The future of privacy-preserving computation under Gaia-X

03/05/2023

Eneko Gómez, Tecnalia

Borja Urkizu, Tecnalia

Agenda

01

Introduction: Privacy-preserving in the context of Gaia-X

Privacy-Preserving Computation (PPC)

Conventional data privacy cryptographic procedures

- Data in transit: e.g., VPN, SSL...
- Data at rest: e.g., hard disk encryption...

What about data in use?

- Regular operation
  - Sequence
    1. Decrypt
    2. Compute
    3. Encrypt result
  - Data exposed during computation
  - Unacceptable for sensitive data when computation is performed by a 3rd party (other than the data owner)
- **Privacy-preserving computation**: Advanced cryptographic techniques that allow performing computations on data without access to the raw cleartext data (or even the raw cleartext result)

Privacy-preserving computation

2 main techniques

- Multi-Party Computation (MPC)
  - Data splitted into secret shares at sources
  - Multi-party distributed computing
  - The result is joint for the authorized parties
- Homomorphic Encryption (HE)
  - Data encrypted according a homomorphic context
  - Encrypted data is computed (directly, without being decrypted)
  - The output of the computation is the encrypted result
  - The private key owner decrypts the result

Privacy-preserving in the context of Gaia-X

Gaia-X enables and boosts the creation of data spaces through trusted platforms that comply with common rules, allowing users and providers to trust each other on an objective technological basis, to safely and freely share and exchange data across multiple actors.

But, what about the level of confidence? Trust is not binary.

- Data owners may need to infer information from its source data through the services from providers, without disclosing their sensitive data.
- 3rd parties may need aggregated information from data owners, while having no interest in disclosing their sensitive source data (e.g., statistics, AI model training, etc.). This may apply not only to 3rd parties, but also to consortiums with a common goal, but private sensitive data.

So, what if we wish to take advantage of these data spaces through computation while not disclosing sensitive data? This is where privacy-preserving computation comes into play.

02

Multi-Party Computation

Multi-Party Computation (MPC)

Cryptographic tool that enables multiple parties to jointly perform a computation on their combined data, without disclosing their individual private inputs.

Main properties

- Correctness: The output produced by an algorithm is correct
- Privacy: The secret input of each party cannot be inferred by other parties

Garbled Circuits Protocol

- 1st 2-Party MPC Protocol
- Andrew Yao, 1982
- Millionaire's Problem: Two millionaires having lunch decide that the richest one should pay the bill; however, neither wants to reveal how much money he has

