University of Zagreb Faculty of Electrical Engineering and Computing | Technische Universität Wien | Technische Universität Berlin | RISE

AIoTwin

Twinning action for spreading excellence in Artificial Intelligence of Things

Privacy-Preserving Computation Techniques for Edge AI

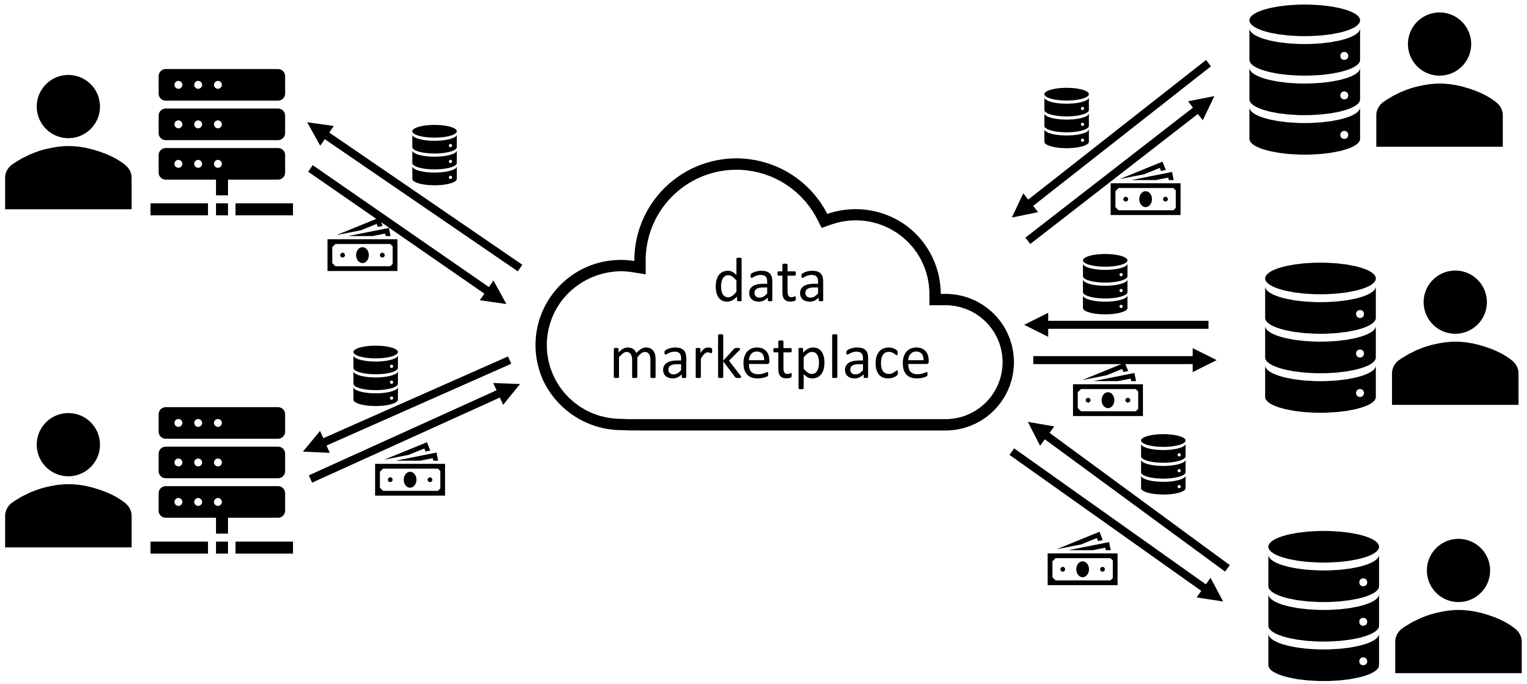
Lodovico Giaretta (RISE) - lodovico.giaretta@ri.se

