Managing Distributed Queries under Personalized Anonymity Constraints

Axel Michel, Benjamin Nguyen, Philippe Pucheral

Managing Distributed Queries under Personalized Anonymity Constraints

Axel Michel[1,2], Benjamin Nguyen[1,2] and Philippe Pucheral[2]
[1]*SDS Team at LIFO, INSA-CVL, Boulevard Lahitolle, Bourges, France*
[2]*PETRUS, INRIA Saclay, Palaiseau, France*
{axel.michel, benjamin.nguyen}@insa-cvl.fr, philippe.pucheral@inria.fr

Keywords: Data Privacy and Security, Big Data, Distributed Query Processing, Secure Hardware

Abstract: The benefit of performing Big data computations over individual's microdata is manifold, in the medical, energy or transportation fields to cite only a few, and this interest is growing with the emergence of smart disclosure initiatives around the world. However, these computations often expose microdata to privacy leakages, explaining the reluctance of individuals to participate in studies despite the privacy guarantees promised by statistical institutes. This paper proposes a novel approach to push personalized privacy guarantees in the processing of database queries so that individuals can disclose different amounts of information (*i.e.* data at different levels of accuracy) depending on their own perception of the risk. Moreover, we propose a decentralized computing infrastructure based on secure hardware enforcing these personalized privacy guarantees all along the query execution process. A performance analysis conducted on a real platform shows the effectiveness of the approach.

1 INTRODUCTION

In many scientific fields, ranging from medicine to sociology, computing statistics on (often personal) private and sensitive information is central to the discipline's methodology. With the advent of the Web, and the massive databases that compose it, statistics and machine learning have become "data science": their goal is to turn large volumes of information linked to a specific individual, called *microdata*, into knowledge. Big Data computation over microdata is of obvious use to the community: medical data is used to improve the knowledge of diseases, and find cures: energy consumption is monitored in smart grids to optimize energy production and resources management. In these applications, real knowledge emerges from the analysis of aggregated microdata, not from the microdata itself[1].

Smart disclosure initiatives, pushed by legislators (*e.g.* EU General Data Protection Regulation (European Union, 2016)) and industry-led consortiums (*e.g.* blue button and green button in the US[2], Midata[3] in the UK, MesInfos[4] in France), hold the promise of a *deluge* of microdata of great interest for analysts. Indeed, smart disclosure enables individuals to retrieve their personal data from companies or administrations that collected them. Current regulations carefully restrict the uses of this data to protect individual's privacy. However, once the data is anonymized, its processing is far less restricted. This is good news, since in most cases, these operations (*i.e.* global database queries) can provide results of tunable quality when run on anonymized data.

Unfortunately, the way microdata is anonymized and processed today is far from being satisfactory. Let us consider how a national statistical study is managed, *e.g.* computing the average salary per geographic region. Such a study is usually divided into 3 phases: (1) the statistical institute (assumed to be a *trusted third party*) broadcasts a query to collect raw microdata along with *anonymity guarantees* (*i.e.*, a privacy parameter like k in the $k-anonymity$ or ε in the *differential privacy* sanitization models) to all users ; (2) each user consenting to participate transmits her microdata to the institute ; (3) the institute computes the aggregate query, while respecting the announced anonymity constraint.

[1]We thus do not consider applications such as targeted advertising, who seek to characterize the users at individual level.

[2]https://www.healthit.gov/patients-families/

[3]https://www.gov.uk/government/news/

[4]http://mesinfos.fing.org/

This approach has two important drawbacks:

1. The anonymity guarantee is defined by the querier (*i.e.* the statistical institute), and applies uniformly to all participants. If the querier decides to provide little privacy protection (*e.g.* a small k in the $k-anonymity$ model), it is likely that many users will not want to participate in the query. On the contrary, if the querier decides to provide a high level of privacy protection, many users will be willing to participate, but the quality of the results will drop. Indeed, higher privacy protection is always obtained to the detriment of the quality and then utility of the sanitized data.
2. The querier is assumed to be trusted. Although this could be a realistic assumption in the case of a national statistics institute, this means it is impossible to outsource the computation of the query. Moreover, microdata centralization exacerbates the risk of privacy leakage due to piracy (Yahoo and Apple recent hack attacks are emblematic of the weakness of cyber defenses[5]), scrutinization and opaque business practices. This erodes individuals trust in central servers, thereby reducing the proportion of citizen consenting to participate in such studies, some of them unfortunately of great societal interest.

The objective of this paper is to tackle these two issues by reestablishing *user's empowerment*, a principle called by all recent legislations protecting the management of personal data (European Union, 2016). Roughly speaking, user's empowerment means that the individual must keep the control of her data and of its disclosure in any situation. More precisely, this paper makes the following contributions:

- proposing a query paradigm incorporating personalized privacy guarantees, so that each user can trade her participation in the query for a privacy protection matching her personal perception of the risk,
- providing a secure decentralized computing framework guaranteeing that the individual keeps her data in her hands and that the query issuer never gets cleartext raw microdata and sees only a sanitized aggregated query result matching all personalized privacy guarantees,
- conducting a performance evaluation on a real dataset demonstrating the effectiveness and scalability of the approach.

