Privacy Preserving SQL Query Execution using an Asymmetric Architecture

Cuong Quoc TO, *Benjamin NGUYEN*, Philippe PUCHERAL

SMIS project

INRIA Rocquencourt, & University of Versailles S[t] Quentin

Outline

- Introduction
- Asymmetric architecture
- General Protocol and simple solutions
- Secure solutions
- Cost model
- Perspectives

INTRODUCTION

Central servers cannot be trusted

The world of data servers today :

- Central (powerful) servers = usual platform for data intensive applications
- To query **distributed users data**, this data is uploaded to the central server, then queried

This data is private and highly sensitive. Nevertheless…

- Privacy violations
- Internal & external attacks on server
- Low ratio Cost/Benefit of an attack

Can they be trusted ???

Our answer is **NO**. (if yours is yes, then you can stop listening to the talk)

Why must users upload their private data ???

Problem Statement

- **Maintain** the functionalities of traditional database servers managing private data while **increasing** privacy protection (availability, durability, expressivity of SQL SFWGH queries, scalability of the system, etc.) by taking into account the fact that central servers are untrusted.
- **Threat model :** honest but curious

Several approaches are possible to securely compute queries:

- Use only a central server (or untrusted components) and use generic (and costly) algorithms.
- Use only a central server and develop (complicated and specific) algorithms.
- Introduce a tangible element of trust, through the use of a **trusted component** and develop a generic methodology to execute any centralized algorithm in this context. ← **our approach**

Trusted Component

Trust can be given for various reasons, we consider here it is *a priori* trust.

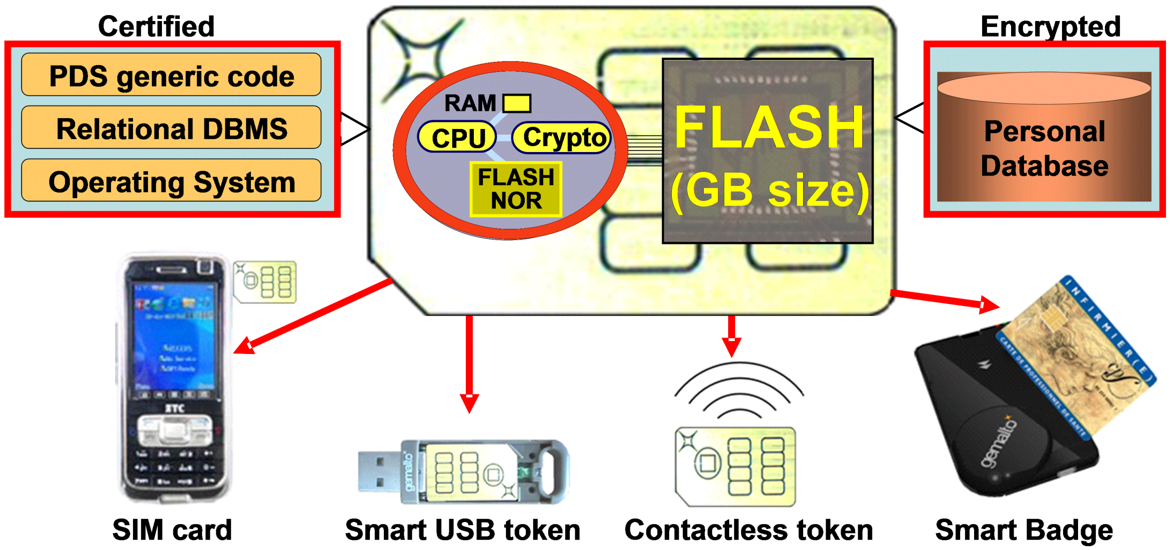
- Hardware secure component
- Social relations
- Contractual
- User (physical) control
- …

We believe that portable secure hardware is a good candidate for *a priori* trust.

This talk is about executing SQL queries using such devices, while maintaining the same functionnalities.

