The Case for Personalized Anonymization of Database Query Results

Axel Michel[1,2], Benjamin Nguyen[1,2] and Philippe Pucheral[2]

[1] LIFO, INSA-CVL, Boulevard Lahitolle, Bourges, France
{axel.michel,benjamin.nguyen}@insa-cvl.fr
[2] PETRUS team, INRIA Saclay & DAVID, UVSQ, Versailles, France
philippe.pucheral@inria.fr

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.
In this paper, we consolidate our previous results to show how it is possible to push personalized privacy guarantees in the processing of database queries. By doing so, individuals can disclose different amounts of information (*i.e.* data at different levels of accuracy) depending on their own perception of the risk, and we discuss the different possible semantics of such models.
Moreover, we propose a decentralized computing infrastructure based on secure hardware enforcing these personalized privacy guarantees all along the query execution process. A complete performance analysis and implementation of our solution show the effectiveness of the approach to tackle generic large scale database queries.

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[3].

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

Smart disclosure initiatives, pushed by legislators (*e.g.* EU General Data Protection Regulation [5]) and industry-led consortiums (*e.g.* blue button and green button in the US[4], Midata[5] in the UK, MesInfos[6] 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.

This approach has two important drawbacks:

1. The anonymity guarantee is defined by the querier (*i.e.* the statistical institute), and applied 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 thus 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[7]), 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 empowerment*, a principle called by all recent legislations protecting the

[4] https://www.healthit.gov/patients-families/
[5] https://www.gov.uk/government/news/
[6] http://mesinfos.fing.org/
[7] Yahoo 'state' hackers stole data from 500 million users - BBC News. www.bbc.co.uk/news/world-us-canada-37447016

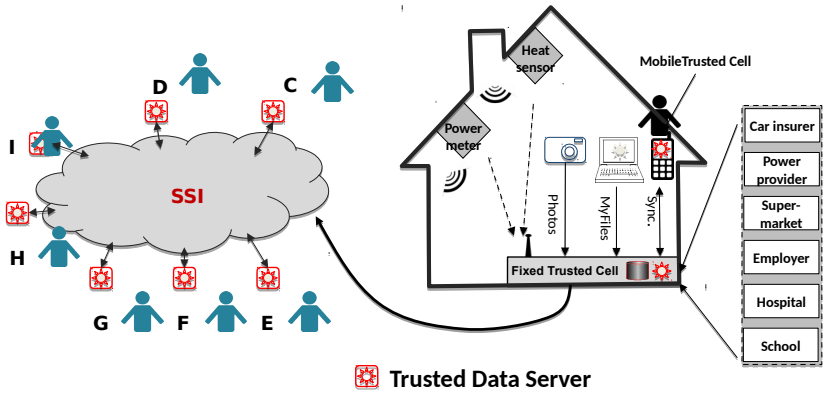

Fig. 1: Trusted Cells reference architecture [2].

management of personal data [5]. Roughly speaking, user empowerment means that the individual must keep the control of her data and of its disclosure in any situation. More precisely, this paper, which is an extended version of [17] makes the following contributions (new contributions in bold):

- propose 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,
- **present and discuss possible semantics of personalized privacy queries**,
- **propose efficient algorithms to manage the collection and distributed computation of queries over large quantities of data**
- provide 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,
- conduct a performance evaluation **and large scale implementation** on a real dataset demonstrating the effectiveness and scalability of the approach.

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 how to manage personalized queries in sql. Section 4 covers a discussion of the semantics of this model. Section 5 discusses the complex algorithmic issues that arise due to the use of personalized anonymity. Section 6 shows the results of our implementation, demonstrating the efficiency of the approach in real case scenarios. Finally, Section 7 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 [19] 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 [19] requires each class to contain at least k indistinguishable records, thus each sensitive data will be associated with at least k records. Many other models have been introduced since 2006, such as ℓ-Diversity [16] or t-Closeness [14]. Each model further constrains 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.

A different concept is *differential privacy*, introduced by Dwork in [6]. Differential privacy is more adapted to interactive query answering. Its' 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. The concept of *sticky policies* presented by Trablesi, Neven, Ragget et al. [22] consists in making a policy about authorization (*i.e.* what the querier can do) and obligation(*i.e.* what the querier must do) which will stick to the user data.

Table 1: 3−anonymous and 2−diverse table [17].

