Introduction to Multi-Party Computation and Applications

Mariana Gama

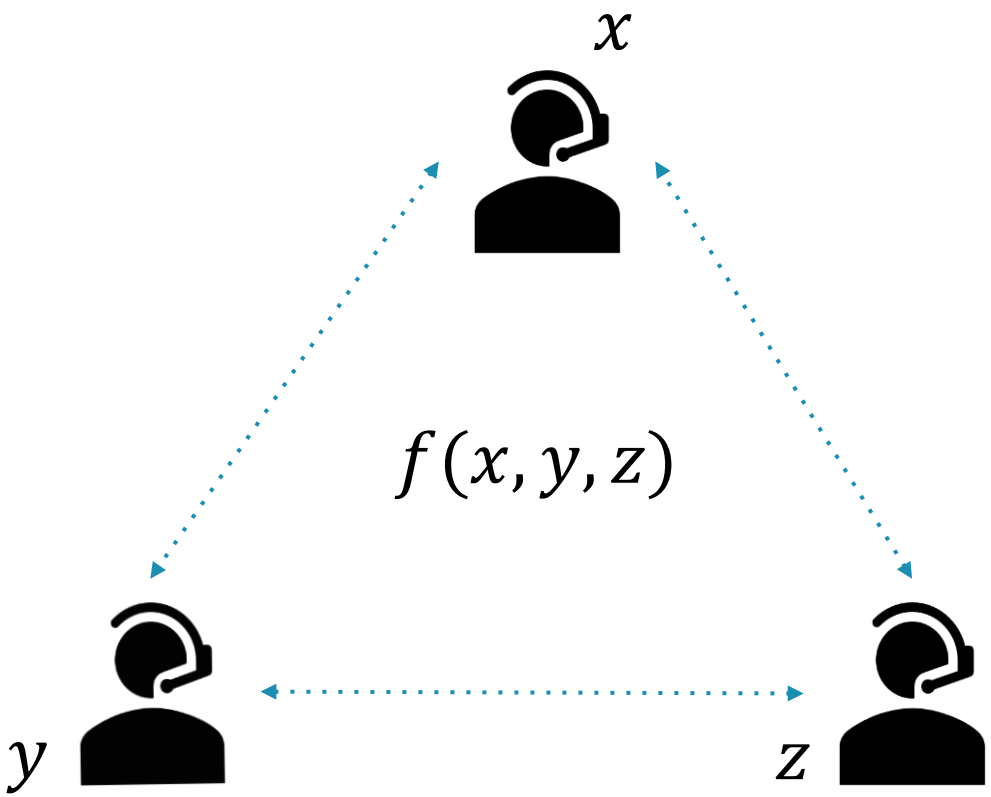








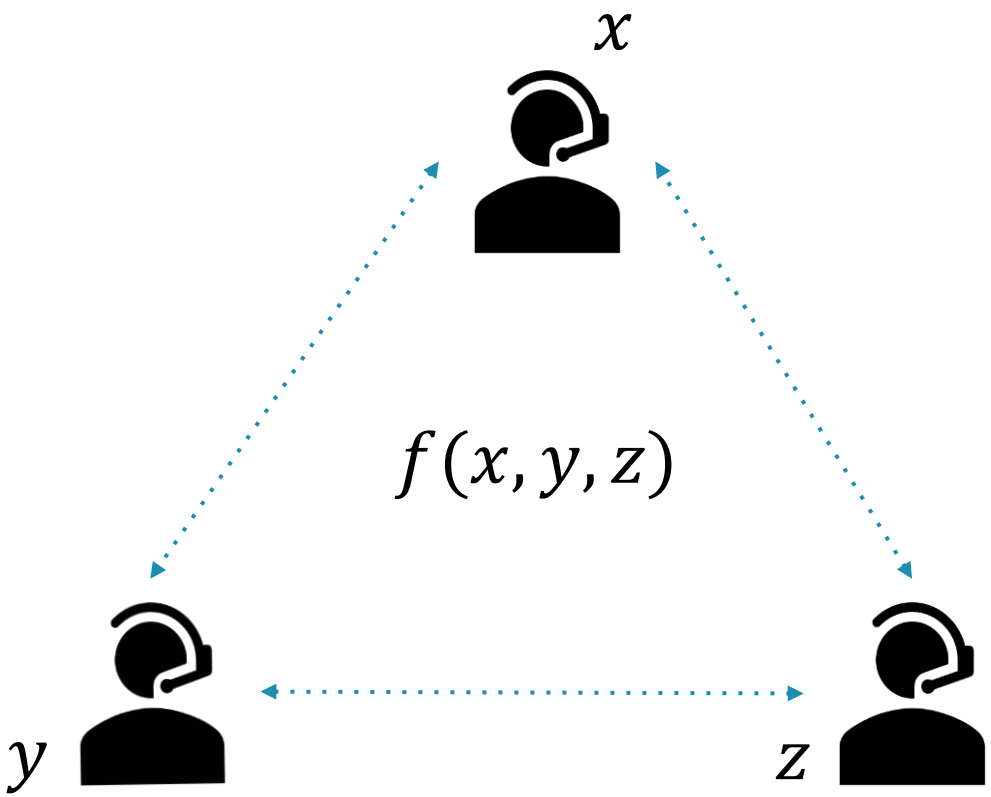
What is MPC?



- Cryptographic technique that allows **computation over encrypted data**.