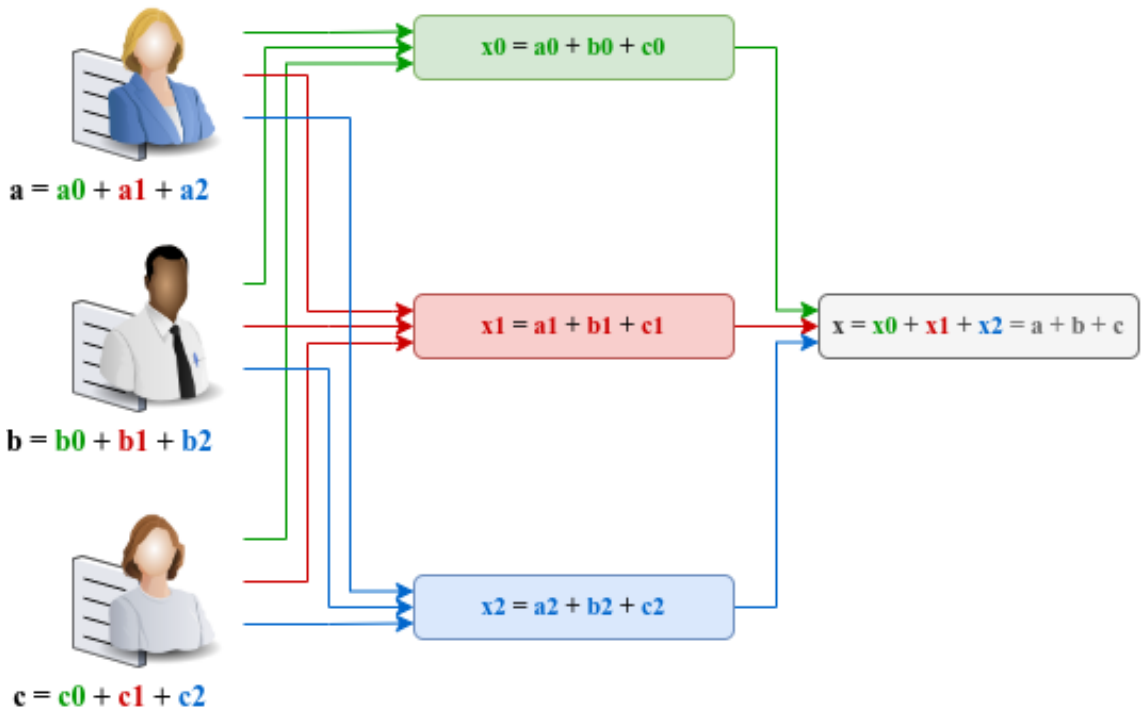
MPC

Simplified process

1. Split input data into pieces (secret shares)
2. Operate over secret shares
3. Recombine secret shares to get result

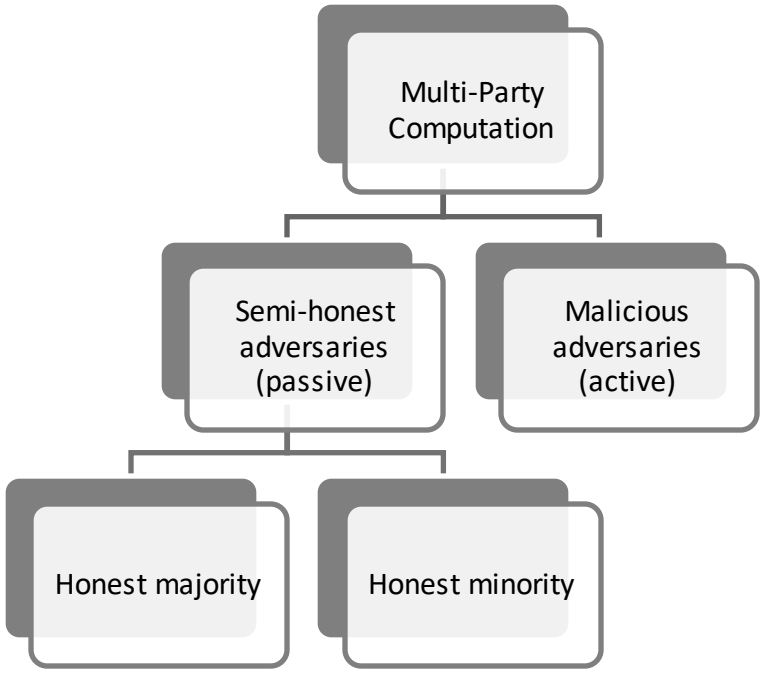
In practice

- More complex
- Involves more communication rounds



Registration of participants → Generation of secret shares → Distribution of secret shares → Computation → Collection of secret results → Reconstruction

MPC



Classification of MPC protocols according to their security models

- *Semi-honest*: Run the protocol honestly, but try to learn from other parties
- *Malicious*: Attempt to alter the execution of the protocol

MP-SPDZ

Most mature open-source software to run multiple secure MPC protocols supporting a variety of security models.

- Source code
  - https://github.com/data61/MP-SPDZ
  - Forked from https://github.com/bristolcrypto/SPDZ-2
  - Code:
    - Python: Algorithm compilation
    - C++: Protocol execution
  - License: BSD-3
- Documentation: https://mp-spdz.readthedocs.io/
- Machine learning features
- MPC Protocols:

Security model	Mod prime / GF(2^n)	Mod 2^k	Bin. SS	Garbling
Malicious, dishonest majority	MASCOT / LowGear / HighGear	SPDZ2k	Tiny / Tinier	BMR
Covert, dishonest majority	CowGear / ChaiGear	N/A	N/A	N/A
Semi-honest, dishonest majority	Semi / Hemi / Temi / Soho	Semi2k	SemiBin	Yao's GC / BMR
Malicious, honest majority	Shamir / Rep3 / PS / SY	Brain / Rep3 / PS / SY	Rep3 / CCD / PS	BMR
Semi-honest, honest majority	Shamir / ATLAS / Rep3	Rep3	Rep3 / CCD	BMR
Malicious, honest supermajority	Rep4	Rep4	Rep4	N/A
Semi-honest, dealer	Dealer	Dealer	Dealer	N/A