THE ASYMMETRIC ARCHITECTURE

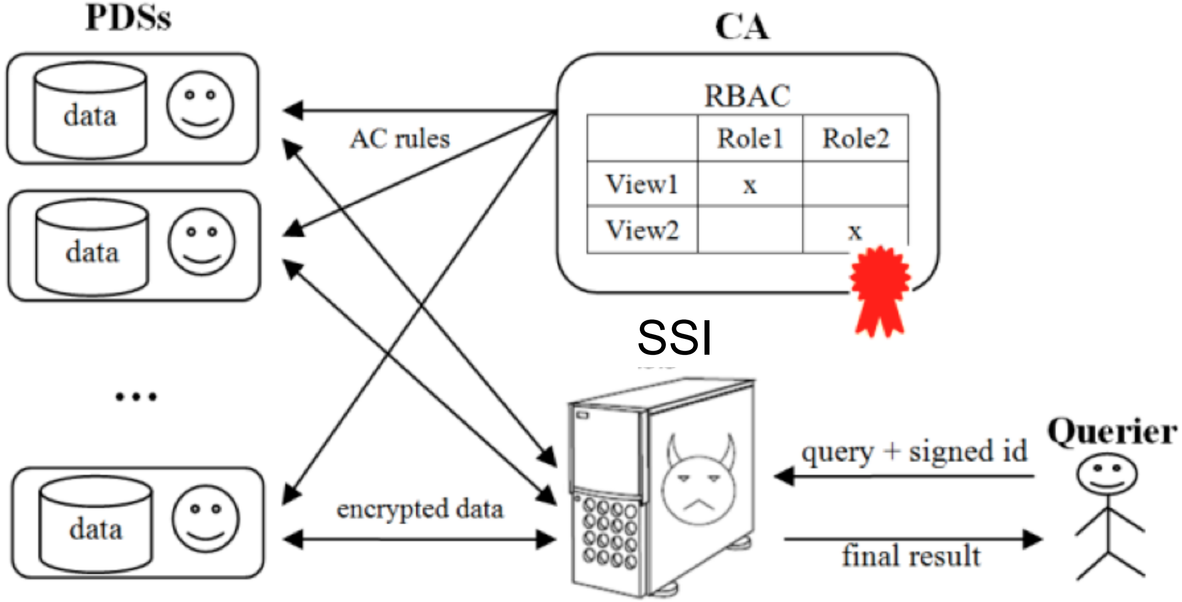
The SMIS Vision : (re)-introducing Secure Portable Tokens (SPTs)



Properties exhibited by a "Secure Portable Token"

1. High security:
- High ratio Cost/Benefit of an attack;
- Secure against its owner;
2. Modest computing resources;
3. Low availability: physically controlled by its owner; connects and disconnects at it will

Asymmetric Architecture



Token	SSI
High security	Untrusted
Modest computing resources	High computing resources
Low availability	High availability

And the querier

- We assume that the end user is a querier who has specific access control rights that are given. These rights can depend on the data owner.
- AC must be enforced i.e. the end user must not obtain any information it is not allowed to see given the AC rules.
- The computation must be private i.e. the SSI must not be able to determine the data that it manages.
- Data contained and managed by an SPT is assumed to be safe (including vs its owner).

Threat model

1. The SSI is the attacker
- The SSI wants to discover raw data
- The SSI and Querier do not collude

2. The querier is the attacker
- The querier wants to obtain ungranted information
- The SSI and Querier can collude

If the collusion of SSI and Querier does not bring any additional information to Querier then Case 2 is the traditional problem studied by AC in databases.

We have studied case 1 for the moment.

GENERAL PROTOCOL

Overview

Computing a query on such an architecture follows 3 steps

1. The querier broadcasts (credentials, query) couple
2. Each PDS decides locally whether to participate or not in the query depending on AC rules and opt-in/opt-out choices.
3. A distributed protocol is established between participating PDS and SSI such that the final result can be delivered to the querier.

/!\ Depending on the complexity of the query, the SSI may only store intermediate results of may play a more active role in the computation

General protection idea

- All the data stored and managed by the SSI is encrypted.

- The problem is therefore to protect against frequency based attacks

→ Our attack hypothesis : the adversary exploits prior knowledge about data distribution to infer some of the plaintext values of ciphertexts.

- « Informal » tradeoff : the more « secure » the encryption, the less operations the SSI will be able to perform on the encrypted data.

Simple example

Querier : INSEE

Authorized view :

Select function, salary

From users

Where salary > 3000

1. Query is broadcast to all users
2. Each user decides whether to answer or not
3. If a user answers, data is sent encrypted to the SSI using a non deterministic scheme (to defeat frequency based attacks)
4. Querier downloads and decrypts the data

Not-so-simple example 1/

Querier : INSEE
Authorized view :

Select function, **AVG(salary)**
From users

1. Query is broadcast to all users
2. Each user decides whether to answer or not
3. If a user answers, data is sent encrypted to the SSI using a non deterministic scheme (to defeat frequency based attacks)
4. Querier downloads and decrypts the data, and /!\ performs the aggregation /!\

This is not an acceptable protocol !

Not-so-simple example 2/

Querier : INSEE

Authorized view :

Select function, **AVG(salary)**

From users

1. Query is broadcast to all users
2. Each user decides whether to answer or not
3. If a user answers, data is sent encrypted to the SSI using a /!\ deterministic scheme /!\ and SSI performs grouping of values with the same key
4. Querier downloads and decrypts the data

This is not an acceptable protocol !

SECURE PROTOCOLS

(1) Random noise solution

Querier : INSEE

Authorized view :