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

The concept of personalized privacy was first introduced by Gedik and Liu [9]. They proposed an algorithm to provide location based services with privacy guarantees in a mobile network. Each users can specify the privacy parameter which a third party will ensure when querying the location service. Their algorithm is based on the *Clique* problem and called the Clique-Cloak algorithm. A graph is created linking *neighbour* users together (*i.e.* when they emit a query) and when a clique is identified, the server sends the clique to the location service. The result received by the location service is then sent to each mobile which filters the result to get the information respectively expected by each user. Other works [18,3] follow the same principle and propose different algorithms providing personalized privacy associated to location based services. Mokbel Mohamed F. et al. proposed the *Casper* [18] algorithm. The idea is to keep up to date location data in a trusted third party server called anonymization server. This anonymization server maintains multiple copies of the data at different levels of granularity forming a pyramid with fine grained data at the base and coarse grained data at the top. When a user queries a location based service, the anonymization server sends back the appropriate cell ensuring the $k-$anonymity guarantee chosen by the user. The result is finally filtered by the user mobile or the anonymization server. Bamba Bhuvan et al. introduced the principle of *PrivacyGrid* [3]. The PrivacyGrid algorithm partitions location map into rectangular cells forming a grid. Users locations are kept up to date on the grid and, when a user queries the location service, the user cell is merged with neighbour cells to satisfy the expected k parameter. These works are all based on a trusted third party which ensures the privacy guarantees. Compared to these works, the approach presented in this paper proposes a solution providing personalized privacy not dedicated to location data and without the use of a trusted (centralized) third party. Our approach is based on pre-anonymizing - any form of - data at their source, that is directly at collection time, following the user's privacy recommandations. The queries on the collected dataset are then computed in a decentralized way without the intervention of any trusted (centralized) third party. Participants in this decentralized secure protocol further group data of various individuals to match the privacy guarantees announced by the querier. Note that contrary to location based services, computing real-time queries is not the concern here.

Personalized differential privacy has been first presented by Zach Jorgensen et al. in [11]. They proposed two mechanisms where each user specifies a personalized ϵ and where the result satisfies all users constraints. The first mechanism is naive and applies differential privacy with the greatest ϵ constraint. Their second mechanism is based on a non-uniform sampling algorithm using two parameters. The first parameter is an ϵ given by each user ensuring a personalized differential privacy and the second is a global ϵ ensuring a minimum degree of privacy. Another personalized differential privacy algorithm has been presented by Haoran Li et al. in [13]. They use a partitioning-based mechanism to achieve personalized differential privacy. Their algorithm consists in partitioning the dataset by grouping user data having close ϵ constraints and then applying an algorithm which ensures the differential privacy with the lowest ϵ value of the partition (*i.e.* the highest privacy guarantee).

Another approach is to give the possibility to the user to personalize the generalization of her sensitive attributes. Xiaokui Xiao and Yufei Tao introduce in their article [24] the concept of guarding node. Each user can specify a guarding node (*i.e.* a node on the taxonomy tree of the sensitive attribute) which limits the precision of the sensitive value. This guarantees to users that they cannot be linked to any sensitive value below the guarding node in the taxonomy tree with a given probability. So the sensitive attribute can be generalized instead of generalizing more than enough quasi-identifiers. This provides a better quality to the overall anonymized data with a guaranteed probability of identification. In this paper, we also exploit the idea of giving a greater importance to the sensitive attributes.

2.2 Reference Computing Architecture

Concurrently with smart disclosure initiatives, the *Personal Information Management System* (PIMS) paradigm has been conceptualized [1], 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 [2], 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

community. *SQL/AA* (SQL Asymmetric Architecture) is a protocol to execute standard SQL queries on the Trusted Cells architecture [20,21]. 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 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 [20,21,17] for a more detailed presentation of the SQL/AA protocol.

2.4 Problem Statement

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.

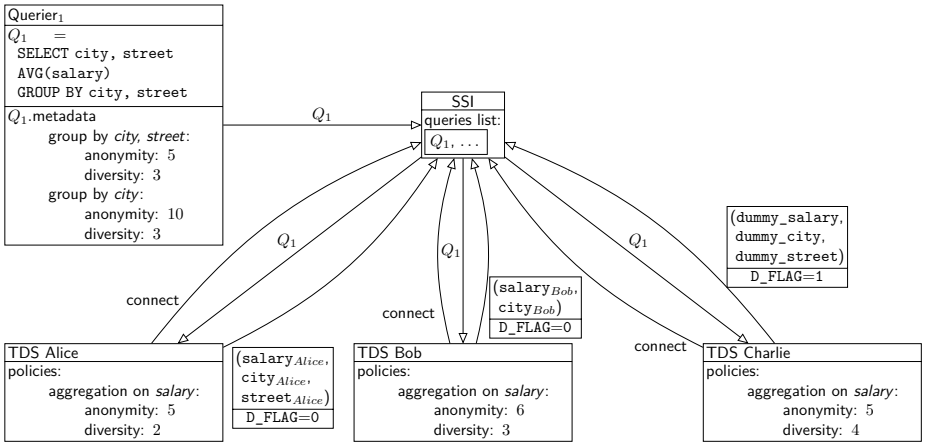

Fig. 2: Example of collection phase with anonymity constraints [17].

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 3.