[5]Yahoo 'state' hackers stole data from 500 million users - BBC News. www.bbc.co.uk/news/world-us-canada-37447016

The rest of the paper is organized as follows. Section 2 presents related works and background materials allowing the precisely state the problem addressed. Section 3 details the core of our contribution. Section 4 shows that the overhead incurred by our algorithm, compared to a traditional query processing technique, remains largely tractable. Finally, Section 5 concludes.

2 STATE OF THE ART AND PROBLEM STATEMENT

2.1 Related Works on Privacy-Preserving Data Publishing

Anonymization has been a hot topic in data publication since the 1960's for all statistical institutions wanting to publish aggregate data. The objective of most of these data publishing techniques is to provide security against an attacker who is going to mount *deanonymization* attacks, which will link some sensitive information (such as their salary or medical diagnosis) to a specific individual. Particular attention was drawn to the problem by Sweeney, the introduction of the k-anonymity model (Sweeney, 2002) that we consider in this paper. k-anonymity is a partition based approach to anonymization, meaning that the original dataset, composed of individual's microdata, is partitionned, through generalization or suppression of values, into groups who have similar values which will then be used for grouping.

The partition-based approach splits the attributes of the dataset in two categories: a quasi-identifier and some sensitive data. A quasi-identifier (denoted QID) is a set of attributes for which some records may exhibit a combination of unique values in the dataset, and consequently be identifying for the corresponding individuals (e.g., ZipCode, BirthDate, Gender). The sensitive part (denoted SD) encompasses the attribute(s) whose association with individuals must be made ambiguous (e.g., Disease).

Partition-based approaches essentially apply a controlled degradation to the association between individuals (represented in the dataset by their quasi-identifier(s)) and their sensitive attribute(s). The initial dataset is deterministically partitioned into groups of records (classes), where quasi-identifier and sensitive values satisfy a chosen partition-based privacy model. The original k-Anonymity model (Sweeney, 2002) requires each class to contain at least k indistinguishable records, thus each sensitive data will be associated with at least k records. Many other

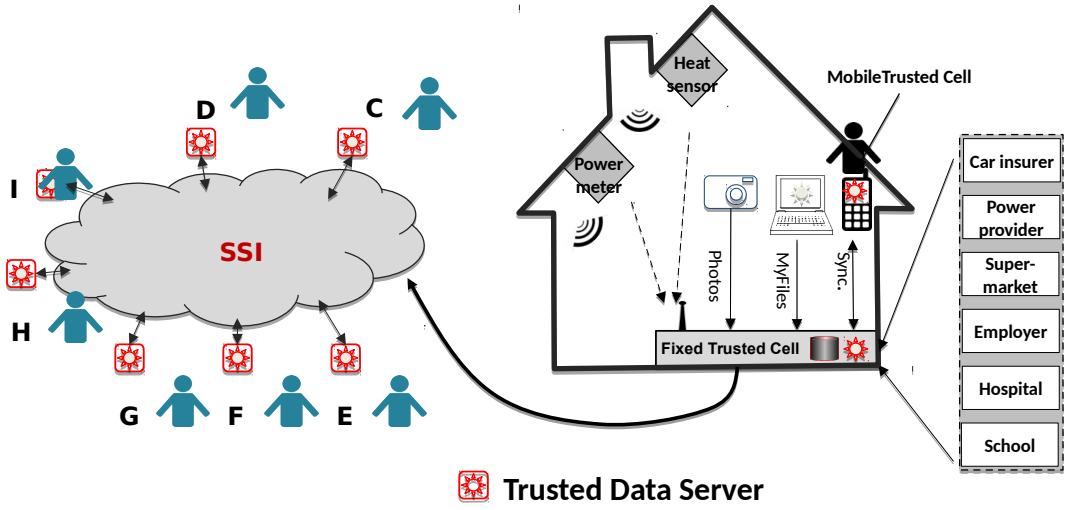

Figure 1: Trusted Cells reference architecture.

models have been introduced since 2006, such as ℓ-Diversity (Machanavajjhala et al., 2006) or t-Closeness (Li et al., 2010). Each model further constrain the distribution of sensitive data within each class, tackling different adversarial assumptions. For example, the ℓ-Diversity principle requires that the set of sensitive data associated to each equivalence class be linked to ℓ different sensitive values. t-closeness requires each class to have a similar distribution of sensitive values. To illustrate this, table 1 shows a 3-anonymous and 2-diverse version of a dataset. This means, for each tuple, at least two others have the same quasi-identifier (*i.e.* 3-anonymity) and for each group of tuples with the same quasi-identifier, there are at least two distinct sensitive values (*i.e.* 2-diversity). It is important to note that the higher the k and ℓ, the better the privacy protection, but the lower the precision (or quality) of the query.

Table 1: 3−anonymous and 2−diverse table.

Quasi-identifier		Sensitive
ZIP	Age	Condition
112**	> 25	Cancer
112**	> 25	Cancer
112**	> 25	Heart Disease
1125*	*	Heart Disease
1125*	*	Viral Infection
1125*	*	Cancer

A different concept is *differential privacy*, introduced by Dwork in (Dwork, 2006). Differential privacy is more adapted to interactive query answering. It's advantage is to provide formal guarantees regardless of the knowledge of the adversary. However, differential privacy limits the type of computation which can be made on the data. Moreover, fixing the privacy parameter ε is a cumbersome and not intuitive task, out of reach of lambda individuals.