- Parties can compute a **joint function** $f(x, y, z)$ such that the inputs x, y and z remain private.

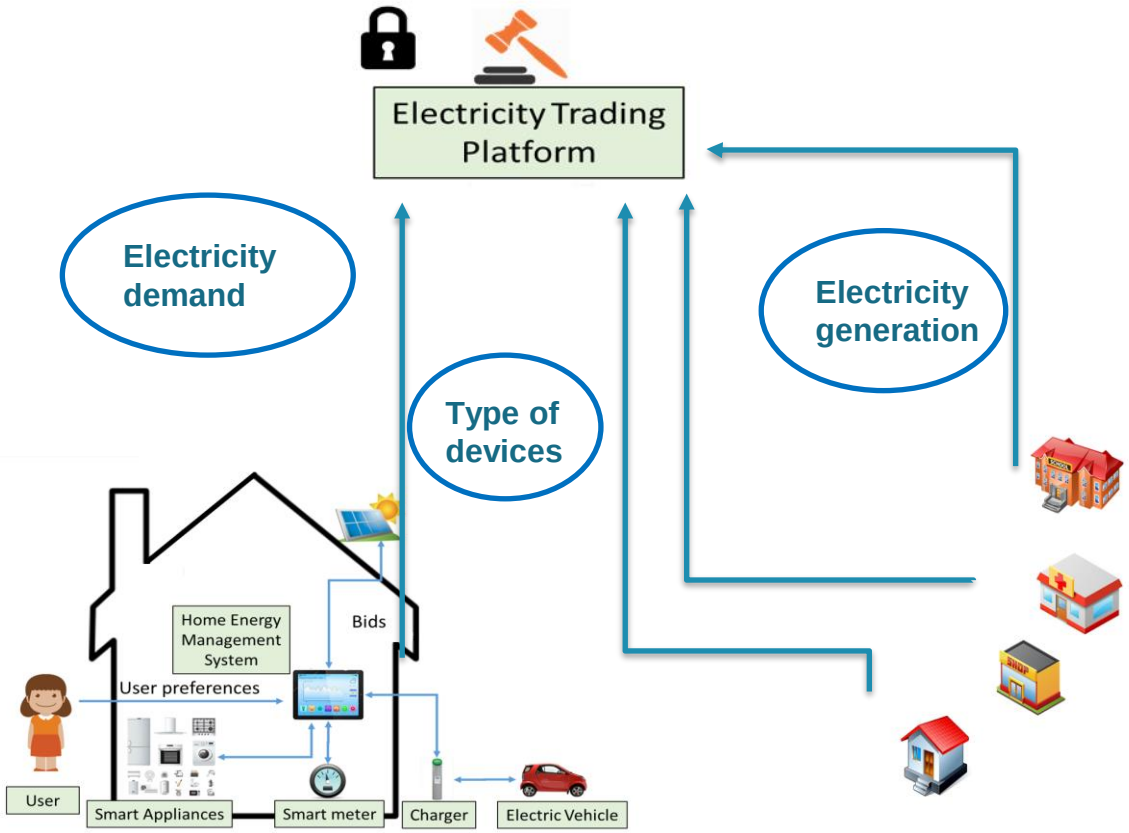
What is MPC?



- Cryptographic technique that allows **computation over encrypted data**.
  → similar to FHE, but faster in practice :D
  (however, we need multiple parties)

- Parties can compute a **joint function** $f(x, y, z)$ such that the inputs x, y and z remain private.

MPC applications



- Auctions
- Dark markets
- P2P electricity trading
- Fraud detection
- Surveys
- Data storage
- …

<u>Some MPC companies:</u>

- Partisia, Roseman Labs, Cybernetica, FragmentiX, Unbound Security, and many more