2nd AIoTwin Summer School – 16-18.09.2024, Dubrovnik, Croatia

Funded by the European Union

Background: Data Marketplaces

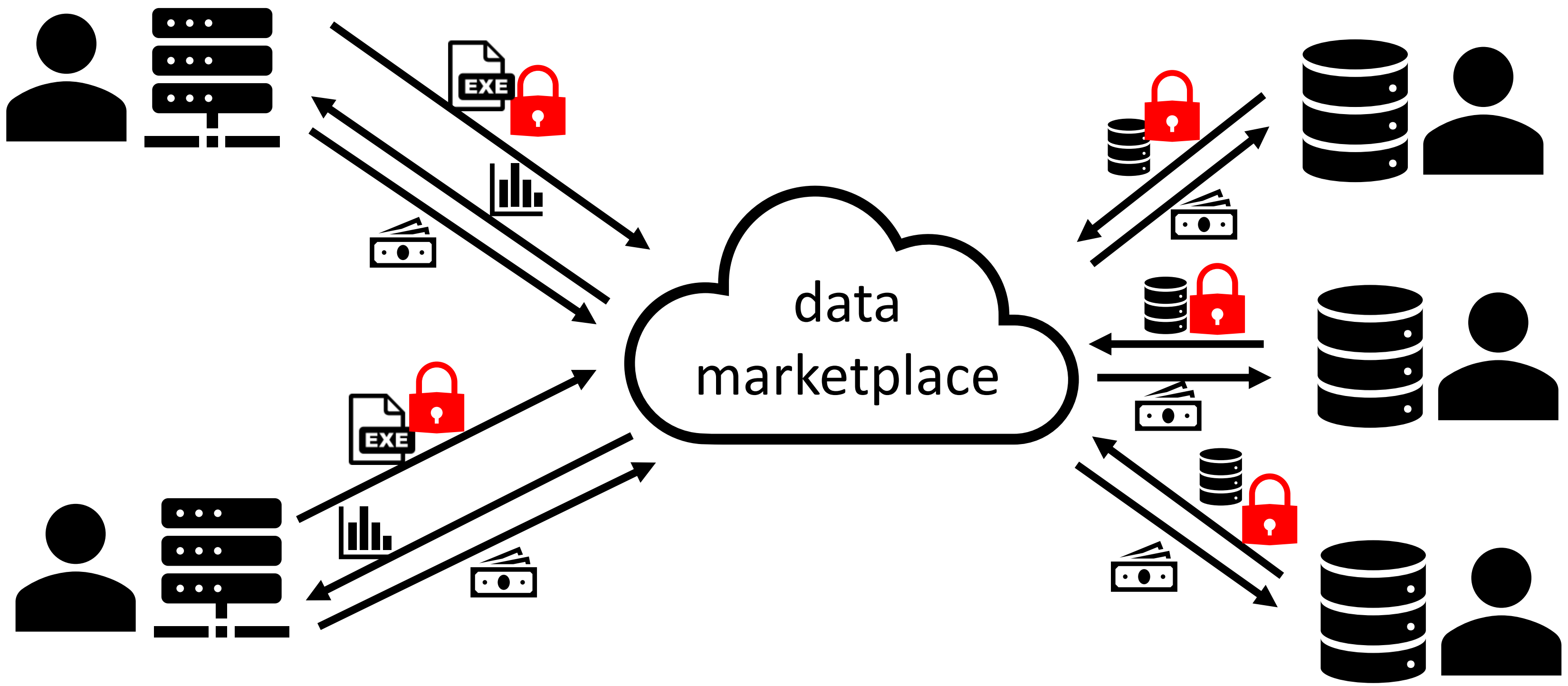
Typical data marketplace:



no data protection!

Background: Data Marketplaces

Our goal:



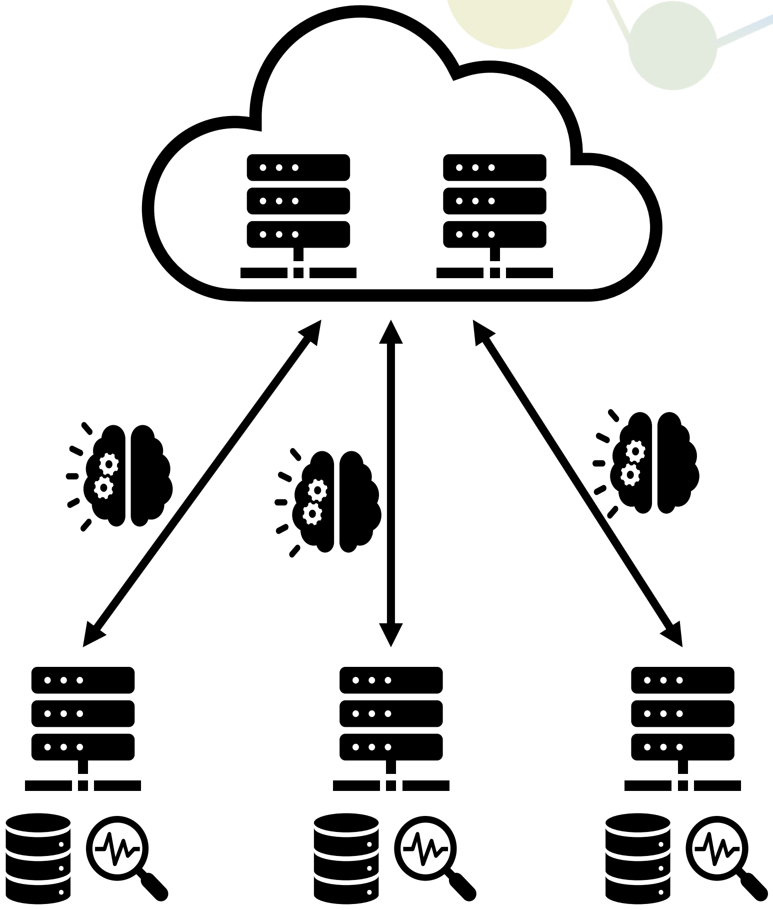
Privacy-Preserving Computation Techniques