MP-SPDZ

- Shortcoming: Does not provide orchestration
  - You must know the other parties
    - Connection: Internet/VPN
    - Endpoint: IP/domain, port
    - Security: Certificate distribution
  - The algorithm developer must distribute the algorithm to the other parties
  - You must be aware of the MPC protocol to run
  - You must be aware of the nature of the required data to provide
    - Integer, fixed point, floating point…
    - Number of samples
    - Nature of the data itself (what does the data represent?)
    - Filters
  - Synchronization: All parties must start the execution within a constrained time window
- MP-SPDZ covers the compilation and execution of the algorithm, but is utterly unaware of the whole ecosystem involving participants and data

MPC Language

- Python subset
- https://mp-spdz.readthedocs.io/en/latest/Compiler.html
- Basic types (running time)
  - sint: Secret integer (in the protocol-specific domain)
  - cint: Clear integer (in the protocol-specific domain)
  - regint: Clear 64-bit integer
  - sfix: Secret fixed-point number (represented as secret integer)
  - cfix: Clear fixed-point number (represented as clear integer)
  - sfloat: Secret floating-point number
- Container types (running time)
  - MemValue: Single value in memory
  - Array: Array accessible by public index
  - Matrix: Matrix
  - MultiArray: Multidimensional array
- Decorators for loops on custom running time clear integers
  - `range(my_int)` -> `@for_range(my_sint)`

MPC Algorithm Example

```
N_players = 3
N_samples = 4

sum = sfix.Array(N_samples)
sum.assign_all(0)

# Conversion not required, except for demonstrating decorators
n_players = cint(N_players)
@for_range(n_players)
def _(i):
    # Loop not required (Array overrides sum operator), except for comparing loops
    for j in range(N_samples):
        sum[j] += sfix.get_input_from(i)

avg = sum / sfix(N_players)

avg_reveal0 = avg.reveal_to(0)
avg_reveal0.binary_output()

print_ln_to(0, '%s', avg_reveal0)
```

MPC Demo

- 3 MP-SPDZ parties running as docker containers on the same PC
- Steps
  1. Deploy parties (suitably configured to interact between them)
  2. Develop algorithm
  3. Compile algorithm
  4. Run the algorithm
  5. Check the result (on revealing parties)
- Objectives
  - Check the technology
  - Arrange a development environment where to compile and test the algorithms, before distributing them to third parties (who will be the ones who will really add value with their private data, while maintaining the data secret)
- https://github.com/tecnaliacybersec/gaiax-techx-mpc

MPC Demo

- Docker image: tecnaliacybersec/mp-spdz:latest
  - Custom MP-SPDZ docker image
  - All supported protocols compiled
  - Container keeps up running (until explicitly stopped)
- Px-Data/ -> Player-Data/
  - 'hosts' networking file: Docker domains + ports
  - 'Px.key: Own private key (for TLS communication)
  - 'Px.pem': Each party's certificate (for TLS communication)
  - '*.0': Each certificate's hashed link

    ```
    c_rehash Player-Data
    ```
  - Output files
- Programs/
  - 'Source': Algorithms' source code
  - 'Bytecode': Compiled byte codes
  - 'Schedules' : Compiled algorithms' metadata

MPC Demo

- 1. Check material
  - Algorithm: `docker/Programs/Source/test.mpc`
  - Networking file: `docker/Px-Data/hosts`
  - Input files: `docker/Px-Data/Input-Px-0`
- 2. Launch docker containers

```
cd docker
docker-compose -p mpc up –d
docker ps
```

- 3. Compile algorithm (from player 0)

```
docker exec -it node0.mpc ./compile.py test 3 4
```

- 4. Run algorithm (all players, Shamir Secret Sharing protocol)

```
docker exec -it node0.mpc ./shamir-party.x --help
docker exec -it node0.mpc ./shamir-party.x 0 test-3-4 -N 3 -ip ./Player-Data/hosts
docker exec -it node1.mpc ./shamir-party.x 1 test-3-4 -N 3 -ip ./Player-Data/hosts
docker exec -it node2.mpc ./shamir-party.x 2 test-3-4 -N 3 -ip ./Player-Data/hosts
```