Secret sharing

secret input x



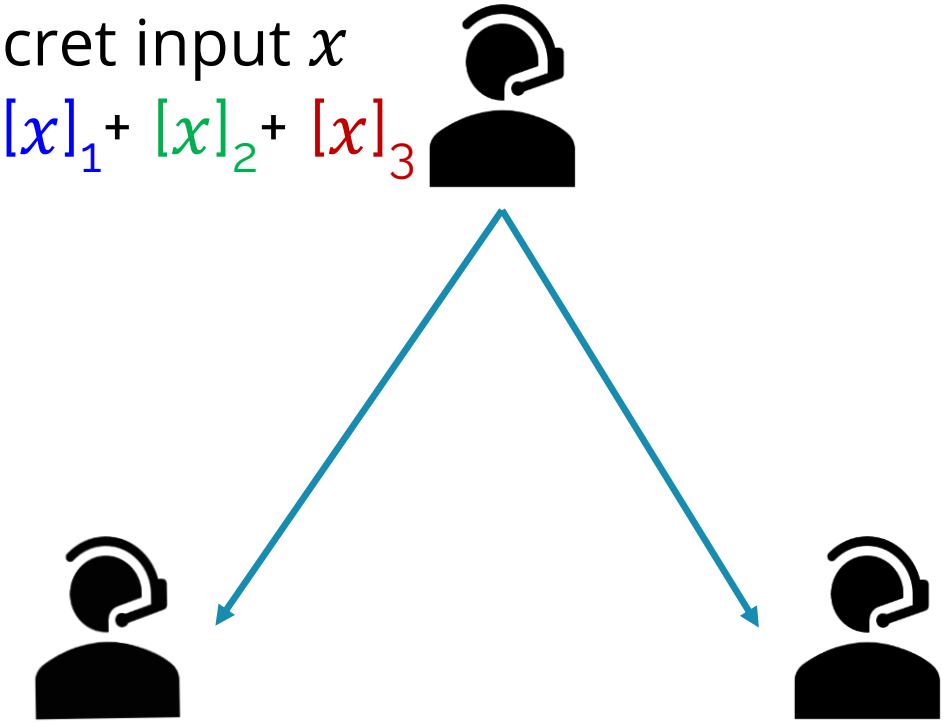




Secret sharing

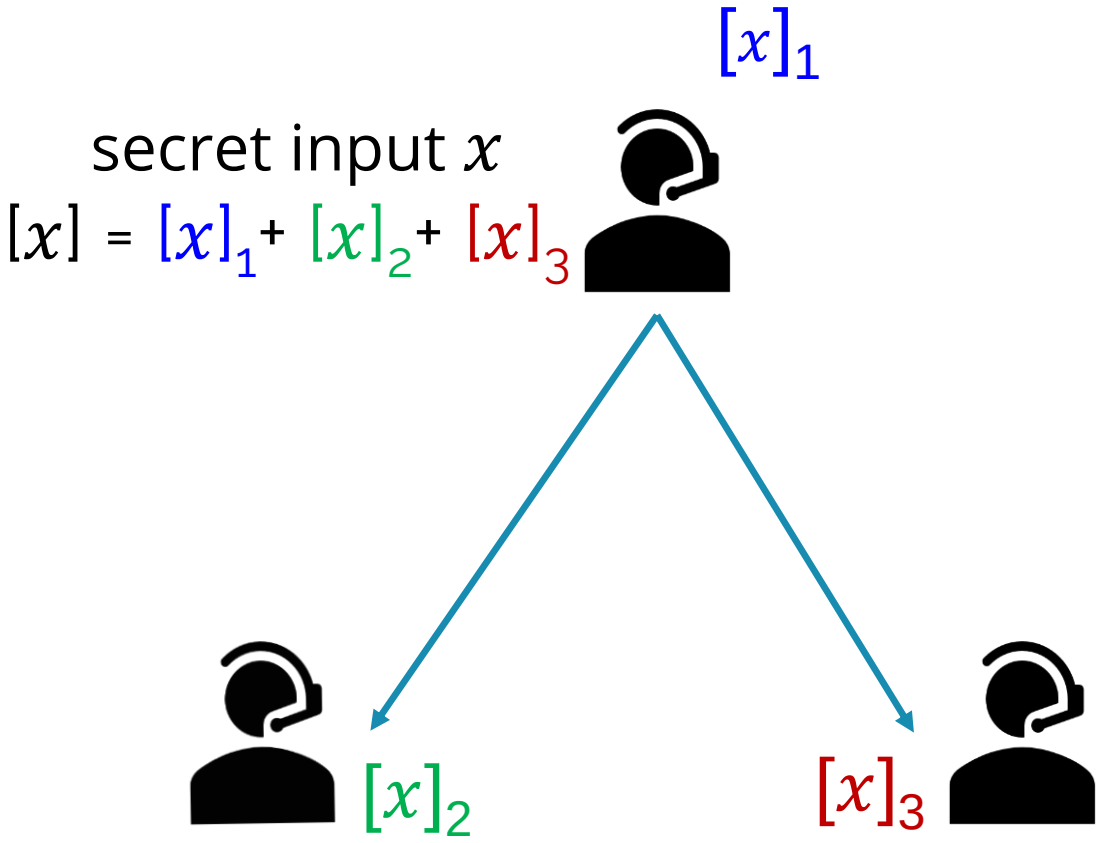
secret input x

$[x] = [x]_1 + [x]_2 + [x]_3$



- $[x]_1$, $[x]_2$ and $[x]_3$ are **secret shares**.

Secret sharing



- $[x]_1$, $[x]_2$ and $[x]_3$ are **secret shares**.
- Each secret share by itself looks random and reveals nothing about the private value.

Operations on secret shared values

Additive secret sharing

Secret value = share 1 + share 2 + share 3

$$[x] = [x]_1 + [x]_2 + [x]_3$$

Additions:

Multiplications:

Operations on secret shared values

Additive secret sharing

Secret value = share 1 + share 2 + share 3

$$[x] = [x]_1 + [x]_2 + [x]_3$$

Additions: $[a] + [b] = ([a]_1+[b]_1) + ([a]_2+[b]_2) + ([a]_3+[b]_3)$ 

Multiplications:

Operations on secret shared values

Additive secret sharing

Secret value = share 1 + share 2 + share 3

$$[x] = [x]_1 + [x]_2 + [x]_3$$

Additions: $[a] + [b] = ([a]_1+[b]_1) + ([a]_2+[b]_2) + ([a]_3+[b]_3)$ ✓

Multiplications: $[a] \cdot [b] = ([a]_1 \cdot [b]_1) + ([a]_2 \cdot [b]_2) + ([a]_3 \cdot [b]_3)$ ✗

Operations on secret shared values

Additive secret sharing

Secret value = share 1 + share 2 + share 3

$$[x] = [x]_1 + [x]_2 + [x]_3$$

Additions: $[a] + [b] = ([a]_1+[b]_1) + ([a]_2+[b]_2) + ([a]_3+[b]_3)$ ✓

Multiplications: $[a] \cdot [b] = ([a]_1 \cdot [b]_1) + ([a]_2 \cdot [b]_2) + ([a]_3 \cdot [b]_3)$ ✗

$$([a]_1+[a]_2+[a]_3) \cdot ([b]_1+[b]_2+[b]_3) = ([a]_1 \cdot [b]_1) + ([a]_2 \cdot [b]_2) + ([a]_3 \cdot [b]_3) + ([a]_1 \cdot [b]_2) + ([a]_1 \cdot [b]_3) + \ldots$$

