Secure Computation

Maryam Zarezadeh

Associate Researcher

maryam.zarezadeh@barkhauseninstitut.org

Some slides taken from lectures of Ran Cohen, Yehuda Lindell, Mike Rosulek
**Definition:**

- **Secure computation (SC)** (also known as **Secure multi-party computation (SMPC)**, **multi-party computation (MPC)**) is a subfield of cryptography with the goal of creating methods for parties to jointly compute a function over their inputs while keeping those inputs private.

- **SC** protocols can enable data scientists and analysts to compliantly, securely, and privately compute on **distributed data** without ever exposing or moving it.

- Researchers are making **SC** faster and easier to use for application software developers.
Scenario: Private Auction

Many parties wish to execute a private auction

• The highest bid wins

• Only the highest bid (and bidder) is revealed
**Scenario: Private Auction**

Many parties wish to execute a private auction

- The highest bid wins
- Only the highest bid (and bidder) is revealed

**Solution:** use a trusted auctioneer
Secure Computation

• In the scenario the solution of an external trusted third party works
• Trusting a third party is a very strong assumption
• Can we do better?
• We would like a solution with the same security guarantees, but without using any trusted party
Secure Computation

**Goal:** use a protocol to emulate the trusted party
The setting

• Parties $P_1, \ldots, P_n$
• Party $P_i$ has private input $x_i$
• The parties wish to jointly compute a function $y = f(x_1, \ldots, x_n)$
• The computation must preserve certain security properties, even if some of the parties collude and maliciously attack the protocol
• Normally, this is modeled by an external adversary $\mathcal{A}$ that corrupts some parties and coordinates their actions
Security Requirements

- **Correctness**: parties obtain correct output (even if some parties misbehave)
- **Privacy**: only the output is learned (nothing else)
- **Independence of inputs**: parties cannot choose their inputs as a function of other parties’ inputs
- **Fairness**: if one party learns the output, then all parties learn the output
- **Guaranteed output delivery**: all honest parties learn the output
Auction Example – Security Requirements

- **Correctness**: $\mathcal{A}$ can’t win using lower bid than the highest
- **Privacy**: $\mathcal{A}$ learns an upper bound on all inputs, nothing else
- **Independence of inputs**: $\mathcal{A}$ can’t bid one dollar more than the highest (honest) bid
- **Fairness**: $\mathcal{A}$ can’t abort the auction if his bid isn’t the highest (i.e., after learning the result)
- **Guaranteed output delivery**: $\mathcal{A}$ can’t abort (stronger than fairness, no DoS attacks)
Who is Richer?

Millionaires’ Problem

$X > Y$ ?!!
Secure string matching

Bob’s Genome: ACGT…

Alice’s Genome: ACTG…

Can Alice and Bob compute a function of their private data without exposing anything about their data besides the result?
Secret Sharing

$s$ from $F_p$

$F_p = (Z_p, +, \cdot)$ is a field

$S \setminus s_1$  $S \setminus s_2$  $\ldots$  $S \setminus s_n$

$\gg$ Choose random shares $s_1, \ldots, s_n$ from $F_p$ s.t. $s_1 + \ldots + s_n = s$

$\gg$ $S = \{s_1, \ldots, s_n\}$

$\gg$ Together all the parties know $S$

$\gg$ Individual party has no information about $S$. 
Secure Addition $\gamma = x_1 + x_2 + x_3$ (assume $n=3$ parties)

No party even with unbounded power learns nothing more than $\gamma$!
Secure bit multiplication \( y = x_1 \cdot x_2 \)

\[
y = x_1 \cdot x_2 \\
= (x_{11} + x_{12}) \cdot (x_{21} + x_{22}) \\
= (x_{11} \cdot x_{21} + x_{11} \cdot x_{22} + x_{12} \cdot x_{21} + x_{12} \cdot x_{22})
\]

\[
\begin{array}{cccc}
x_1 & x_2 \\
x_{11} & x_{12} & x_{21} & x_{22} \\
x_{12} & \cdot & x_{22} & = x_{12} \cdot x_{22} \\
x_{11} & \cdot & x_{21} & = x_{11} \cdot x_{21}
\end{array}
\]
Oblivious Transfer (OT)