Another approach is to make an agreement between the user and the querier. Te concept of *sticky policies* presented by Trablesi, Neven, Ragget et al. (Trabelsi et al., 2011) consists to make a policy about authorization (*i.e.* what the querier can do) and obligation(*i.e.* what he querier must do) which will stick to the user data.

2.2 Reference Computing Architecture

Concurrently with smart disclosure initiatives, the *Personal Information Management System* (PIMS) paradigm has been conceptualized (Abiteboul et al., 2015), and emerges in the commercial sphere (*e.g.* Cozy Cloud, OwnCloud, SeaFile). PIMS holds the promise of a Privacy-by-Design storage and computing platform where each individual can gather her complete digital environment in one place and share it with applications and other users under her control. The *Trusted Cells architecture* presented in (Anciaux et al., 2013), and pictured in Figure 1, precisely answers the PIMS requirements by preventing data leaks during computations on personal data. Hence, we consider Trusted Cells as a reference computing architecture in this paper.

Trusted Cells is a decentralized architecture by nature managing computations on microdata through the collaboration of two parties. The first party is a (potentially large) set of personal *Trusted Data Servers* (TDSs) allowing each individual to manage her data with tangible elements of trust. Indeed, TDSs incorporate tamper resistant hardware (*e.g.* smartcard, secure chip, secure USB token) securing the data and code against attackers and users' misusages. Despite the diversity of existing tamper-resistant devices, a

TDS can be abstracted by (1) a Trusted Execution Environment and (2) a (potentially untrusted but cryptographically protected) mass storage area where the personal data resides. The important assumption is that the TDS code is executed by the secure device hosting it and then cannot be tampered, even by the TDS holder herself.

By construction, secure hardware exhibit limited storage and computing resources and TDSs inherit these restrictions. Moreover, they are not necessarily always connected since their owners can disconnect them at will. A second party, called hereafter *Supporting Server Infrastructure* (SSI), is thus required to manage the communications between TDSs, run the distributed query protocol and store the intermediate results produced by this protocol. Because SSI is implemented on regular server(s), *e.g.* in the Cloud, it exhibits the same low level of trustworthiness.

The resulting computing architecture is said *asymmetric* in the sense that it is composed of a very large number of low power, weakly connected but highly secure TDSs and of a powerful, highly available but untrusted SSI.

2.3 Reference Query Processing Protocol

By avoiding delegating the storage of personal data to untrusted cloud providers, Trusted Cells is key to achieve user empowerment. Each individual keeps her data in her hands and can control its disclosure. However, the decentralized nature of the Trusted Cells architecture must not hinder global computations and queries, impeding the development of services of great interest for the community. *SQL/AA* (SQL Asymmetric Architecture) is a protocol to execute standard SQL queries on the Trusted Cells architecture (To et al., 2014; To et al., 2016). It has been precisely designed to tackle this issue, that is executing global queries on a set of TDSs without recentralizing microdata and without leaking any information.

The protocol, illustrated by the Figure 2, works as follows. Once an SQL query is issued by a querier (*e.g.* a statistic institute), it is computed in three phases: first the *collection phase* where the querier broadcasts the query to all TDSs, TDSs decide to participate or not in the computation (they send dummy tuples in that case to hide their denial of participation), evaluate the `WHERE` clause and each TDS returns its own encrypted data to the SSI. Second, the *aggregation phase*, where SSI forms partitions of encrypted tuples, sends them back to TDSs and each TDS participating to this phase decrypts the input partition, removes dummy tuples and computes the aggregation function (*e.g.* `AVG`, `COUNT`). Finally the *filtering phase*, where TDSs produce the final result by filtering out the `HAVING` clause and send the result to the querier. Note that the TDSs participating to each phase can be different. Indeed, TDSs contributing to the collection phase act as data producers while TDSs participating to the aggregation and filtering phases act as trusted computing nodes. The tamper resistance of TDSs is the key in this protocol since a given TDS belonging to individual i_1 is likely to decrypt and aggregate tuples issued by TDSs of other individuals $i_2, \ldots, i_n$. Finally, note that the aggregation phase is recursive and runs until all tuples belonging to a same group have been actually aggregated. We refer the interested reader to (To et al., 2014; To et al., 2016) for a more detailed presentation of the SQL/AA protocol.

2.4 Problem Statement

In order to protect the privacy of users, queries must respect a certain degree of anonymity. Our primary objective is to push personalized privacy guarantees in the processing of regular statistical queries so that individuals can disclose different amount of information (i.e., data at different level of accuracy) depending on their own perception of the risk. To the best of our knowledge, no existing work has addressed this issue. For the sake of simplicity, we consider SQL as the reference language to express statistical/aggregate queries because of its widespread usage. Similarly, we consider personalized privacy guarantees derived from the k-anonymity and ℓ-diversity models because (1) they are the most used in practice, (2) they are recommended by the European Union (European Union, 2014) and (3) they can be easily understood by individuals[6]. The next step in our research agenda is to extend our approach to other query languages and privacy guarantees but this ambitious goal exceeds the scope and expectation of this paper

Hence, the problem addressed in this paper is to propose a (SQL) query paradigm incorporating personalized (k-anonymity and ℓ-diversity) privacy guarantees and enforcing these individual guarantees all along the query processing without any possible leakage.

[6]The EU Article 29 Working Group mention these characteristics as strong incentives to make these models effectively used in practice or tested by several european countries (e.g., the Netherlands and French statistical institutes).