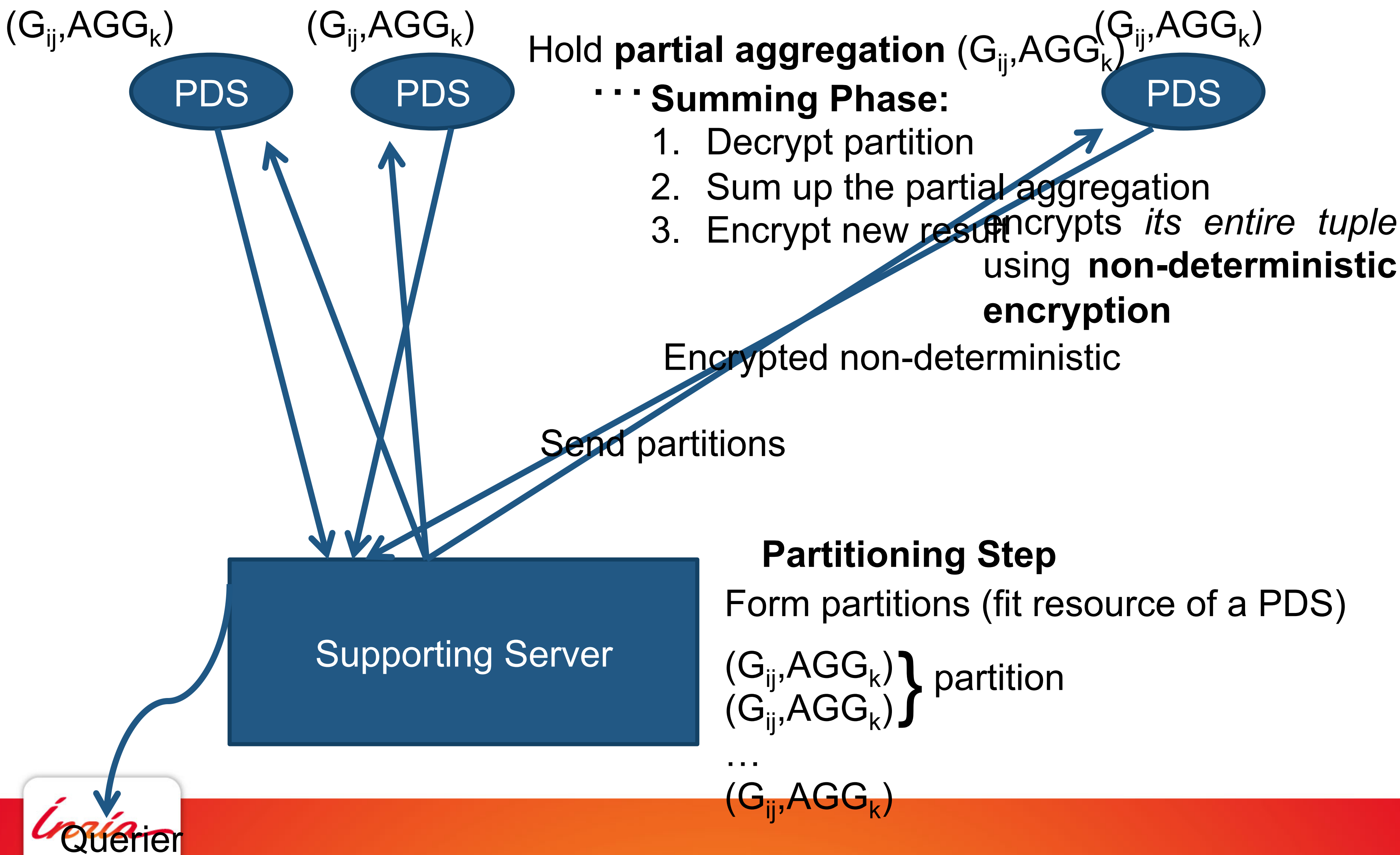
Select function, **AVG(salary)**

From users

1. Query is broadcast to all users
2. Each user decides whether to answer or not
3. If a user answers, it sends its true tuple and n_f false tuples to the SSI using a deterministic encryption scheme
4. SSI performs grouping of values with the same key
5. Querier downloads, decrypts and filters the data

If a sufficient number of fake tuples are added, the distribution is sufficiently perturbated.

(2) Using non-deterministic encryption and secure count method



Using non-deterministic encryption and secure count method

- **Strength:** encrypted Age and encrypted aggregation of Age using *non-deterministic encryption* ⇨ SSI cannot learn anything from this protocol.
- **Weakness:** when number of groups G is too large, a PDS cannot download full aggregation.

(3) Using Nearly Equi-depth Histogram

1. Decrypt partitions
2. Select appropriate tuples
3. Calculate aggregate value
4. Encrypt result [E(Age$_i$), E(Sum of Salary$_i$)]

Note: one partition can contain multiple groups of Age

1. Decrypt partitions
2. Calculate aggregate value
3. Encrypt result [E(Age$_i$), E(Sum of Salary$_i$)]

Note: this phase, 1 partition contains 1 group of Age

Forming nearly equi-depth histogram:

- Divide ∑ tuples/ ∑ buckets = average tuples in bucket.
- Group consecutive frequencies until reaching the averag
- Hash each bucket to a random number