Operations on secret shared values

Additive secret sharing

Secret value = share 1 + share 2 + share 3

$$[x] = [x]_1 + [x]_2 + [x]_3$$

Additions: $[a] + [b] = ([a]_1+[b]_1) + ([a]_2+[b]_2) + ([a]_3+[b]_3)$ ✓

Multiplications: $[a] \cdot [b] = ([a]_1 \cdot [b]_1) + ([a]_2 \cdot [b]_2) + ([a]_3 \cdot [b]_3)$ ✗

$$([a]_1+[a]_2+[a]_3) \cdot ([b]_1+[b]_2+[b]_3) = ([a]_1 \cdot [b]_1) + ([a]_2 \cdot [b]_2) + ([a]_3 \cdot [b]_3) + ([a]_1 \cdot [b]_2) + ([a]_1 \cdot [b]_3) + \dots$$

Need to communicate to calculate these terms!

Operations on secret shared values

Additive secret sharing

Secret value = share 1 + share 2 + share 3

$$[x] = [x]_1 + [x]_2 + [x]_3$$

Additions: $[a] + [b] = ([a]_1+[b]_1) + ([a]_2+[b]_2) + ([a]_3+[b]_3)$ ✓

Multiplications: $[a] \cdot [b] = ([a]_1 \cdot [b]_1) + ($

$([a]_1+[a]_2+[a]_3) \cdot ([b]_1+[b]_2+[b]_3) = ([a$

$[a] \cdot [b]$ – multiplication: **1 round** of communication

$[a] < [b]$ – comparison: **7 rounds** of communication

Using if-statements

Suppose the MPC parties have secret shared value [x], where x is a bit.
They want to do the following:

```
if [x] then
        [y] = [a]
else
        [y] = [b]
```

Using if-statements

Suppose the MPC parties have secret shared value [x], where x is a bit.
They want to do the following:

```
if [x] then
        [y] = [a]
else
        [y] = [b]
```

Both branches of the if-statement should be executed (or [x] will be revealed).
We then obliviously select between the branches.

Using if-statements

Suppose the MPC parties have secret shared value [x], where x is a bit.
They want to do the following:

```
if [x] then
        [y] = [a]
else
        [y] = [b]
```



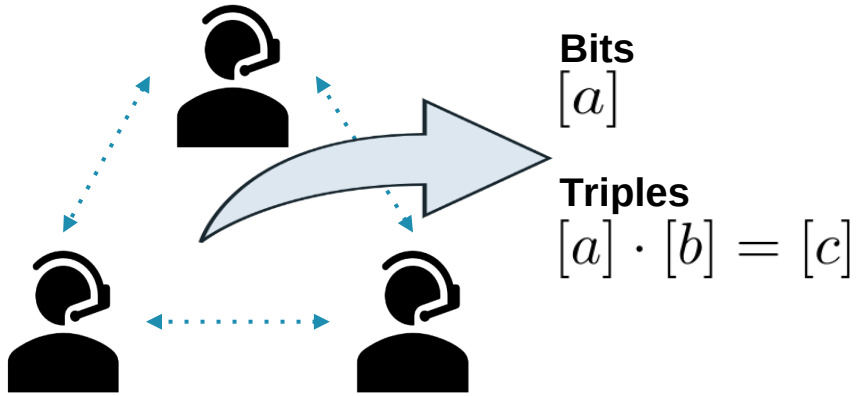
$$[y] = [a]\cdot[x] + [b]\cdot(1-[x])$$

Both branches of the if-statement should be executed (or [x] will be revealed).
We then obliviously select between the branches.

Computation phases

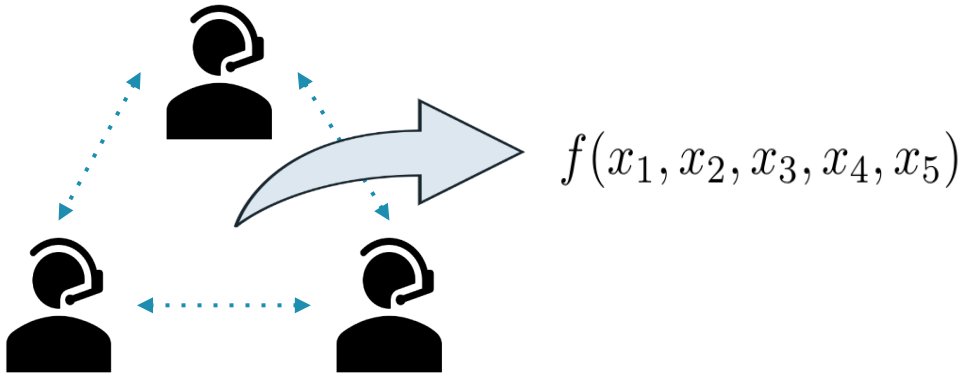
Offline phase

Independent of the inputs,
run while market is closed



Online phase

Utilises pre-calculated data



Adversaries and their power

Number of malicious parties:

- Honest majority
- Dishonest majority

Adversaries and their power

Number of malicious parties:

- Honest majority
- Dishonest majority

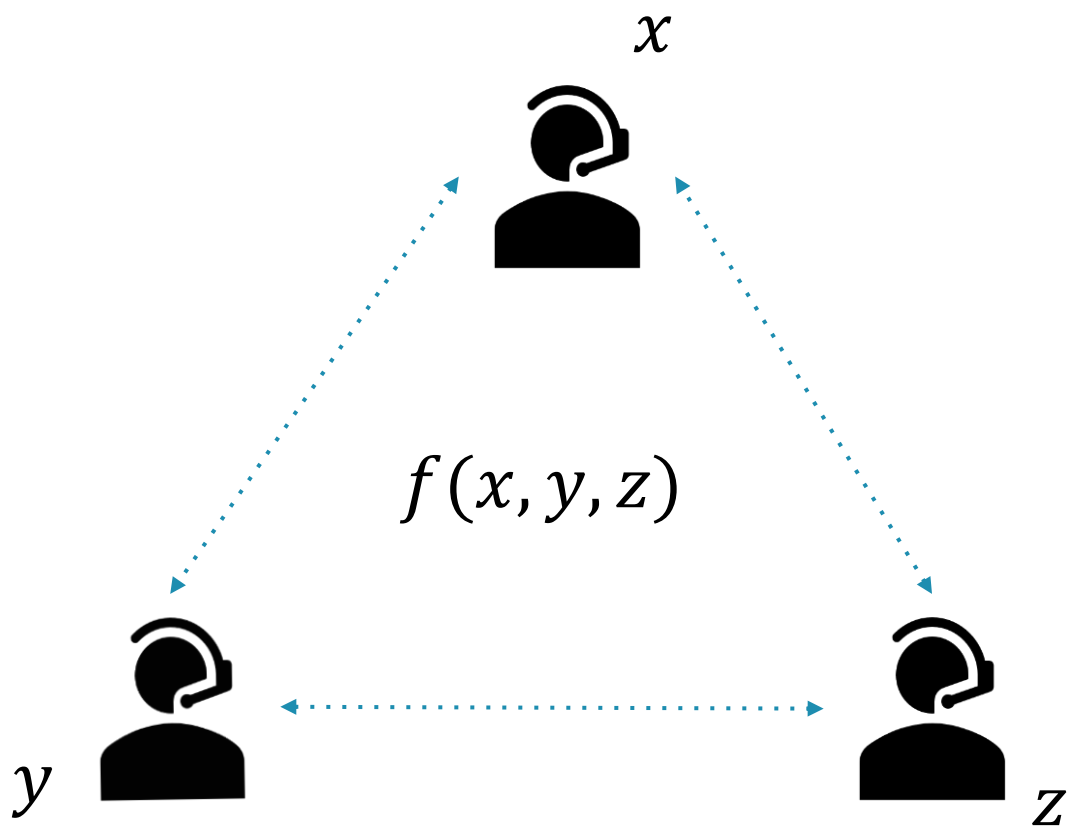
Type of malicious parties:

- Passive
- Active

Who are the MPC parties?

One possibility:

MPC parties are also the input providers.



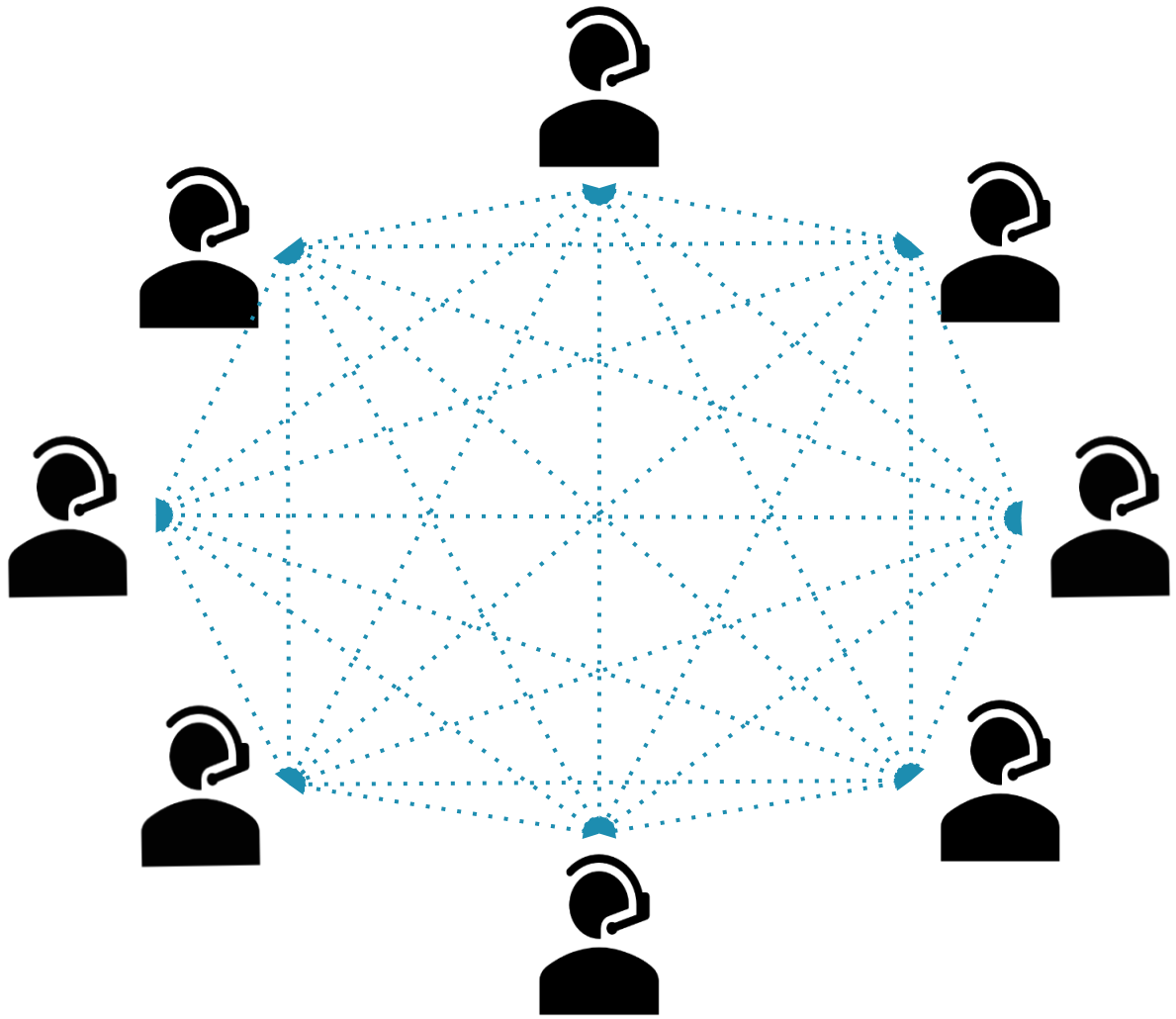
Who are the MPC parties?