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

Fig. 3: Regular SQL query form [17].

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 [20,21] 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 2, 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 2, 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*[8] 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 2, 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` clause of the query.

Figure 2 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*[9] 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 [20,21] 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

[8] https://www.w3.org/P3P/
[9] https://www.w3.org/community/odrl/

Table 2: Filtering phase [17].

(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

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)` [10] are computed in addition to the aggregation asked by the querier. As explained next, these clauses are used respectively to check the $k-$anonymity and $\ell-$diversity guarantees). 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 tuple `(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). In this same example, note that another choice could have been done regarding tuple `(Le Chesnay, ******)`. Instead of removing it because it does not meet the privacy constraints, it could

[10] 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* [7].

have been merged with tuple `(Le Chesnay, Dom. Voluceau)`. The result would loose in preciseness but will gain in completeness. We discuss more deeply the various semantics which can be associated to personalized anonymization in Section 4.

How to generalize. The example pictured in Table 2 illustrates data generalization in a simplistic case, that is a Group By query performed on a single attribute which can be expressed with only two levels of preciseness `<City,Street>` and `<City>`. In the general case, Group By queries can be performed on several attributes, each having multiple levels of preciseness. 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) ? The querier must indicate to the TDSs how to generalize the data to best meet her objectives, by indicating which attributes to generalize, in which order, and what privacy guarantees will be enforced after each generalization. In the following example, we consider the UCI Adult dataset [15], 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;
```

Fig. 4: ^{k_i}SQL/AA query example[17].

4 Semantic Issues Linked to K_i-Anonymisation

The example presented in Table 2 highlighted the fact that no single personalized anonymization semantics fits all situations. Typically, personalized anonymization introduces an interesting tradeoff between preciseness and completeness of the query results. Figure 5 illustrates this tradeoff and shows that two different result tables might both respect the privacy constraints, while containing different tuples. In other words, it is important to be able to define the formal characteristics that can be guaranteed, even though some are contradictory. We call *semantics* of a k_i-anonymised query the rules that lead to publishing tuples under privacy guarantees. These semantics are evaluated during the filtering phase of the query.

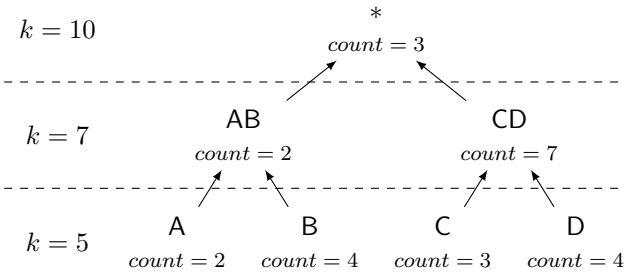

Fig. 5: Post-aggregation phase example

Let Q be a query computing an aggregation on a set of tuples, grouping on an attribute a of finite domain $dom(a)$. We note $|dom(a)|$ the size of the domain. For the sake of simplicity, we assume that the querier specifies the generalization process by means of a labeled generalization *tree* (thus each value has only one parent) which explains how each value must be generalized, and which indicates the privacy guarantees enforced at each generalization level of a.

We plot on Figure 5 an example where $dom(a) = \{A, B, C, D\}$. The example represents the post-aggregation result of query Q (before the filtering phase). For readability purpose, the k privacy guarantee labels are indicated on the left. Each node shows the total number of participants in the given aggregation group. It is important to note that at this point, a microdata tuple will only have contributed to a *single* group, depending on its privacy preferences.

Example: Group AB is composed of the tuples of 2 participants, with values A or B for a, and with a privacy guarantee of $k \geq 7$. Clearly as only 2 participants contributed to the group in this case, this guarantee cannot be enforced, and thus this group cannot be published *as* is in the resulting query answer.

Different decisions, which will impact the semantics of the Group By query, can be made regarding this group: (i) discard the group, and the corresponding microdata tuples, (ii) merge it with a more general group (*i.e.* a parent node

in the generalization tree), (iii) add tuples from a more specific but compatible group to it. We discuss next different realistic semantics for the Group By query and how to attain them. Two concurrent objectives can actually be pursued: generalizing the microdata tuples the least possible or dropping the least possible microdata tuples. Note that when publishing the results of such queries, although a microdata tuple is only counted once, the domains of the groups can overlap.

1. ***Selective* semantics.** These semantics prioritize the precision of the result over its completeness, while enforcing all privacy guarantees. In other words, the querier wants to keep the best quality groups as possible, that is at the finest generalization level, and accepts to drop groups that do not achieve the privacy constraints.
2. ***Complete* semantics.** Conversely, these semantics prioritize the completeness of the result over its precision. The querier wants to keep the most microdata tuples as possible, and accepts to generalize and merge groups to achieve this objective.

Selective semantics are rather straightforward since they correspond to a locally optimisable algorithm. They can be implemented as the algorithm 1.

Algorithm 1 *Selective* algorithm