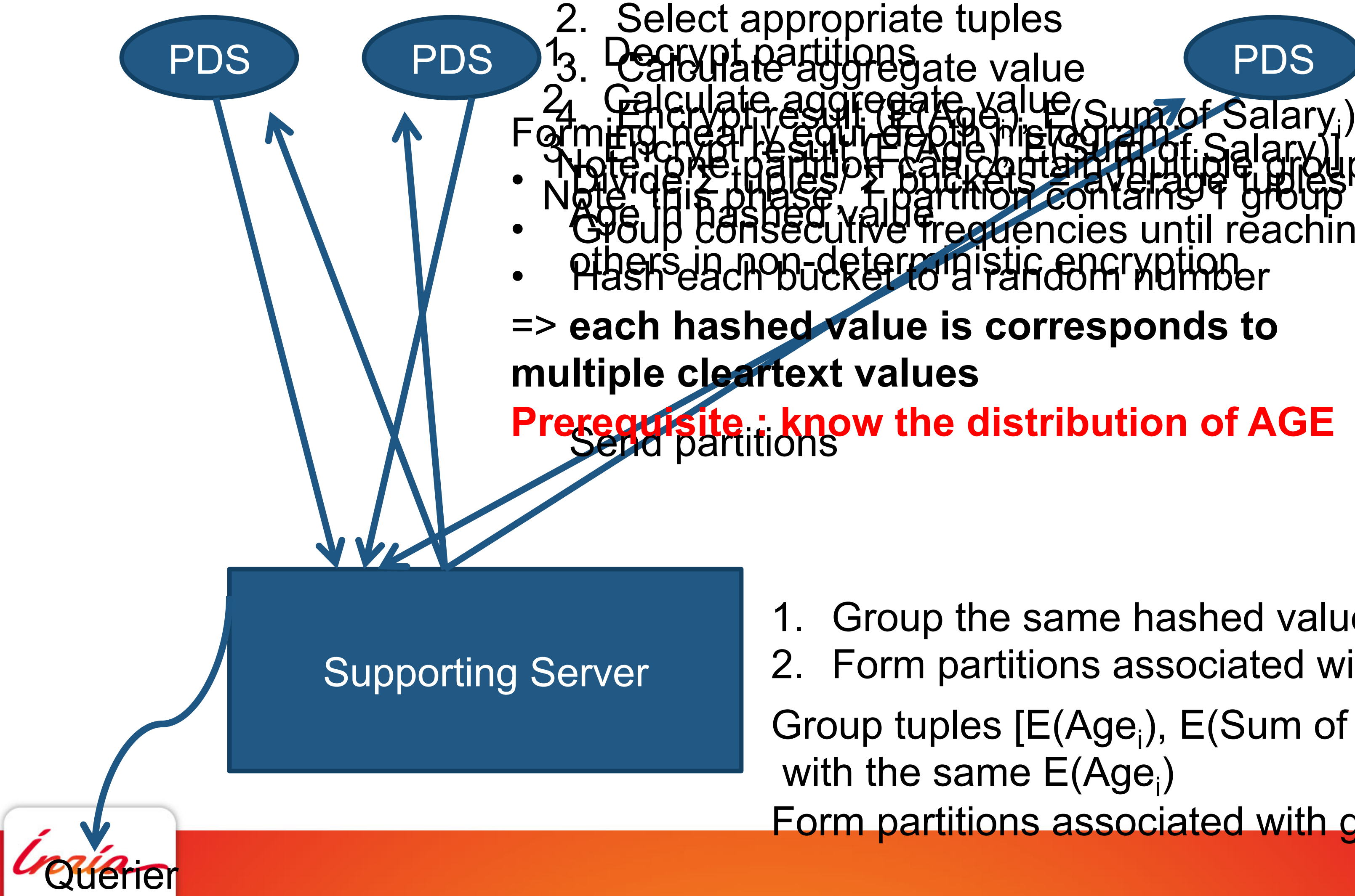
=> **each hashed value is corresponds to multiple cleartext values**

Prerequisite : know the distribution of AGE

Age in hashed value

others in non-deterministic encryption

Send partitions



1. Group the same hashed values of Age
2. Form partitions associated with groups

Group tuples [E(Age$_i$), E(Sum of Salary$_i$)] with the same E(Age$_i$)