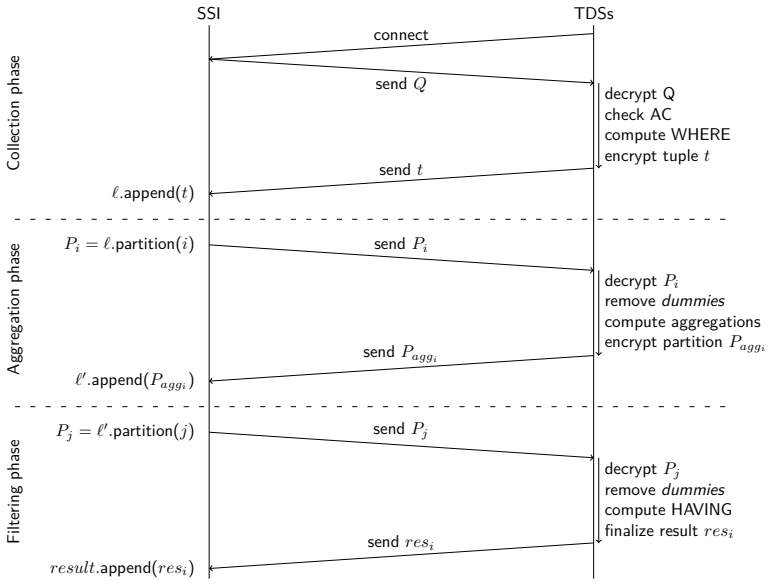

Figure 2: SQL/AA Protocol.

3 PERSONALIZED ANONYMITY GUARANTEES IN SQL

3.1 Modeling Anonymisation Using SQL

We make the assumption that each individual owns a local database hosted in her personal TDS and that these local databases conform to a common schema which can be easily queried in SQL. For example, power meter data (resp., GPS traces, healthcare records, etc) can be stored in one (or several) table(s) whose schema is defined by the national distribution company (resp., an insurance company consortium, the Ministry of Health, etc). Based on this assumption, the querier (i.e., the statistical institute) can issue regular SQL queries as shown by the Figure 4.

For the sake of simplicity, we do not consider joins between data stored in different TDSs but internal joins which can be executed locally by each TDS are supported. We refer to (To et al., 2014; To et al., 2016) for a deeper discussion on this aspect which is not central to our work in this paper.

Anonymity Guarantees are defined by the querier, and correspond to the k and ℓ values that will be achieved by the end of the process, for each group produced. They correspond to the commitment of the querier towards any query participant. Different k and ℓ values can be associated to different granularity of grouping. In the example pictured in Figure 3, the querier commits to provide $k \geq 5$ and $\ell \geq 3$ at a (`City`,`Street`) grouping granularity and $k \geq 10$ and $\ell \geq 3$ at a (`City`) grouping granularity.

Anonymity Constraints are defined by the users, and correspond to the values they are willing to accept in order to participate in the query. Back to the example of Figure 3, Alice's privacy policy stipulates a minimal anonymization of $k \geq 5$ and $\ell \geq 3$ when attribute Salary is queried.

According to the anonymity guarantees and constraints, the query computing protocol is as follows. The querier broadcasts to all potential participants the query to be computed along with metadata encoding the associated anonymity guarantees. The TDS of each participant compares this guarantees with the individual's anonymity constraints. This principle shares some similarities with *P3P*[7] with the matching between anonymity guarantees and constraints securely performed by the TDS. If the guarantees exceed the individual's constraints, the TDS participates to the query by providing real data at the finest grouping granularity. Otherwise, if the TDS finds a grouping granularity with anonymity guarantees matching her constraints, it will participate, but by providing a degraded version of the data, to that coarser level of granularity (looking at Figure 3, answering the `group by city, street` clause is not acceptable for Bob, but answering just with `city` is). Finally, if no match can be found, the TDS produces fake data (called *dummy tuples* in the protocol) to hide its denial of participation. Fake data is required to avoid the querier from inferring information about the individual's privacy policy or about her membership to the `WHERE`