- Secure Multi-Party Computation
- Fully-Homomorphic Encryption
- Trusted Execution Environments
- Differential Privacy

Scenario

“Typical” Federated Learning scenario

- Training a model on sensitive data from different data providers (sensors, users, hospitals, banks, …)
- Same applies to any ML/statistics/data aggregation

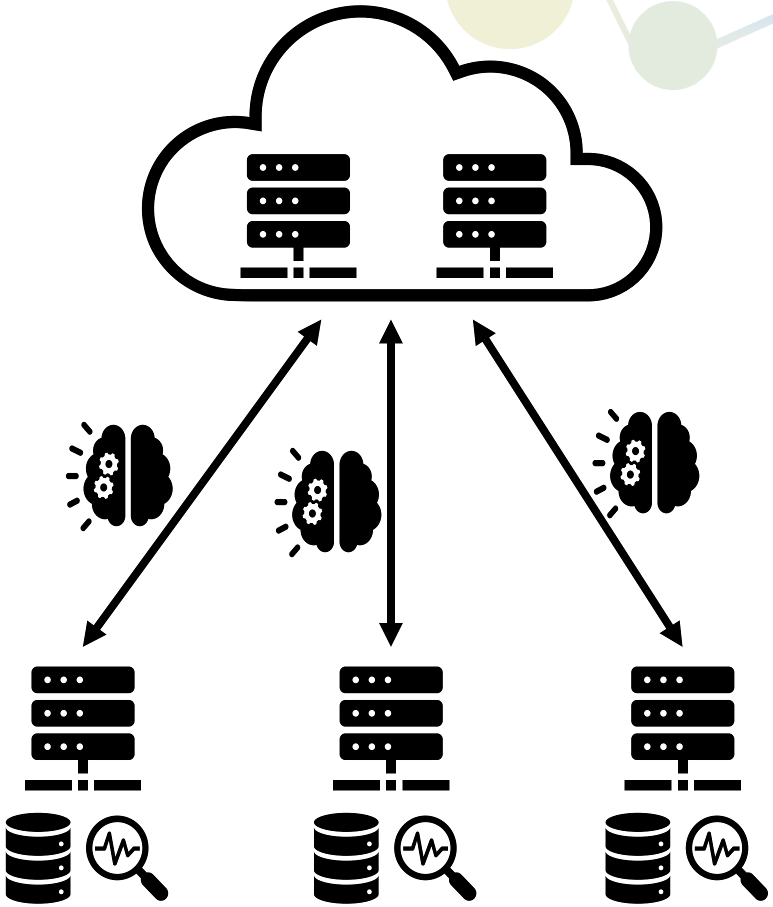


Communications must reveal local information!