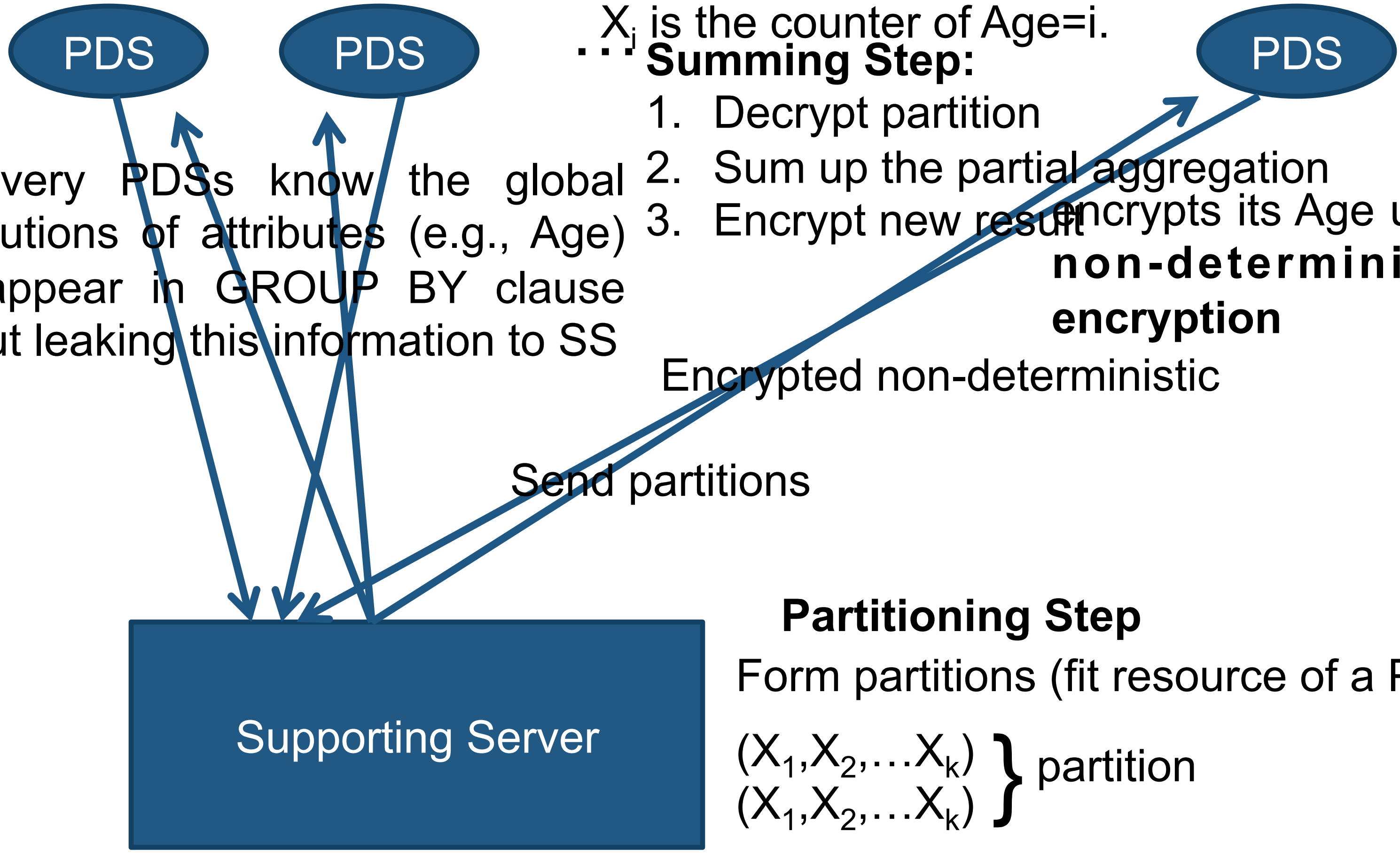
Form partitions associated with groups

Tool : Using secure count method to discover dataset distributions

$(X_1,X_2,\ldots X_k)$ $(X_1,X_2,\ldots X_k)$ … $(X_1,X_2,\ldots X_k)$

Hold **partial distribution** $(X_1,X_2,\ldots X_k)$

X_i is the counter of Age=i.



Let every PDSs know the global distributions of attributes (e.g., Age) that appear in GROUP BY clause without leaking this information to SS

Summing Step:

1. Decrypt partition
2. Sum up the partial aggregation
3. Encrypt new result

encrypts its Age using **non-deterministic encryption**

Encrypted non-deterministic

Send partitions

Partitioning Step

Form partitions (fit resource of a PDS)

$(X_1,X_2,\ldots X_k)$
$(X_1,X_2,\ldots X_k)$ } partition
…
$(X_1,X_2,\ldots X_k)$