[7] https://www.w3.org/P3P/

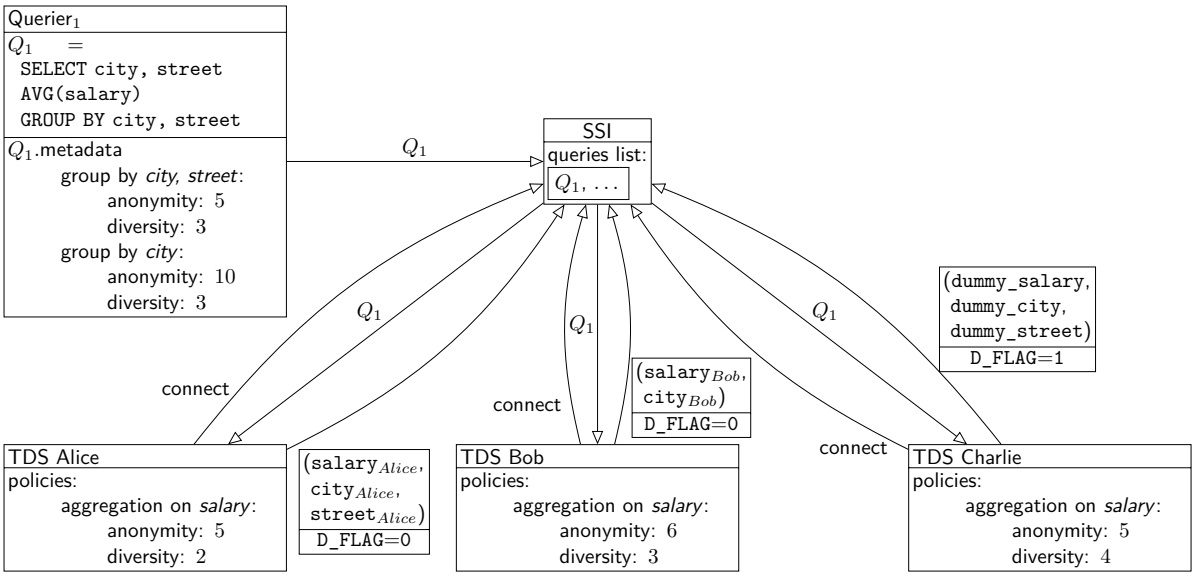

Figure 3: Example of collection phase with anonymity constraints.

```
SELECT <Aggregate function(s)>
FROM <Table(s)>
WHERE <condition(s)>
GROUP BY <grouping attribute(s)>
HAVING <grouping condition(s)>
```

Figure 4: Regular SQL query form.

clause of the query.

Figure 3 illustrates this behavior. By comparing the querier anonymity guarantees with their respective constraints, the TDSs of Alice, Bob and Charlie respectively participate with fine granularity values (Alice), with coarse granularity values (Bob), with dummy tuples (Charlie).

The working group *ODRL*[8] is looking at some issues similar to the expression of privacy policies. However, this paper is not discussing about how users can express their privacy policy in a standard way.

3.2 The ^{k_i}SQL/AA protocol

We now describe our new protocol, that we call ^{k_i}SQL/AA to show that it takes into account many different k values of the i different individuals. ^{k_i}SQL/AA is an extension of the SQL/AA protocol (To et al., 2014; To et al., 2016) where the enforcement of the anonymity guarantees have been pushed in the *collection*, *aggregation* and *filtering* phases.

Collection phase: After TDSs download the query, they compare the anonymity guarantees announced by the querier with their own anonymity constraints. As discussed above (see Section 3.1) TDSs send real data at the finest grouping granularity compliant with their anonymity constraints or send a dummy tuple if no anonymity constraint can be satisfied.

Aggregation phase: To ensure that the anonymization guarantees can be verified at the filtering phase, clauses `COUNT(*)` and `COUNT(DISTINCT A)` are computed in addition to the aggregation asked by the querier. `COUNT(*)` will be used to check that the $k-$anonymity guarantee is met while `COUNT(DISTINCT A)` will be used to check the $\ell-$diversity guarantee on attribute (or group of attributes) A on which the aggregate function applies (*e.g.* salary in our example)[9]. If tuples with varying grouping granularity enter this phase, they are aggregated separately, *i.e.* one group per grouping granularity.

Filtering phase: Besides `HAVING` predicates which can be formulated by the querier, the `HAVING` clause is used to check the anonymity guarantees. Typically, $k-$anonymity sums up to check `COUNT(*)` $\geq k$ while $\ell-$diversity is checked by `COUNT(DISTINCT A)` $\geq \ell$. If these guarantees are not met for some tuples, they are not immediately discarded. Instead, the protocol tries to merge them with a group of coarser granularity encompassing them. Let us consider the example of Table 2(a). The tuple `(Bourges, Bv.Lahitolle, 1600)` is merged with the tuple `(Bourges, ******, 1400)` to form the tu-

[8] https://www.w3.org/community/odrl/

[9] Since this clause is an holistic function, we can compute it while the aggregation phase by adding naively each distinct value under a list or using a cardinality estimation algorithm such as *HyperLogLog* (Flajolet et al., 2007).

Table 2: Filtering phase.

(a) Example of a post aggregation phase result

city	street	AVG(salary)	COUNT(*)	COUNT (DISTINCT salary)
Le Chesnay	Dom. Voluceau	1500	6	4
Le Chesnay	******	1700	9	6
Bourges	Bv. Lahitolle	1600	3	3
Bourges	******	1400	11	7

(b) Privacy guarantees of the query

Attributes	k	ℓ
city, street	5	3
city	10	3

(c) Data sent to the querier

city	street	AVG(salary)
Le Chesnay	Dom. Voluceau	1500
Bourges	******	1442.86

ple (Bourges, ******, 1442.86). Merges stop when all guarantees are met. If, despite merges, the guarantees cannot be met, the corresponding tuples are removed form the result. Hence, the querier will receive every piece of data which satisfies the guaranties, and only these ones, as shown on Table 2(c).