Objective: prevent unauthorized usage of the data

Types of Privacy Leakage

- Training-time Leaks
- Inference-time Leaks

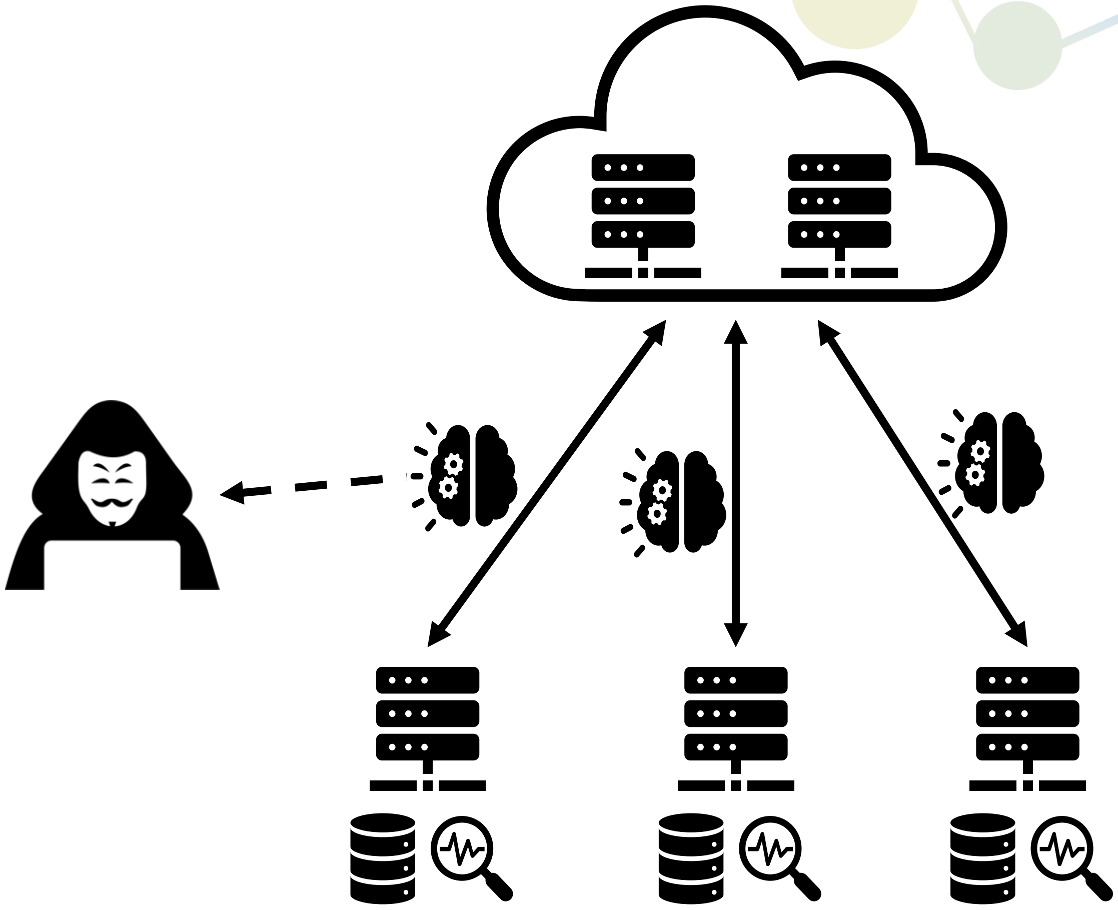


Training-Time Leaks

Training data features can be
reversed-engineered from gradients

By:

- Third-party interception

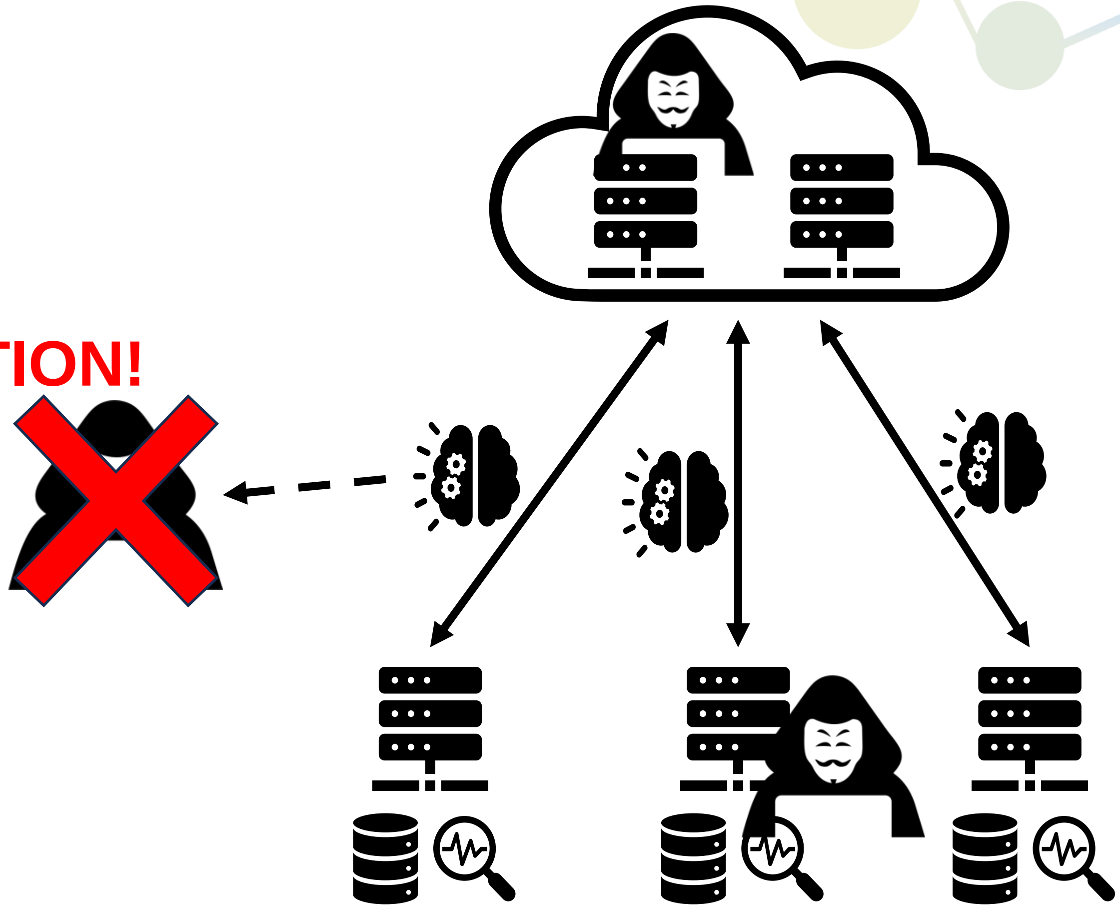


Training-Time Leaks

Training data features can be
reverse-engineered from gradients

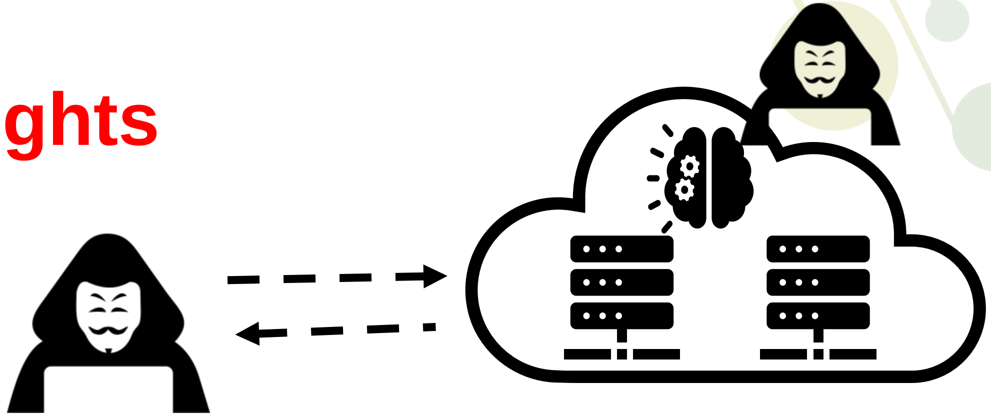
By:

- ~~Third-party interception~~ **USE ENCRYPTION!**
- Malicious participants
- Compromised participants

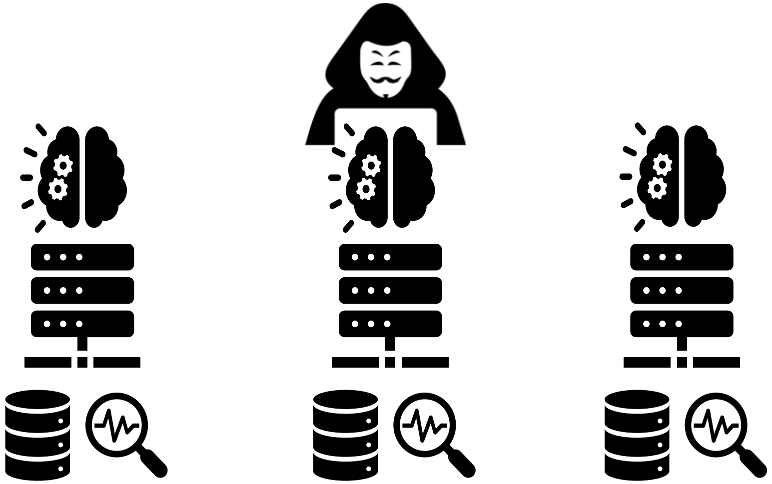


Inference-Time Leaks

- Training data features can be **reverse-engineered from model weights**
  - By participants
  - By anyone else with whom the model is shared



- Training data features can be **reverse-engineered from model I/O**
  - By any user of the model



Privacy-Preserving Computation Techniques

	Training-Time Leakage	Inference-Time Leakage
0. Encryption!!!	✓*	
1. Secure Multi-Party Computation	✓	
2. Fully-Homomorphic Encryption	✓	
3. Trusted Execution Environments	✓	
4. Differential Privacy	✓**	✓**

Secure Multi-Party Computation (SMC/MPC)

- Broad family of techniques
  - Peer-to-peer
  - Online, interactive

- **Basic concept:**
  - N players, each with their own datapoint x_i compute a function $f(x_1, \ldots, x_N)$ via peer-to-peer communications, without revealing any of the x_i

- Example time!

SMC: Secret Sharing

Most used approach for Secure Multi-Party Computation

- The secret is used to decode information in SMC
- Each of the N players owns a **unique share of the secret**
- At least **$1 \leq t \leq N$ shares are needed** to decode the information

- Can withstand up to $t - 1$ colluding players
- Can withstand up to $N - t$ dropouts

SMC: Pros/Cons

Pros:

- Peer-to-peer, designed for multiple players
- Simple to understand
- Acceptable overheads

Cons:

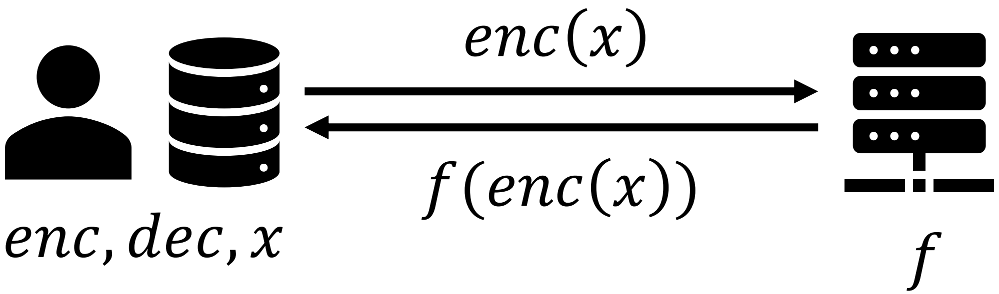
- Online, interactive
- Secure secret construction is hard
- Algorithm-specific