- 5. Check output

Exercise: Variance

$$\mathrm{Var}(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

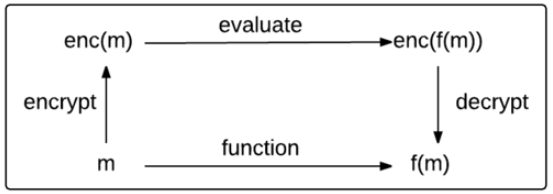
- Extend the example to calculate the variance
- Hints:
  - Same input cannot be captured twice
    - Save into Matrix as reading from input when calculating the average
  - Most operations can be performed without looping
    - 2 N-element Arrays: Operation applies to each i-element of the arrays
      - [10, 20, 30, 40] - [1, 2, 3, 4] = [9, 18, 27, 36]
    - Array & scalar: Same scalar applied to each i-element of the array
      - [10, 20, 30, 40] - 5 = [5, 15, 25, 35]
- Compile, execute and check the output
  - You can also "play" by changing inputs, executing protocols other than Shamir-Party, etc.

03

Homomorphic Encryption

Homomorphic Encryption (HE)

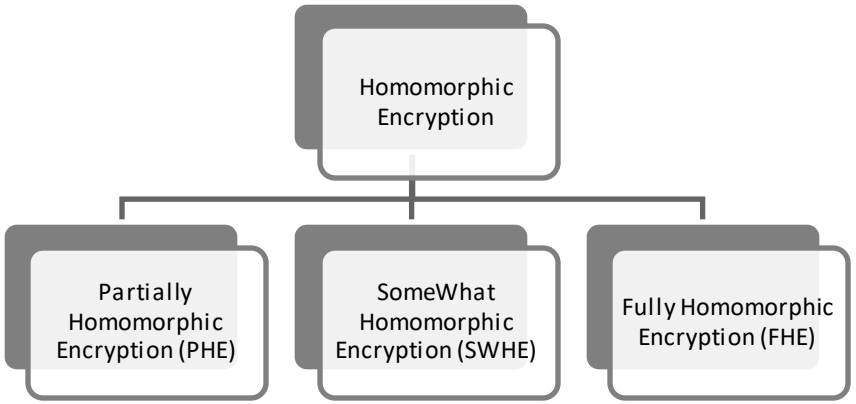
Encryption mechanism that allows computations to be performed directly on encrypted data so that the result, when decrypted, is the same as that expected if the operations would have been performed on the unencrypted data.



Properties

- Data owner does not know the algorithm
- Algorithm owner has no access to cleartext

HE



Scheme	Computational Model	Use
TFHE	Boolean	Comparisons
BGV, BFV	Modular Arithmetic	Integer arithmetic Scalar multiplication
CKKS	Floating point Arithmetic	Polynomial approximation Machine learning models

Classification of HE protocols according to their operation support

- PHE: Sum or multiplication
- SWHE: Sum and multiplication, limited number of iterations
- FHE: Sum and multiplication

Complex functions require polynomial approximations

Both ends share a cryptographic context

HE Libraries

Library	LattiGO	OpenFHE	SEAL
Developer	EPFL - Laussane, Switzerland	OpenFHE	Microsoft
Project	https://github.com/tuneinsight/lattigo	https://github.com/openfheorg/openfhe-development	https://github.com/microsoft/SEAL
Language	Go	C++11	C++17
Version	v4.1.0	v1.0.3	V4.1.1
HE Schemes	BGV/BFV, CKKS, Full-RNS	BFV/BGV, CKKS, FHEW/TFHE	BFV, CKKS
License	Apache 2.0	BSD 2-Clause	MIT
Strengths	Easiness	Lightweight Precision	Performance

MPC vs HE

MPC	HE
N-parties	2-parties Threshold FHE: N-parties
Data NOT shared (secret shares)	Data shared (encrypted)
Algorithm bytecode shared	Algorithm NOT shared
Result available to any M ≤ N parties	Result available to private key owner Threshold FHE
Synchronized execution Communication	Deferred execution
	Huge encryption overhead
Regular performance	Low performance
Wide range of functions	Limited functions Polynomial approximation
Runnable	Libraries
NO orchestration	NO orchestration

Conclusion

- Depends on the use case

04

Higher level frameworks: Brokel

The benefits of the use of a privacy aware framework

- Fills the gaps of MPC/HE technologies with respect to the ecosystem involving participants and data
- Traditional benefits of a framework (improves development speed, best practices, ...)
- Improvements over PPC technologies
- Integration with other frameworks/services

Brokel Framework

- **Objective**
  - "Framework for secure execution of algorithms over public networks, preserving the privacy of both data and algorithms"

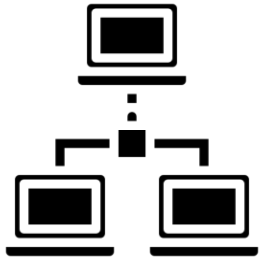
- **Two approaches using Privacy Preserving Technologies**
  - Multi-Party Computation (MPC)
  - Homomorphic Encryption (HE)

From now on, we will focus on the orchestration of MPC campaigns to end up orchestrating the algorithm we have developed.