- **Sender** holds two bits \( x_0 \) and \( x_1 \).
- **Receiver** holds a choice bit \( b \).
- **Receiver** should learn \( x_b \), **sender** should learn **nothing**.
Secure bit multiplication $y = x_1 \cdot x_2$

$P_1$

$a_0 \rightarrow 1$-out-of-$2$ OT

$a_1 \rightarrow b$

$P_2$

$b \rightarrow (1-b) \cdot a_0 + b \cdot a_1 = a_b$

$a_0 = 0 \rightarrow 1$-out-of-$2$ OT

$a_1 = x_1 \rightarrow b = x_2$

$(1-x_2) \cdot 0 + x_2 \cdot x_1 = x_1 \cdot x_2$
How to Define Security

Option 1: property-based definition

• Define a list of security requirements for the task
• Analyze security concerns for each specific problem
• Difficult to analyze complex tasks
• How do we know if all concerns are covered?
• Definitions are application dependent (no general results, need to redefine each time).
• **Option 2: the real/ideal paradigm**

  • Whatever an adversary can achieve by attacking a real protocol can also be achieved by attacking an ideal computation involving a trusted party

  • Formalized via a simulator

  • The real/ideal model paradigm:
    • **Ideal model:** parties send inputs to a trusted party, who computes the function and sends the outputs.
    • **Real model:** parties run a real protocol with no trusted help.

  • **Informally:** a protocol is secure if any attack on a real protocol can be carried out in the ideal model.

  • Since no attacks can be carried out in the ideal model, security is implied.
The Security Definition:

For every real adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that:

**REAL**

Protocol interaction

$\approx$

**IDEAL**

Trusted party
Ideal World

1) Each party sends its input to the trusted party
2) The trusted party computes $y = f(x_1, \ldots, x_n)$
3) Trusted party sends $y$ to each party
Real World

Parties run a protocol $\pi$ on inputs $(x_1, \ldots x_n)$
Simulation-Based Security
Simulation-Based Security

Distinguisher $\mathcal{D}$
Simulation-Based Security

Distinguisher $\mathcal{D}$

Adversary $\mathcal{A}$
Simulation-Based Security

Simulator $S$  Distinguisher $\mathcal{D}$  Adversary $\mathcal{A}$
Simulation-Based Security

The distinguisher $\mathcal{D}$:

- Gives inputs to parties
- Gets back output from parties and from adversary/simulator
- Guesses which world it is real/ideal

Protocol $\pi$ securely computes $f$ if $\forall \mathcal{A} \exists \mathcal{S} \forall \mathcal{D}$ distinguishing success is “small”
Sanity check

✓ Correctness
✓ Privacy
✓ Independence of inputs

✓ Fairness
✓ Guaranteed output delivery
The Definition Cont’d

A definition of an SC task involves defining:

- **Functionality**: what do we want to compute?
- **Security type**: how strong protection do we want?
- **Adversarial model**: what do we want to protect against?
- **Network model**: in what setting are we going to do it?
The Functionality

- The code of the trusted party
- Captures inevitable vulnerabilities
- Sometimes useful to let the functionality talk to the ideal-world adversary (simulator)
- We will focus on secure function evaluation (SFE), the trusted party computes $y = f(x_1, \ldots, x_n)$
Security Type

- **Computational**: a probabilistic polynomial time (PPT) distinguisher
  - The real & ideal worlds are *computationally* indistinguishable
- **Statistical**: all-powerful distinguisher, *negligible* error probability
  - The real & ideal worlds are *statistically* close
- **Perfect**: all-powerful distinguisher, *zero* error probability
  - The real & ideal worlds are *identically* distributed
Adversarial Model

• Adversarial behavior
  – Semi honest: honest-but-curious. corrupted parties follow the protocol honestly, $A$ tries to learn more information.
  – Malicious: corrupted parties can deviate from the protocol in an arbitrary way