Fully-Homomorphic Encryption (FHE)

- **Homomorphic encryption:** a family of **encryption schemes** that supports certain operations on cyphertexts

$$f(x) = dec(f(enc(x)))$$

- **Fully-homomorphic encryption:** support for arbitrary sequences of operations



Fully-Homomorphic Encryption (FHE)

- **High overheads**
  - Large noise-tolerant encrypted representation
  - Each simple operation adds noise → payload must be decrypted every few steps
  - Treat decryption as an encrypted operation → unlimited steps hack!

- Not originally designed for FL
  - Mostly **designed for client/server scenarios**
  - Can be useful for privacy-preserving inference
  - Sometimes used as a component within SMC
  - Extensions for multiple clients do exist

FHE: Pros/Cons

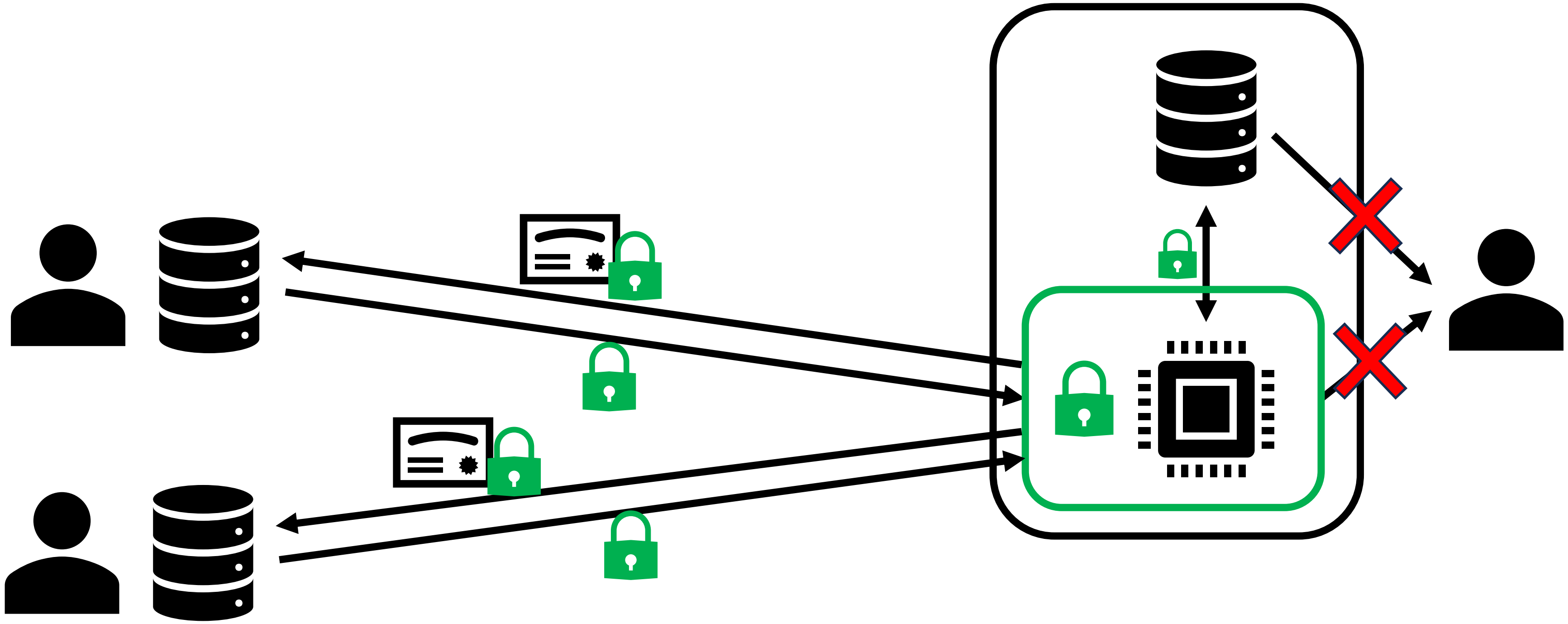
Pros:

- Simple to use, hard to get wrong
- Algorithm-agnostic

Cons:

