- in the malicious model

Construct protocols based on

- Oblivious Transfer
- More on Random Shares
- Circuit computation
- Bit Commitment
- Coin tossing
- Zero knowledge
- Homomorphic encryption

Directions of Multiparty Computation

- Privacy-preserving data mining considers the problem of running data mining algorithms on confidential data that is not supposed to be revealed –even to the party running the algorithm.
  - Option 1: The data is divided among two or more different parties
  - Option 2: Some statistical data that is to be released may contain confidential data; hence, it is first modified so that the data does not compromise anyone's confidential data, and it is possible to obtain meaningful results by running data mining algorithms on the modified data set.

Example

- **PIR**: Private Information Retrieval
- Aim: allow a client to query a database without the server learning what the query is
- Definition:
  - Correctness: when the server is *semi-honest*, the client always obtains the correct result
  - Privacy: the view of any (even *malicious*) server when the client query is i is indistinguishable from its *view* when the client's query is j.
  - Sometimes privacy is also required (only the query is known, nothing else).

Example(?)

- **Distributed data mining**
  - Each party sends its database to the trusted party
  - Trusted party gathers the union of all databases, runs the data mining algorithm, and send the output

For more information,

- Foundations of Cryptography II, Oded Goldreich
- Secure Multiparty Computation for Privacy-Preserving Data Mining by Lindell, and Pinkas.