• Adversarial power
  – Polynomial time: the adversary is allowed to run in (probabilistic) polynomial time (and sometimes, expected polynomial time), computational security
  – Computationally unbounded: the adversary has no computational limits whatsoever, information-theoretic security
Adversarial Model

- **Adversarial corruption**
  - **Static**: the set of corrupted parties is defined before the execution of the protocol begins. Honest parties are always honest, corrupted parties are always corrupted.
  - **Adaptive**: $\mathcal{A}$ can decide which parties to corrupt during the course of the protocol, based on information it dynamically learns.
  - **Mobile**: $\mathcal{A}$ can jump between parties. Honest parties can become corrupted, corrupted parties can become honest again.
Communication Model

- **Point-to-point**: fully connected network of pairwise channels.
- **Broadcast**: additional broadcast channel

**Message delivery:**
- **Synchronous**: the protocol proceeds in rounds. Every message that is sent arrives within a known time frame
- **Asynchronous (eventual delivery)**: the adversary can impose arbitrary (finite) delay on any message
- **Fully Asynchronous**: the adversary has full control over the network, can even drop messages
Execution Environment

• **Stand alone:**
  – A single protocol execution at any given time (isolated from the rest of the world)

• **Concurrent general composition:**
  – Arbitrary protocols are executed concurrently
  – An Internet-like setting
  – Requires a strictly stronger definition
  – Captured by the *universal composability (UC)* framework
The Stand-Alone Model

One set of parties executing a single protocol in isolation
The Concurrent Model

Many parties running many protocol executions
UC real model

Environment

Arbitrary interaction

write inputs/
read outputs

Protocol interaction
UC ideal model

- Trusted party
- Environment

Arbitrary interaction

write inputs/
read outputs
UC Security

Environment

Protocol interaction

Trusted party

REAL

IDEAL
Relaxing the Definition

- Recall the ideal world (with guaranteed output delivery)
  1) Each party sends its input to the trusted party
  2) The trusted party computes $y = f(x_1, \ldots, x_n)$
  3) Trusted party sends $y$ to each party
- This ideal world is overly ideal
- In general, fairness cannot be achieved without an honest majority
- A relaxed definition is normally considered
Security with Abort

- Ideal world without fairness and guaranteed output delivery:
  a. Each party sends its input to the trusted party
  b. The trusted party computes $y = f(x_1, \ldots, x_n)$
  c. Trusted party sends $y$ to the adversary
  d. The adversary responds with continue/abort
  e. If continue, trusted party sends $y$ to all parties; If abort, trusted party sends $\bot$ to all parties
  f. Correctness, privacy, independence of inputs are satisfied
Adversarial model

• In this lecture we consider:
  – Adversary: semi honest / malicious with static corruptions
  – Synchronous P2P network with a broadcast channel
  – Stand-alone setting
  – Probabilistic polynomial time (PPT) adversary & distinguisher
    (computational security)
Secure AND: $\Pi_{\text{AND}}$

Bob sends $a \land b$ to Alice
Alice and Bob both output $a \land b$
Functionality

Alice

\( a \rightarrow a \land b \)

Bob

\( b 
\rightarrow a \land b \)

F AND
• **Theorem.** $\Pi_{\text{AND}}$ is indistinguishable from $F_{\text{AND}}$ from the perspective of an semi-honest adversary.

• $\exists$ simulator $S_1$, s.t. $(\text{View}_{p_1,\text{real}}, \text{Output}_{p_1,\text{real}}) \approx (\text{View}_{p_1,\text{ideal}}, \text{Output}_{p_1,\text{ideal}})$

• $\exists$ simulator $S_2$, s.t. $(\text{View}_{p_2,\text{real}}, \text{Output}_{p_2,\text{real}}) \approx (\text{View}_{p_2,\text{ideal}}, \text{Output}_{p_2,\text{ideal}})$
Semi-honest vs Malicious

• Now to confuse you all...