One possibility:

MPC parties are also the input providers.

⬇

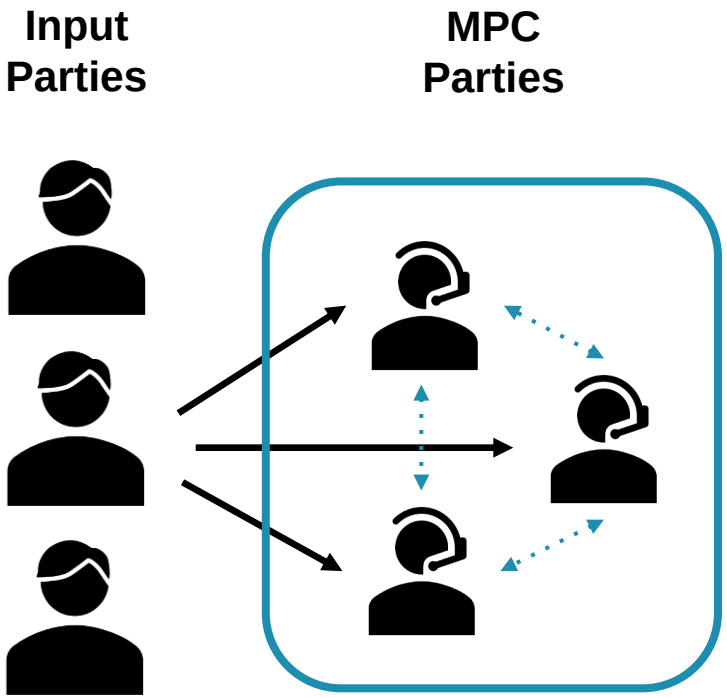
when there are many input providers, communication increases



Who are the MPC parties?

Another possibility:

MPC parties and the input providers are separate entities.

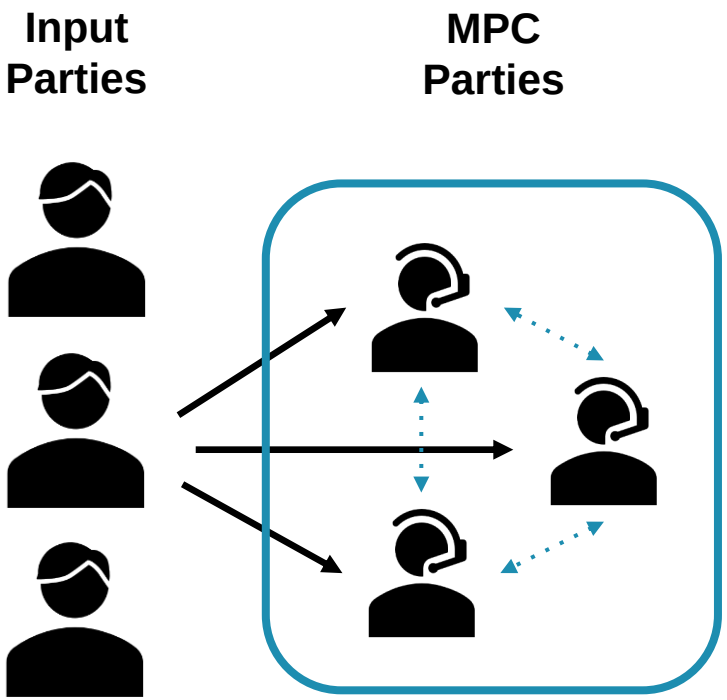


Who are the MPC parties?