Brokel: Environments

- **Cloud**:
  - Environment for "*developer*"
  - Shared among "*data providers*"

- **Edge**:
  - Environment for "*data provider*"
  - Private environment where source data "lives"
  - Data never leaves (in plain text) this environment



Brokel: Cloud concepts

- **Schema**: Represents the format and nature of the data
  - JSON Schema
  - Number of columns, column data type, column description, …

- **Campaign**: Represents a MPC execution
  - Specifies the schema that the provided data must match
  - Data providers as "candidates" to provide data for execution
  - Allows to share the algorithm among "*data providers*"
  - Provides synchronization: All "*data providers*" start the execution within a constrained time window

Brokel: Agent concepts

- **Workflow**: Defines the workflow to get data from its source and adapt it to the desired schema
  - Provides a way to retrieve data and transform it
  - Ensures that the format of the workflow output is compliant with the scheme

- **Plan**: MPC algorithm execution management for a given campaign and workflow data output
  - Allows to participate in a MPC campaign
  - Enables execution of most MPC protocols
  - Manages the result of a MPC execution

Brokel: Profiles

- **Developer**: Develops the MPC algorithm
  - Defines schema
  - Creates a MPC campaign
    - Schema required
  - Selects MPC campaign candidatures
  - Develops MPC algorithm
  - Orders (synchronizes) the execution of the MPC campaign

- **Data Provider**: Provides MPC execution
  - Creates a workflow to retrieve and transform data
  - Creates a plan that links a workflow to a MPC campaign
  - Executes MPC algorithm against the workflow data output

Brokel MPC Demo

- Environment pre-generated:
  - MPC service deployed
  - 3 edge MPC data providers

- **Objective**
  - Orchestrate the execution of the developed algorithm by using Brokel Framework

Brokel MPC Demo: Steps I

1. [DEVELOPER] Create "temperature" schema: Temperature in Kelvins
2. [DATA_PROVIDER] Create "temperature" workflow: Get data from source and transform
3. [DEVELOPER] Create "temperature" campaign
    1. Schema: "temperature"
4. [DATA_PROVIDER] Create plan
    1. Set "temperature" workflow
    2. Postulate as "temperature" campaign candidate
5. [DEVELOPER] Accept "temperature" campaign candidatures

Brokel MPC Demo: Steps II

1. [DEVELOPER] Develop MPC algorithm matching "temperature" campaign :
    1. Number of players
    2. Number of samples
2. [DEVELOPER] Upload MPC algorithm
3. [DEVELOPER] Order MPC execution
4. [DATA_PROVIDER] Execute MPC algorithm
    1. Retrieve MPC algorithm
    2. Execute MPC algorithm via the required MPC protocol
5. [DATA_PROVIDER] Get MPC execution results (authorized only)

05

Conclusion

Conclusion

- Privacy Preserving Computation (PPC) addresses major trust problems in data spaces like Gaia-X
  - Service providers: Secure execution environments
  - Data providers: Data privacy
- Different technologies for different use cases
  - Multi-Party Computation
  - Homomorphic Encryption
  - Combinations
- Ecosystem orchestration

Be part of **Gaia-x** and **create**
a **future** that is both **open** and **fair**!

We are **happy** to **have you with us**
and **welcome you**.

About us

- Eneko Gómez
  - Tecnalia – Cybersecurity & Blockchain
  - Senior Researcher
  - Telecommunication Engineer
  - eneko.gomez@tecnalia.com
- Borja Urkizu
  - Tecnalia – Cybersecurity & Blockchain
  - Senior Researcher
  - Computer Science Engineer
  - borja.urquizu@tecnalia.com
- Iñaki Seco
  - Tecnalia – Cybersecurity & Blockchain
  - Senior Researcher
  - Telecommunication Engineer
  - inaki.seco@tecnalia.com