• It is clear that any protocol that is secure in the presence of malicious adversaries is secure in the presence of semi-honest adversaries
  ◦ A malicious adversary is stronger, and can always behave semi-honestly…

• But, the simulator in the ideal model is also stronger
  ◦ It can change its input

• Does this make a difference?
A Protocol for Binary AND: $\Pi_{x \land y}$

• Input: $P_1$ has an input bit $x$ and $P_2$ has an input bit $y$.
• Output: The binary value $x \land y$ for $P_2$ only.

• The protocol:
  1. $P_1$ sends $P_2$ its input bit $x$.
  2. $P_2$ outputs the bit $x \land y$. 
Semi-honest vs Malicious

**Claim.** $\Pi_{x^y}$ securely computes the binary AND function in the presence of malicious adversaries.

**Claim.** $\Pi_{x^y}$ does not securely compute the binary AND function in the presence of semi-honest adversaries.
Semi-honest vs Malicious

- Fixing this absurdity
  - Allow a semi-honest adversary to also change its input
  - Arguably, this is legitimate (to choose input)
  - This is called augmented semi-honest

- Theorem:
  - Security for malicious adversaries implies security for augmented semi-honest adversaries
# Private set intersection (PSI)

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>p x o</td>
<td>s o n</td>
</tr>
<tr>
<td>n r e</td>
<td>i a y</td>
</tr>
<tr>
<td>s u m</td>
<td>w r u</td>
</tr>
</tbody>
</table>
Private set intersection (PSI)

Alice: p x o
      n r e
      s u m

Bob:  s o n
      i a y
      w r u
Private set intersection (PSI)

Alice

Bob

p x o

s o n

n r e

? ? ?

s u m

? r ?
Private set intersection (PSI)

Alice

Bob

n r ?
s ?

i a y

w r u
Private set intersection (PSI)

\{ \text{my phone contacts} \} \cap \{ \text{users of your service} \}
Private set intersection (PSI)

{ my phone contacts } ∩ { users of your service }
PSI on small sets (hundreds)

- private availability poll
- key agreement techniques
PSI on **small sets** (hundreds)
- private availability poll
- key agreement techniques

PSI on **large sets** (millions)
- double-registered voters
- OT extension; combinatorial tricks
PSI on **small sets** (hundreds)
- private availability poll
- key agreement techniques

PSI on **large sets** (millions)
- double-registered voters
- OT extension; combinatorial tricks

PSI on **asymmetric** sets (100 : billion)
- contact discovery; password checkup
- offline phase; leakage
PSI on **small sets** (hundreds)
- private availability poll
- key agreement techniques

PSI on **asymmetric** sets (100 : billion)
- contact discovery; password checkup
- offline phase; leakage

PSI on **large sets** (millions)
- double-registered voters
- OT extension; combinatorial tricks

Computing on the intersection
- sales statistics about intersection
- generic secure computation
Keyword Search

• **Input:**
  – Server: database \( X=\{(x_i,p_i)\} \), \( 1 \leq i \leq N \)
    - \( x_i \) is a keyword
    - \( p_i \) is the payload
  – Client: search word \( w \)

• **Output:**
  – Server: nothing
  – Client:
    - \( p_i \) if \( \exists i : x_i = w \)
    - otherwise nothing
Searchable Encryption

Alice

Encrypted data

Search keyword

Encrypted files which contains keyword

Server
Private Information Retrieval (PIR)

Let $x = x_1, x_2, \ldots, x_n \in \{0,1\}^n$. The user is interested in $x_i$ for some $i \in \{1, \ldots, n\}$.

Server: $x = x_1, x_2, \ldots, x_n \in \{0,1\}^n$

User: $i \in \{1, \ldots, n\}$
**k-Server PIR**

**Correctness:** User obtains $x_i$

**Privacy:** No *single* server gets information about $i$

$x \in \{0,1\}^n$

User

$S_1$

$S_2$

$S_k$
A machine is **oblivious** if its sequence of accessing (memory) locations is **indistinguishable** for any two inputs with the same length.

- **The server** cannot gain any information from the access pattern of **client**’s Read/Write requests.