Another possibility:

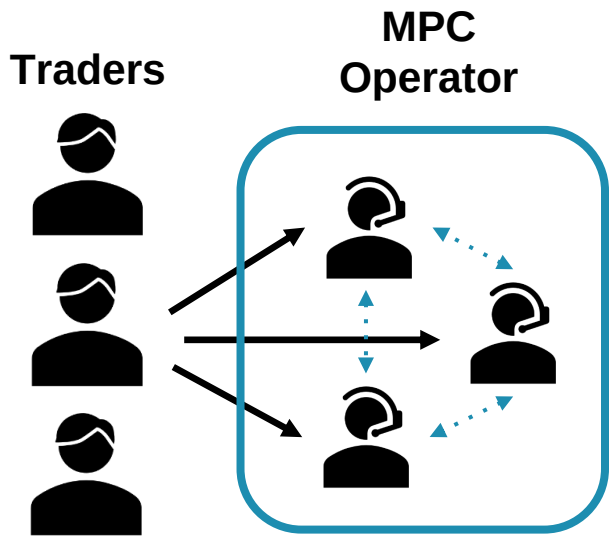
MPC parties and the input providers are separate entities.

Outputs can be revealed publicly, or only to specific individuals.



Dark pools with MPC

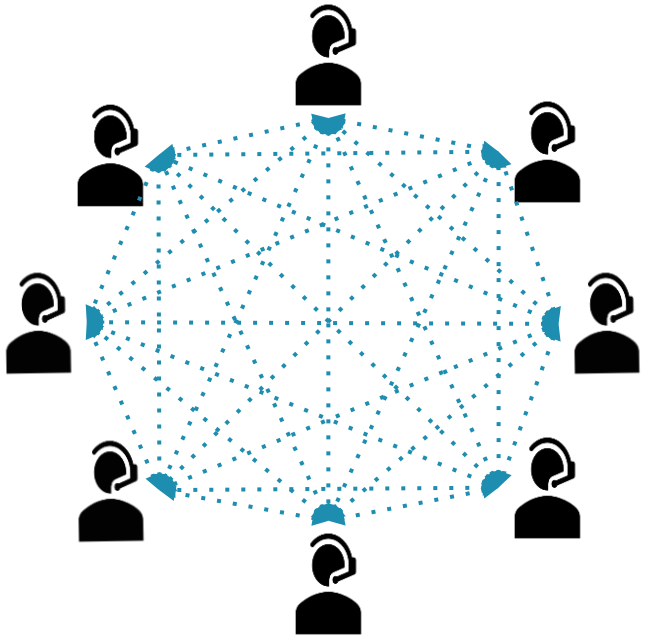
- The operator role is assigned to a set of entities (the MPC parties).
- The order contents will remain hidden as long as those entities do not collude.
- Only successful trades are revealed.



- **MPC joins the dark side [CST19]**
- **Kicking-the-bucket: fast privacy preserving trading using buckets [dGCP+22]**
- All for one and one for all: Fully decentralised privacy-preserving dark pool trading using multi-party computation [dGCSA22]

Dark pools with MPC

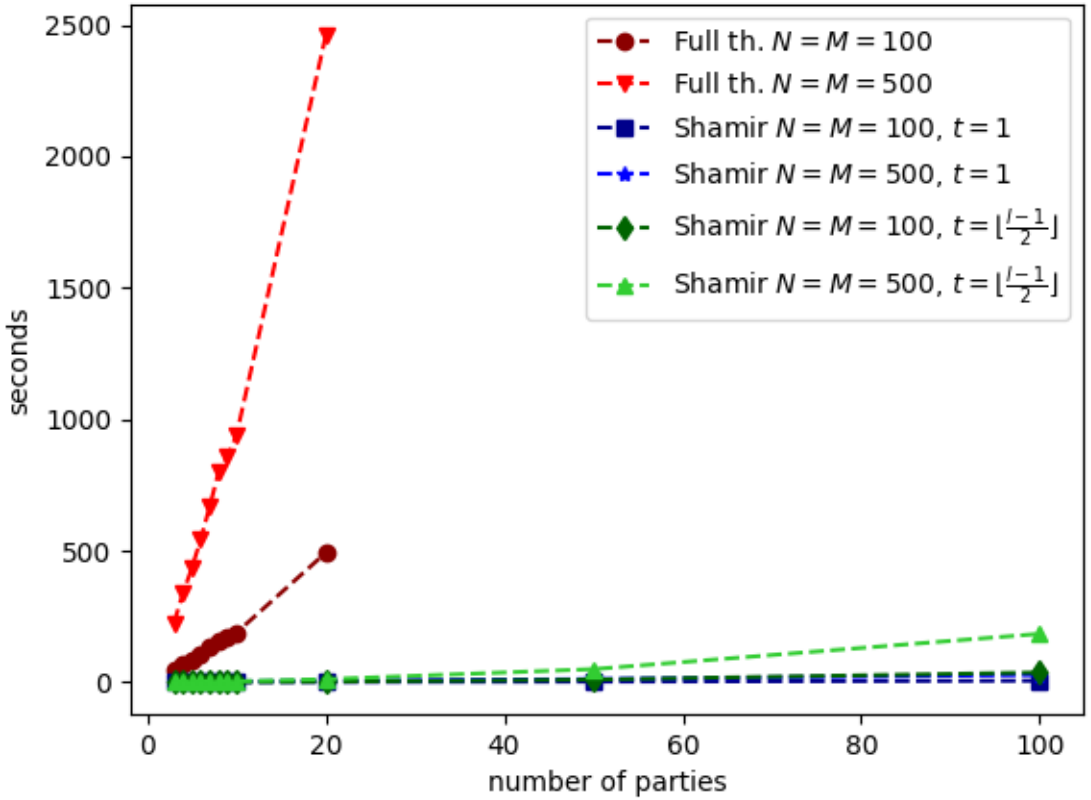
- The operator role is assigned to a set of entities (the MPC parties).
- The order contents will remain hidden as long as those entities do not collude.
- Only successful trades are revealed.