How to generalize. To reach the same k and ℓ values on the groups, the grouping attributes can be generalized in different orders, impacting the quality of the result for the querier. For instance, if the GroupBy clause involves two attributes Address and Age, would it be better to generalize the tuples on Address (*e.g.* replacing <City,Street> by <City>) or on Age (replacing exact values by intervals) ? It would be valuable for the querier to give *"hints"* to the TDSs on how to generalize the data, by indicating which attributes to generalize, and what privacy guarantees will be enforced after each generalization. In the following example, we consider the UCI Adult dataset (Lichman, 2013), we define a GroupBy query GB on attributes Age, Workclass, Education, Marital_status, Occupation, Race, Gender, Native_Country and we compute the average fnlwgt operation OP=AVG(fnlwgt). MD represents the metadata attached to the query. Each metadata indicates which k and ℓ can be guaranteed after a given generalization operation. Depending on the attribute type, generalizing an attribute may correspond to climbing up in a generalization hierarchy (for categorical attributes such as Workclass or Race) or replacing a value by an interval of greater width (for numeric values such as Age or Education). The del operation means that the attribute is simply removed. The ordering of the metadata in MD translates the querier requirements.

```
GB= Age, Workclass, Education, Marital_status,
    Ocupation, Race, Gender, Native_Country;
OP= AVG(fnlwgt);
MD= :                      k=5 l=3,
    age->20:               k=6 l=3,
    workclass->up:         k=8 l=4,
    education->5:          k=9 l=4,
    marital_status->up: k=9 l=4,
    occupation->up:     k=10 l=4,
    race->up:           k=11 l=5,
    gender->del:        k=14 l=6,
    native country->del:k=15 l=7,
    age->40:            k=17 l=8;
```

Figure 5: ^{k_i}SQL/AA query example.

4 EXPERIMENTAL EVALUATION

We have implemented the ^{k_i}SQL/AA protocol on equivalent open-source hardware used in (To et al., 2014; To et al., 2016). The goal of this experimental section is to show that there is very little overhead when taking into account personalized anonymity constraints compared to the performance measured by the original implementation of To et al.. Our implementation is tested using the classical adult dataset of UCI-ML (Lichman, 2013)

4.1 Implementation of ^{k_i}SQL/AA protocol

The implementation of ^{k_i}SQL/AA builds upon the *secure aggregation* protocol of SQL/AA (To et al., 2014; To et al., 2016), recalled in Section 2.3. The secure aggregation is based on non-deterministic encryption as AES-CBC to encrypt tuples. Each TDS encrypts its data with the same secret key but with a different initialization vector such that two tuples with the same value have two different encrypted values. The protection ensures the SSI cannot infer personal informations by the distribution of same encrypted values. To make sure that aggregations are entirely

computed, the SSI uses a divide and conquer partitioning to make the TDS compute partial aggregations on partitions of data. Then the SSI merges partial aggregations to compute the last aggregation. The aggregation phase is illustrated by the Figure 6.

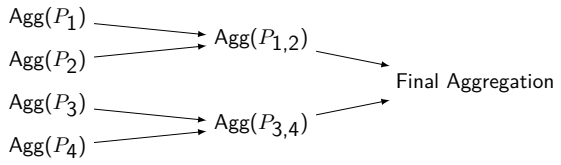

Figure 6: Aggregation phase with four partitions.

In ^{k_i}SQL/AA, the aggregation phase has been kept unchanged from the SQL/AA system since our contribution is on the collection phase and the filtering phase. The algorithm implementing the collection phase of ^{k_i}SQL/AA is given below (see Algorithm 1). As in most works related to data anonymization, we make the simplifying assumption that each individual is represented by a single tuple. Hence, the algorithm always return a single tuple. This tuple is a dummy if the privacy constraints or the `WHERE` clause cannot be satisfied.

Algorithm 1 Collection Phase.

```
procedure COLLECTION_PHASE(Query Q)
    t ← getTuple(Q)
    p ← getConstraints(Q)
    g ← getGuarantees(Q)
    i ← 0
    if verifyWhere(t,Q) then
        while g_i < p do
            if not canGeneralize(t) then
                t ← makeDummy(Q)
            else
                t ← nextGeneralization(t)
                i ← i+1
            end if
        end while
    else
        t ← makeDummy(Q)
    end if
return Encrypt(t)
end procedure
```

The filtering phase algorithm is given by Algorithm 2.

First, the algorithm sorts tuples of the aggregation phase by generalization level, making multiple sets of tuples of same generalization level. The function `verifyHaving` checks if `COUNT(*)` and `COUNT(Disctinct)` match the anonymization guarantees expected at this generalization level. If so, the tuple is added to the result. Otherwise, it is further generalized and merged with the set of higher generalization level. At the end, every tuple which cannot reach the adequate privacy constraints, despite achieving maximum generalization is not included in the result.

Algorithm 2 Filtering Phase.

```
procedure FILTERING_PHASE(Query Q, TuplesSet T)
    sortByGeneralizationLevel(T)
    g ← getGuarantees(Q)
    for i from 0 to MaxGeneralizationLevel(Q) do
        for t ∈ T_i do
            t ← decrypt(t)
            if verifyHaving(t,Q) then
                result.addTuple(t)
            else if canGeneralize(t) then
                t ← nextGeneralization(t)
                T_{i+1}.addTuple(t)
            end if
        end for
    end for
return result
end procedure
```

4.2 Experiments Platform

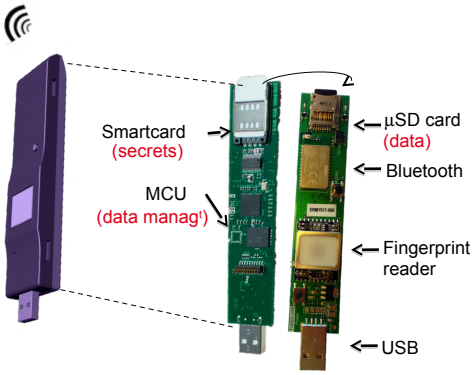

Figure 7: TDS characteristics.