COST MODEL

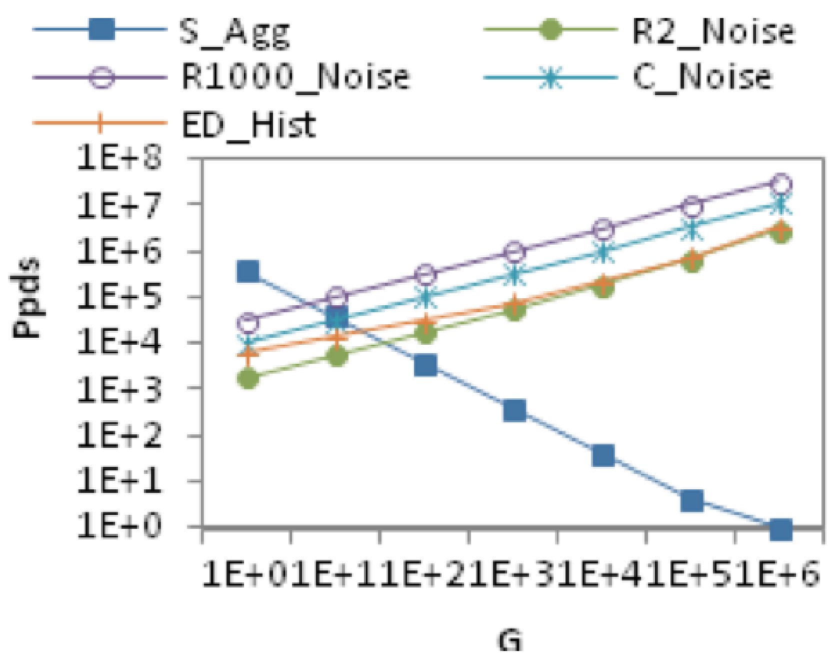
Metrics of interest and parameters

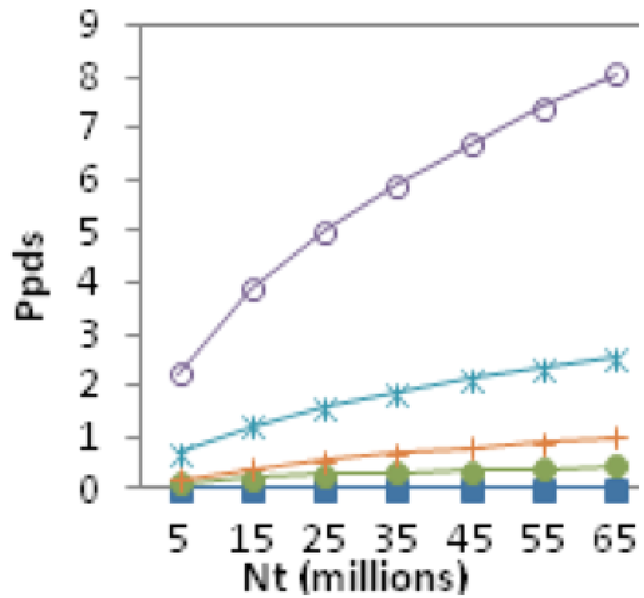
- P_{PDS} : number of PDS participating in the computation of a given phase. This represents parallelism of the protocol
- $Load_Q$: total size of data the PDS and SSI need to process
- T_Q : query response time for contruction and aggregation phases
- T_{local} : average time spent by each PDS participating in the query

Parameters :

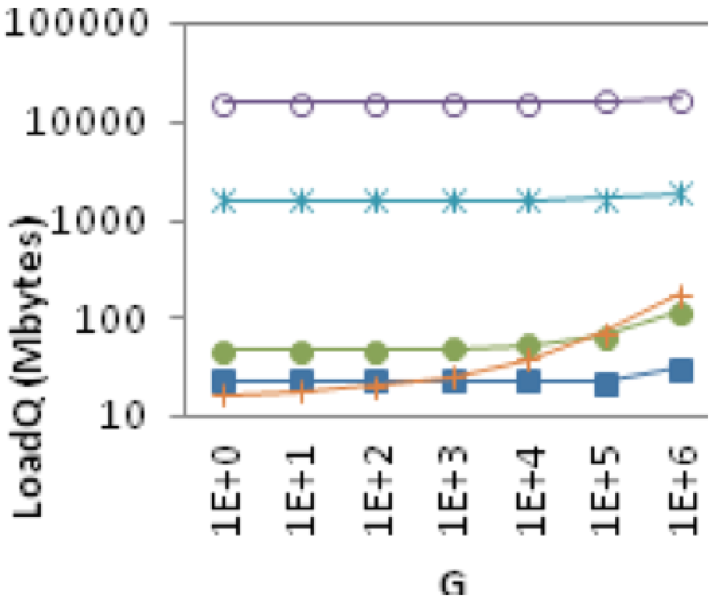
- N_t : size of the dataset (= number of PDS participating) : 5M to 65M
- G : number of groups in the agregation : 1 to 10^6
- time spent by a PDS to process one tuple (transfer, crypto and agg)
- Number of PDS participating in each step of the partial aggregation phase
- Branching factors of the aggregation phase
- Number of fake tuples
- Number of groups in each hash
- …