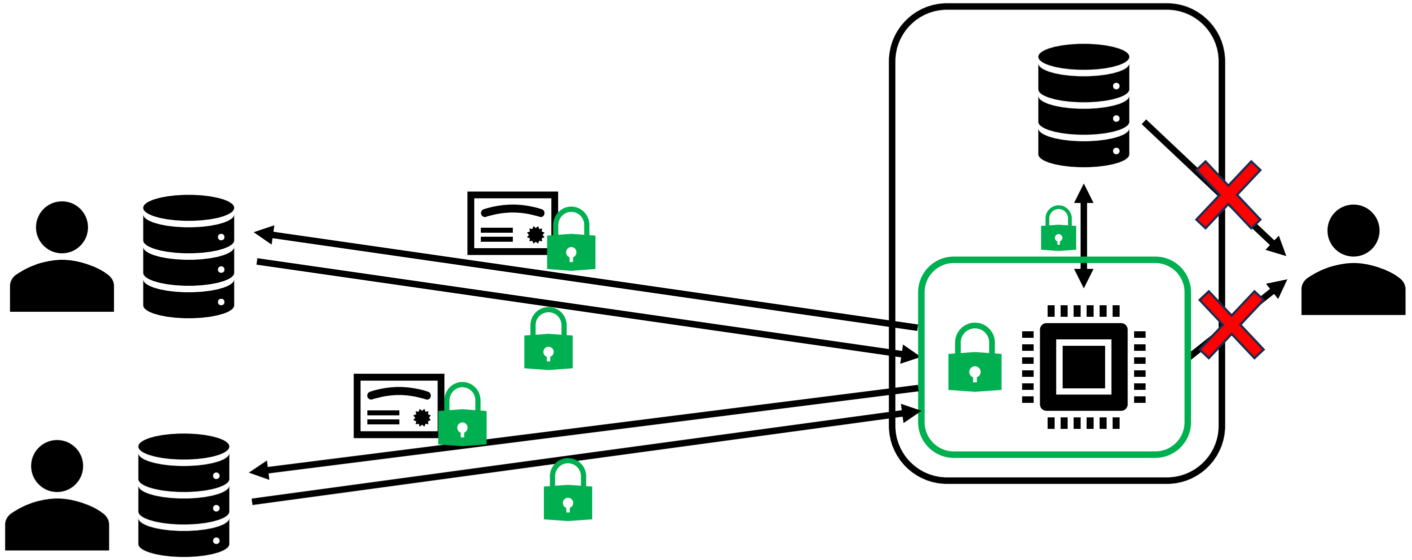
- Mostly designed for 2 players
- Huge overheads

Trusted Execution Environments (TEEs)



Trusted Execution Environments (TEEs)

- Built-in support in most CPUs (Intel SGX, Arm TrustZone, …)
- Kinda notorious for security issues
- Not automatically immune from side-channel attacks
- Recently: GPU TEEs for secure accelerated computing



TEEs: Pros/Cons

Pros:

- Conceptually simple
- Broadly applicable
- Hardware acceleration

Cons:

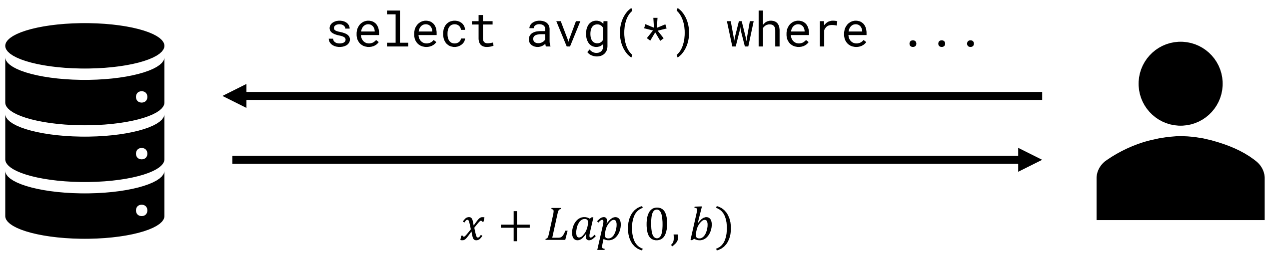
- Need to trust hardware manufacturer
- Tricky to defend against side channels

(Partial) Comparison

	scales well with number of nodes	scales well with problem complexity	is hardware-agnostic	is non-interactive
Secure Multi-Party Computation	✓✓	✓	✓✓	✗
Fully-Homomorphic Encryption	✓	✗	✓✓	✓✓
Trusted Execution Environments	✓✓	✓✓	✗	✓✓

Differential Privacy (DP)

Original goal: limit queries on sensitive data



Solution: add noise to hide the information

"**Privacy budget**" decreases with every overlapping query!