The performance of the ^{k_i}SQL/AA protocol presented above has been measured on the tamper resistant open-source hardware platform shown in Figure 7. The TDS hardware platform consists of a 32-bit ARM Cortex-M4 microcontroller with a maximum frequency of 168MHz, 1MB of internal NOR flash memory and 196kb of RAM, itself connected to a μSD card containing all the personal data in an encrypted form and to a secure element (open smartcard) containing cryptographic secrets and algorithms. The TDS can communicate either through

USB (our case in this study) or Bluetooth. Finally, the TDS embeds a relational DBMS engine, named PlugDB[10], running in the microcontroller. PlugDB is capable to execute SQL statement over the local personal data stored in the TDS. In our context, it is mainly used to implement the WHERE clause during the Collection phase of ^{k_i}SQL/AA.

Our performance tests use the Adult dataset from the UCI machine learning repository (Lichman, 2013). This dataset is an extraction from the American census bureau database. We modified the dataset the same way of (Iyengar, 2002; Bayardo and Agrawal, 2005). We kept eight attributes to perform the `GoupBy` clause, namely `age`, `workclass`, `education`, `marital status`, `occupation`, `race`, `native country` and `gender`. Since our work is based on GROUP BY queries, we also kept the `fnlwgt` (*i.e.* final weight) attribute to perform an `AVG` on it. The final weight is a computed attribute giving similar value for people with similar demographic characteristics. We also removed each tuple with a missing value. At the end we kept 30162 tuples. Attributes `age` and `education` are treated as numeric values and others as categorical values. Since TDS have limited resources, categorical value are represented by a bit vector. For instance, the categorical attribute `workclass` is represented by a 8 bits value and its generalization process is performed by taking the upper node on the generalization tree given in Figure 8. The native country attribute is the largest and requires 49 bits to be represented.

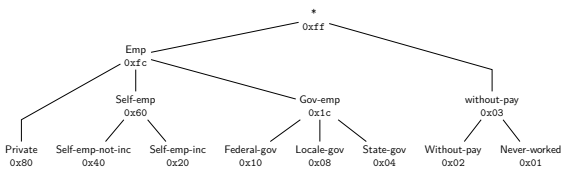

Figure 8: Generalization tree of `workclass` attribute.

4.3 Performance measurements

^{k_i}SQL/AA being an extension of SQL/AA protocol, this section focuses on the evaluation of the overhead incurred by the introduction of anonymisation guarantees in the query protocol. Then, it sums up to a direct comparison between ^{k_i}SQL/AA and the genuine SQL/AA. To make the performance evaluation more complete, we first recall from (To et al., 2016) the comparison between SQL/AA itself and other state of the art methods to securely compute aggregate SQL queries. This comparison is pictured in Figure 9.

[10]https://project.inria.fr/plugdb/en/

The Paillier curve shows the performance to compute aggregation in a secure centralized server using homomorphic encryption, presented in (Ge and Zdonik, 2007). The DES curve uses also a centralized server and a DES encryption scheme (data are decrypted at computation time). Finally, SC curves correspond to the SQL/AA computation with various numbers of groups G (*i.e.* defined by `GroupBy` clause). This figure shows the strength of the massively parallel calculation of TDSs when G is small and its limits when G is really too big. We compare next the overhead introduced by our contribution to SQL/AA.

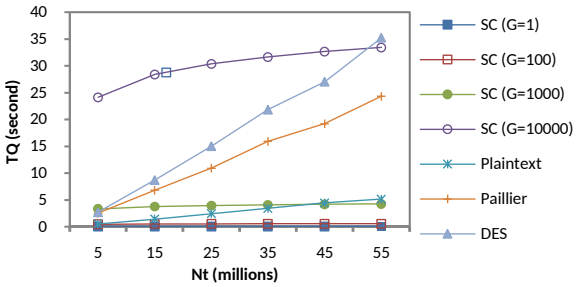

Figure 9: Performance measurements of SQL/AA and state of the art.

Categorical vs. numeric values. We ran a query with one hundred generalization levels, using first categorical, then numerical values. Execution time was exactly the same, demonstrating that the cost of generalizing categorical or numerical values is indifferent.

Collection and Filtering Phases. Figures 10 and 11 show the overhead introduced by our approach, respectively on the collection and filtering phases. The time corresponds to processing every possible generalization of the query presented in Figure 5, which generates the maximal overhead (i.e., worst case) for our approach. The *SQL/AA* bar corresponds to the execution cost of the SQL/AA protocol inside the TDS, the *data transfer* bar corresponds to the total time spent sending the query and retrieving the tuple (approximately 200 bytes at an experimentally measured data transfer rate of 7.9Mbits/sec), the *TDS platform* bar corresponds to the internal cost of communicating between the infrastructure and the TDS (data transfer excluded), and the *Privacy* bar corresponds to the overhead introduced by the kiSQL/AA approach. All times are indicated in milliseconds. Values are averaged over the whole dataset (i.e. 30K tuples).

Collection Phase Analysis. The overhead of the collection phase resides in deciding how to generalize the tuple in order to comply with the local privacy requirements, and the global query privacy constraints. Figure 10 shows that our protocol introduces a 0.25ms overhead for a total average execution time of 3.5ms,

thus under 10% which we consider is a very reasonable cost.