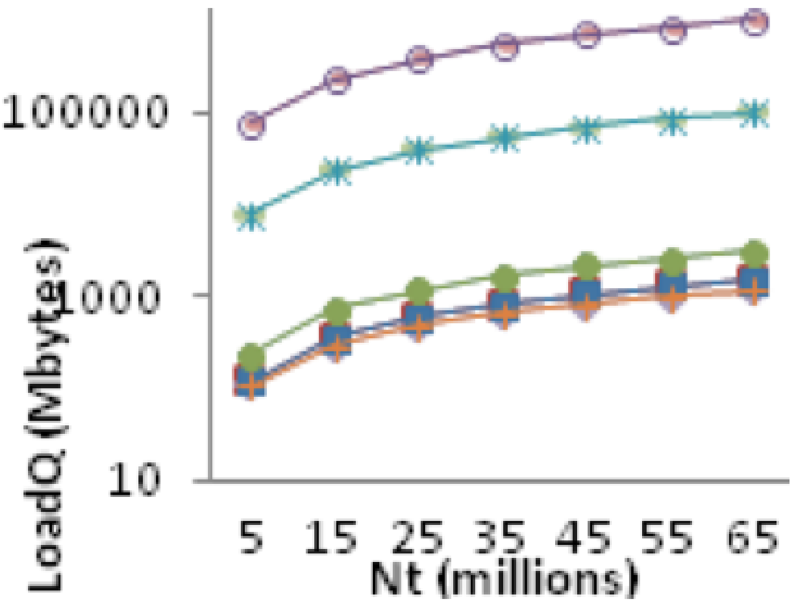
Parallelism



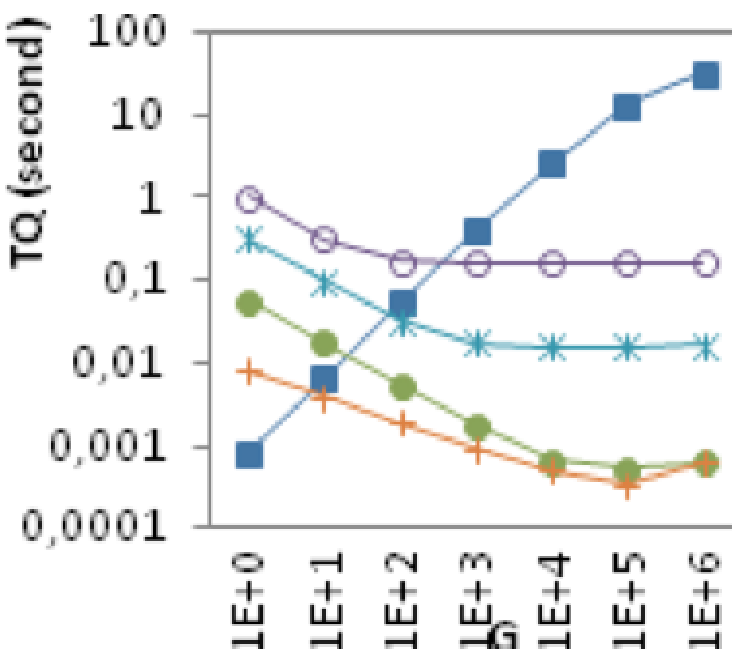


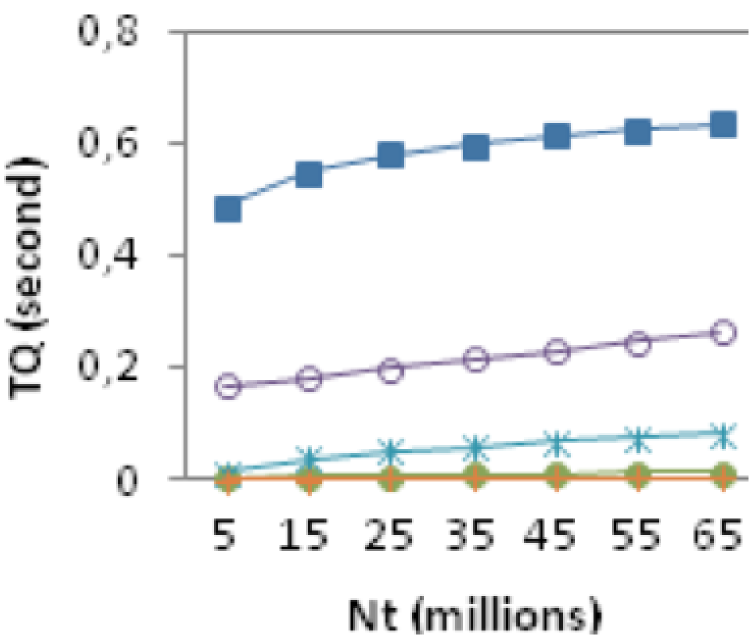
Resource consumption



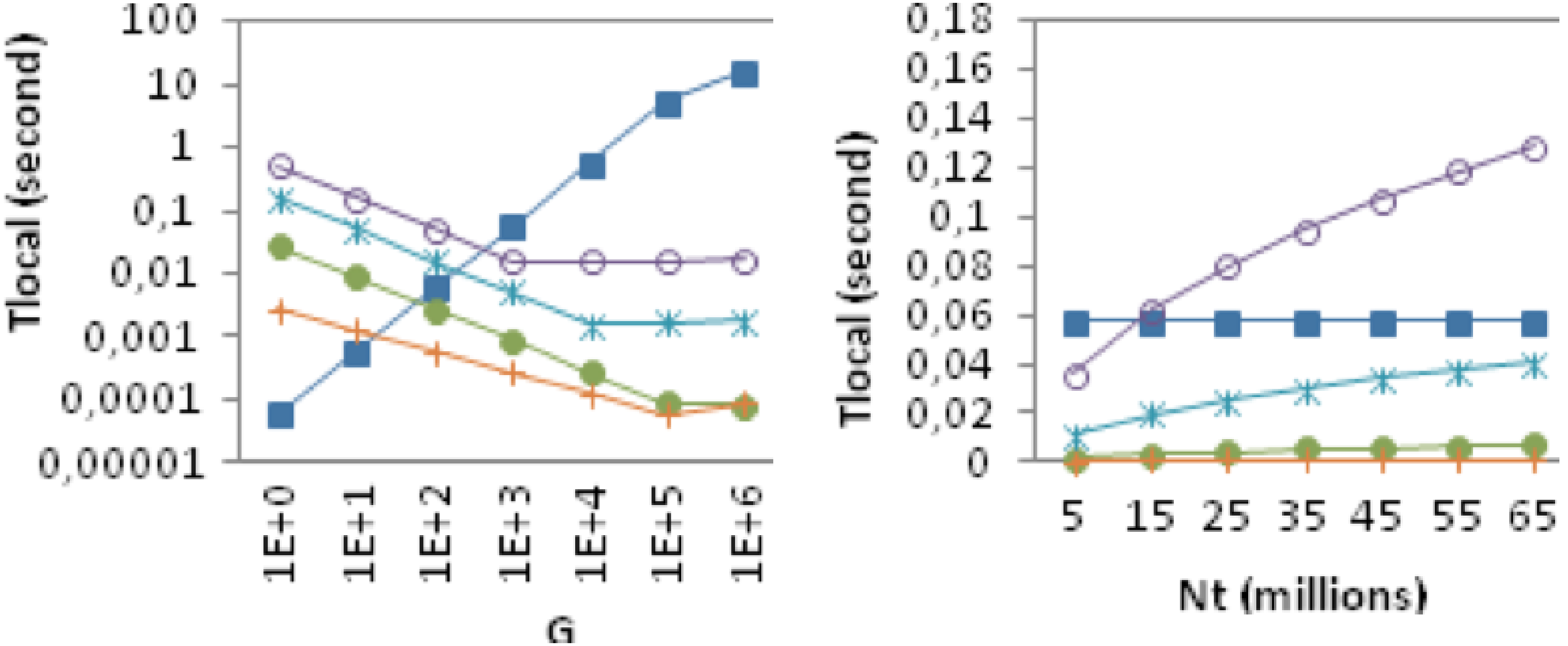


Response time

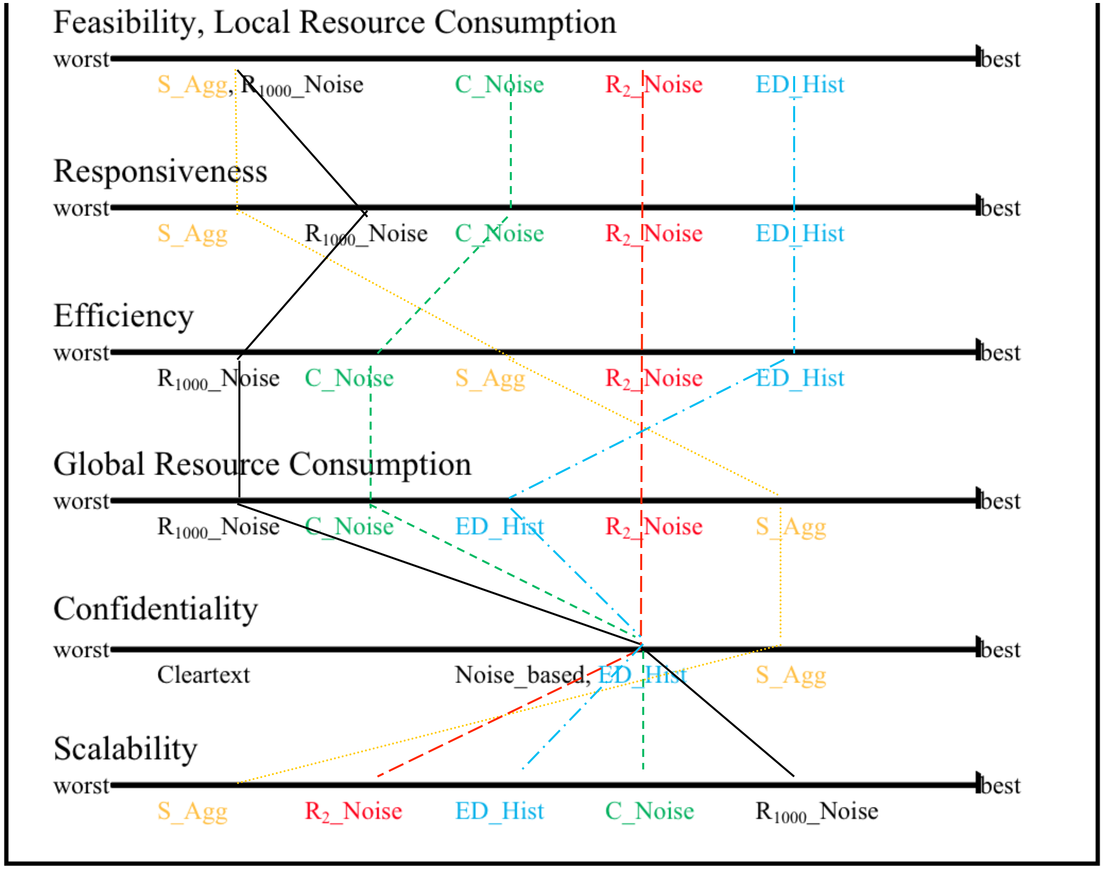




Local Execution Time



Experimental conclusions



CONCLUSION AND FUTURE WORK

Conclusion and future work

- We propose, analyse and evaluation (using a cost model) various algorithms to securely compute SQL Group By queries of private user data on an asymmetric architecture.

- Full implementation on SPTs is currently ongoing.
- Conduct real measurements to validate the cost model.
- Compute theoretical cost bounds (current results are « experimental »)
- Improve or propose better algorithms.

THANK YOU !