Differential Privacy: Noise Scaling

$$\tilde{Q}(D) = Q(D) + noise$$

(ε,δ)-DP:

$$\Pr\left(\tilde{Q}(D) \in O\right) \leq e^{\varepsilon} \Pr\left(\tilde{Q}(D') \in O\right) + \delta \qquad \forall\, D, D'$$

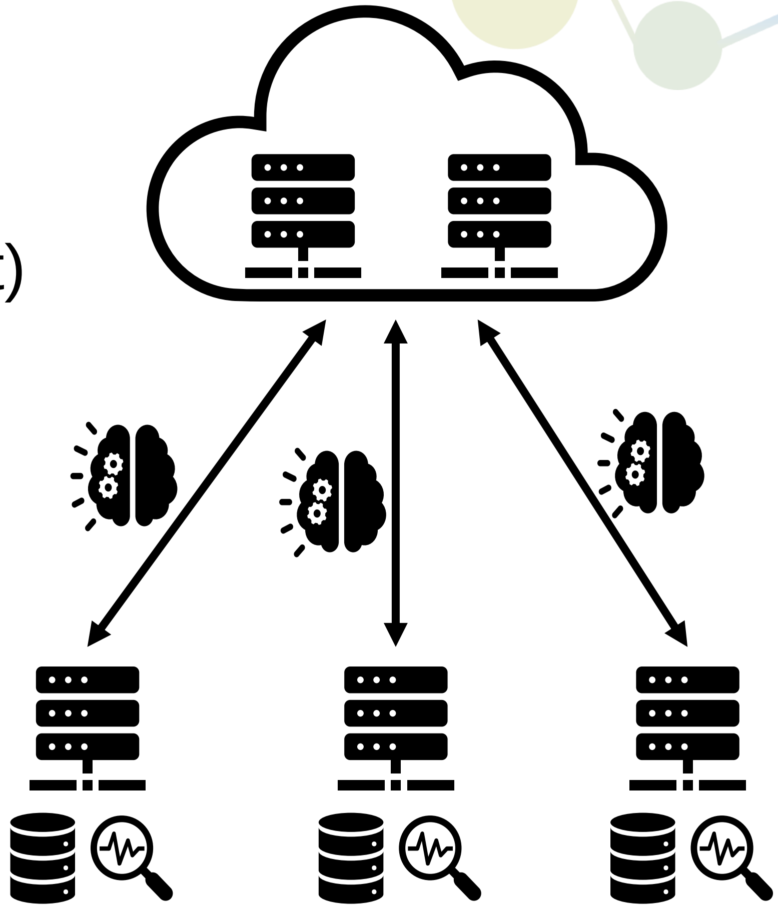
Where D, D' are **any two adjacent datasets** (i.e. that differ in one single entry)

- The bigger the potential difference made by a single entry, the bigger the noise scale
- The more queries needed, the bigger the noise

DP in FL: Two Levels

- **Device-level DP**
  - “Anonymizes” individual datapoints in each device
  - Does not “anonymize” the participating devices
  - Useful for e.g. hospital datasets (one row per patient)
  - No need to trust the aggregator

- **Aggregator-level DP**
  - “Anonymizes” whole participating devices
  - Useful for e.g. smart devices (one device per user)
  - Need to trust the aggregator



DP: Pros/Cons

Pros:

- Do you even have a choice?
- Strong mathematical guarantees

Cons:

- Noise vs utility tradeoff
- Limited number of queries

Conclusion

- **4 key techniques** for privacy-preserving computing
  - Secure Multi-Party Computation
  - Fully-Homomorphic Encryption
  - Trusted Execution Environments
  - Differential Privacy

- **No one-size-fits-all**
  - Must consider pros/cons in the context of a specific application

- **Encrypt and authenticate** everything end-to-end!

Tomorrow's Hands-on: ColonyOS

Necessary preparatory steps

Preparation (1/2): Download requirements

- Install `docker` and `docker-compose` on your machine:
  - Linux: use your distribution's package manager
  - Windows/Mac: we suggest using Docker Desktop (free for non-commercial use)

- Download the ColonyOS files:
  - The environment variables file:
    - Windows: https://raw.githubusercontent.com/colonyos/colonies/main/windowsenv.bat
    - Linux: https://raw.githubusercontent.com/colonyos/colonies/main/docker-compose.env
  - The docker-compose file: https://raw.githubusercontent.com/colonyos/colonies/main/docker-compose.yml
  - The ColonyOS CLI tool:
    - Binaries for all platforms: https://github.com/colonyos/colonies/releases

Preparation (2/2): Test Everything

- Before any other step, always set the environment variables
  - Windows: `windowsenv.bat`
  - Linux: `source docker-compose.env`
- On one terminal, start the virtual ColonyOS environment
  - Set the environment variables (as described above)
  - Run `docker-compose up` (or `docker compose up` depending on version)
- On another terminal, connect to the ColonyOS environment using the CLI tool
  - Set the environment variables (as described above)
  - Run `colonies executor ls`
  - Expected output:

NAME	TYPE	LOCATION	LAST HEARD FROM
dev-docker	container-executor	n/a	2024-06-29 13:37:27

- To shutdown the virtual ColonyOS environment:
  - `docker-compose down` (or `docker compose down` depending on version)

Links

These slides

ColonyOS setup/tutorial

https://github.com/colonyos/tutorials