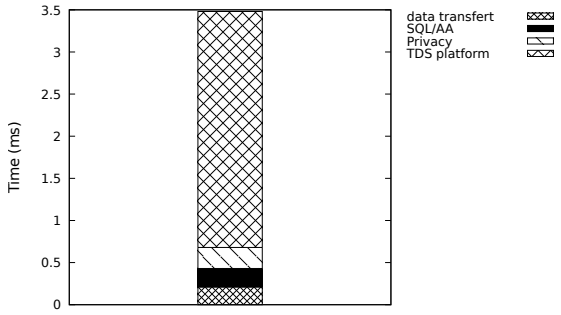

Figure 10: Collection Phase Execution Time Breakdown.

Filtering Phases Analysis. Figure 11 shows breakdown of the filtering phase execution time. The filtering phase takes places once the TDSs have computed all the aggregations and generalizations. The limited resources of the TDSs are bypassed by the SQL/AA system with the help of the (distributed) aggregation phase. Since every group is represented by one tuple, the TDS which computes the filtering phase receives a reduced amount of tuples (called G). To *et al.* have shown that the SQL/AA protocol converges if it is possible for a given TDS to compute G groups during the aggregation phase. As this is the number of tuples that will be processed during the filtering phase, we know that if G is under the threshold to allow its computation via the distributed agregation phase, then it will be possible to compute the filtering phase with our improved protocol. Once again, measurements show that the overhead introduced by ^{k_i}SQL/AA is of only 4% compared to the overall cost of this phase : the overhead introduced is of $0.42ms$ compared to a total cost of $9.8ms$.

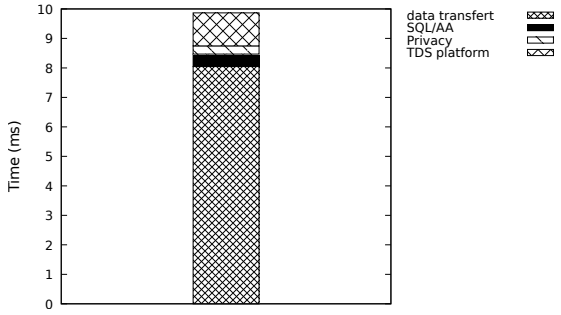

Figure 11: Filtering Phase Execution Time Breakdown.

5 CONCLUSION

In this paper, we presented a novel approach to define and enforce personalized anonymity constraints on SQL GROUP BY queries. To the best of our knowledge, this is the first approach targeting this issue. To this end, we extended the SQL/AA protocol and implemented our solution on secure hardware tokens (TDS). Our experiments show that our approach is clearly useable, with an overhead of a few percent on a total execution time compared with the genuine SQL/AA protocol.

Our current work involves investigating the quality of the anonymization produced, in presence of different anonymization constraints for each individual. We firmly believe that introducing personalized anonymity constraints in database queries and providing a secure decentralized query processing framework to execute them gives substance to the user's empowerment principle called today by all legislations regulating the use of personal data.

REFERENCES

Abiteboul, S., André, B., and Kaplan, D. (2015). Managing your digital life. *Commun. ACM*, 58(5):32–35.

Anciaux, N., Bonnet, P., Bouganim, L., Nguyen, B., Popa, I. S., and Pucheral, P. (2013). Trusted cells: A sea change for personal data services. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*.

Bayardo, R. J. and Agrawal, R. (2005). Data privacy through optimal k-anonymization. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 217–228, Washington, DC, USA. IEEE Computer Society.

Dwork, C. (2006). Differential privacy. In *Proceeding of the 39th International Colloquium on Automata, Languages and Programming*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg.

European Union (2014). ARTICLE 29 DATA PROTECTION WORKING PARTY: Opinion 05/2014 on Anonymisation Techniques.

European Union (2016). Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119/59.

Flajolet, P., Fusy, ï., Gandouet, O., and Meunier, F. (2007). Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *Proceedings of the 2007 International conference on Analysis of Algorithms (AOFA'07)*.

Ge, T. and Zdonik, S. (2007). Answering aggregation queries in a secure system model. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 519–530. VLDB Endowment.

Iyengar, V. S. (2002). Transforming data to satisfy privacy constraints. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 279–288, New York, NY, USA. ACM.

Li, N., Li, T., and Venkatasubramanian, S. (2010). Closeness: A new privacy measure for data publishing. *IEEE Trans. Knowl. Data Eng.*, 22(7):943–956.

Lichman, M. (2013). UCI machine learning repository.

Machanavajjhala, A., Gehrke, J., Kifer, D., and Venkitasubramaniam, M. (2006). l-diversity: Privacy beyond k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 24.

Sweeney, L. (2002). k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570.

To, Q., Nguyen, B., and Pucheral, P. (2014). SQL/AA: executing SQL on an asymmetric architecture. *PVLDB*, 7(13):1625–1628.

To, Q.-C., Nguyen, B., and Pucheral, P. (2016). Private and scalable execution of sql aggregates on a secure decentralized architecture. *ACM Trans. Database Syst.*, 41(3):16:1–16:43.

Trabelsi, S., Neven, G., Raggett, D., Ardagna, C., and et al. (2011). Report on design and implementation. Technical report, PrimeLife Deliverable.