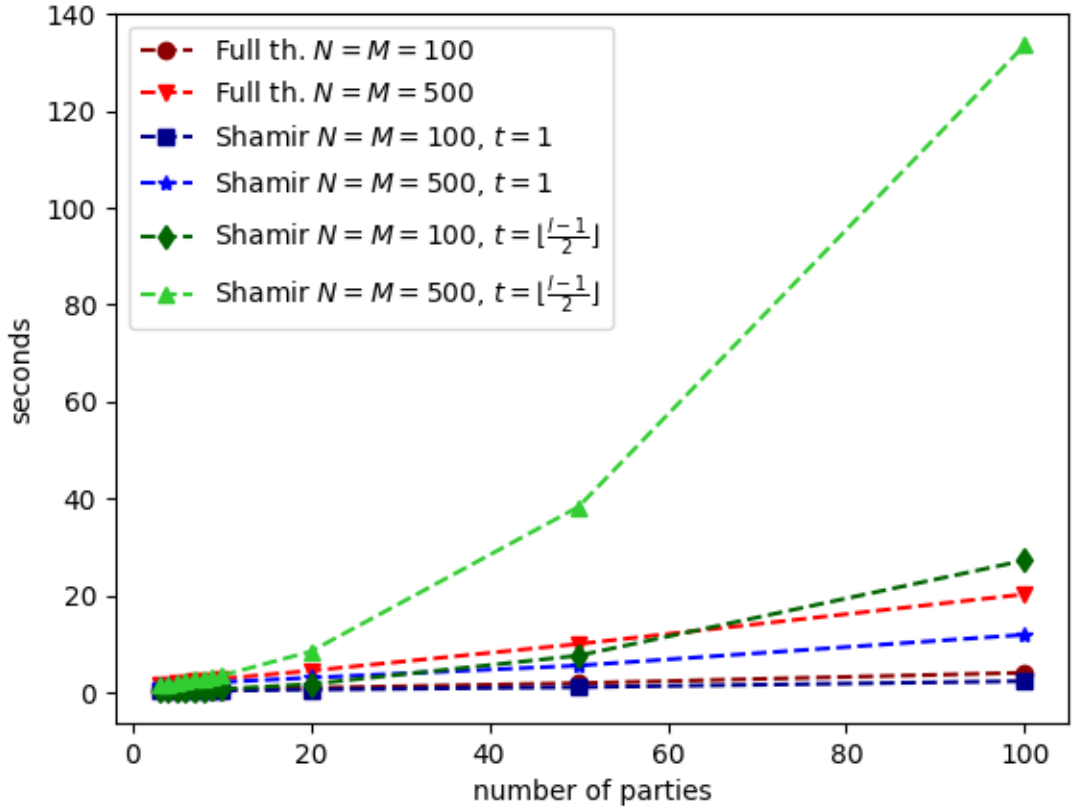
- MPC joins the dark side [CST19]
- Kicking-the-bucket: fast privacy preserving trading using buckets [dGCP+22]
- **All for one and one for all: Fully decentralised privacy-preserving dark pool trading using multi-party computation [dGCSA22]**

Dark pools with MPC
– some benchmarks with many parties

Offline phase



Online phase



MPC frameworks

- **SCALE-MAMBA**
- MP-SPDZ
- ABY
- EMP-toolkit
- Obliv-C
- …

SoK: General Purpose Compilers for Secure Multi-Party Computation

M Hastings, B Hemenway, D Noble, S Zdancewic

MPC frameworks

- **SCALE-MAMBA**
- MP-SPDZ
- ABY
- EMP-toolkit
- Obliv-C
- …



(Some) Data types:

sint: secret integers

cint: public integers

sfix: secret fixed point value

cfix: public fixed point value

sfloat: secret floating point value

cfloat: public floating point value

SoK: General Purpose Compilers for Secure Multi-Party Computation

M Hastings, B Hemenway, D Noble, S Zdancewic

MPC frameworks

- **SCALE-MAMBA** → Example: (the if-statement from before)
- MP-SPDZ
- ABY
- EMP-toolkit
- Obliv-C
- …

```
if [x] then
        [y] = [a]
else
        [y] = [b]
```

→ $[x] = [a]\cdot[x] + [b]\cdot(1-[x])$

SoK: General Purpose Compilers for Secure Multi-Party Computation

M Hastings, B Hemenway, D Noble, S Zdancewic

MPC frameworks

- **SCALE-MAMBA**
- MP-SPDZ
- ABY
- EMP-toolkit
- Obliv-C
- …

Example: (the if-statement from before)

if [x] then
    [y] = [a]
else
    [y] = [b]

→ [x] = [a]·[x] + [b]·(1-[x])

```
x = sint.get_private_input_from(1)  # get secret shared value from party 1
a = sint.get_private_input_from(2)  # get secret shared value from party 2
b = sint.get_private_input_from(3)  # get secret shared value from party 3

y = a*x + b*(1-x)  # compute both branches and select correct one

public_y = y.reveal()  # the reveal method returns a cint corresponding to y
```

SoK: General Purpose C
for Secure Multi-Party Co

M Hastings, B Hemenway, D Noble