```
procedure selective(List L)
    while N = L.pop() do
        if satisfyPrivacy(N) then
            publish(N)
        else
            if hasParent(N) then
                N.mergeTo(parent(N))
            end if
        end if
    end while
end procedure
```

With this algorithm 1, the microdata are generalized along the generalization tree until the group cardinality reaches the privacy guarantee of a given node and are then published with this level of precision. Once a node content is published, it cannot be used by a higher level node to help it get published. Thus a node N cannot be discarded if a higher level node N' is published.

Example: Groups A and B do not respect the privacy constraint, they are thus merged with Group AB. The same holds for groups C and D which are merged with group CD. Groups AB and CD (including the newly generalized tuples) are both published. Group $*$ does not respect the privacy constraint and is discarded.

Complete semantics are more difficult to achieve. Indeed, the objective here is to globally optimize the number of tuples published, and only then look at the

quality of the generalization. The problem can be formalized as a constrained optimization problem with two (ordered) objective functions. F_1 is the priority optimization function which counts the number of microdata tuple published. F_2 is the secondary optimization function which evaluates the overall quality of groups. Functions F_2 could be defined to take into account the semantic relatedness among nodes of the generalization tree by using appropriate metrics (*e.g.* Wu and Palmer[23] metric), but for the sake of simplicity we chose to define the overall quality as the $\sigma_g \in \{groups\}(depth(g) \cdot count(g))$. Maximal quality is obtained if all the microdata tuples are published at maximal depth. Dropped tuples induce maximal penalty in quality, since they do not contribute to the score. Also note that it is not possible to generalize only a subset of the microdata tuples composing one of the initial groups. A group must be generalized completely or not at all. However, descendant groups could be generalized more than some of their ancestor groups. For instance, group CD might be published as is since its already satisfies the privacy guarantee, and groups C and D might be generalized to $*$ to form a group of cardinality 10 which can also be published (instead of dropping the three initial elements in $*$). Hence, regarding F_1, various combinations are valid like $\{(*, count = 10), (AB, count = 8), (CD, count = 7)\}$, $\{(*, count = 18), (CD, count = 7)\}$ or $\{(*, count = 11), (CD, count = 14)\}$. In this example, F_2 should lead to select the former one.

The practical difficulty comes precisely from the computation of this secondary optimization function F_2 (not to mention having to chose what function to use). Indeed, otherwise a trivial answer would simply to generalize all the microdata tuples to $*$, and publish a single group. In a centralized context, a (non linear) constraint solver could be used to compute the optimal result to this problem. However, in a distributed context, generalization must take place during the distributed filtering phase. Thus we propose to use a greedy heuristic to simplify decision making in the *complete* semantics approach.

The difference between these two algorithms is that the second algorithm accepts to generalize a node which could be published as is, if it can help a higher node to get published. Also note that another semantic choice that must be made when implementing the algorithm is when deciding which descendant node to merge, when several possibilities exist.

Example: The algorithm behaves the same as the *selective* algorithm until it has groups AB and CD in R. Then it must decide whether to merge AB or CD with $*$. The decision can be made by chosing the smallest group that still respects the privacy constraint. In this case, as $|AB| = 8$ and $|CD| = 14$, we would chose to merge AB with $*$ and produce $|*| = 11$ and $|CD| = 14$. However we see that the optimal solution would have been to merge C and D with $*$ or merge CD with $*$ and generalize C and D to CD, thus producing $|AB| = 8$, $|CD| = 7$ and $|*| = 10$.

Conclusion: Several different semantics can be applied in order to decide how to publish the groups while respecting privacy constraints. There is no ideal choice in absolute terms and the decision should be application driven. However, besides the preciseness and completeness of the obtained result, the cost of im-

Algorithm 2 Greedy *complete* algorithm.

```
procedure complete(Ordered List (lower node first) L)
    R ←empty list
    while N = L.pop() do
        if satisfyPrivacy(N) then
            R.push(N)
        else if hasDescendant(N) then
            P ← descendant(N)
            R.remove(P)
            P.mergeTo(N)
        else if hasParent(N) then
            N.mergeTo(parent(N))
        end if
        publish(R)
    end while
end procedure
```

plementing these semantics may widely differ. While implementing the *selective* semantics is straightforward, optimizing the computation of the *complete* semantics can be rather complex and costly, especially in a decentralized context. It is part of our future work to propose and evaluate greedy algorithms tackling this challenge.

5 Algorithmic Issues Linked to K_i-Anonymisation

The implementation of ^{k_i}SQL/AA builds upon the *secure aggregation* protocol of SQL/AA [20,21], 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. Hence, all TDSs, regardless of whether they contributed to the collection phase or not, can participate to a distributed computation and decrypt all intermediate results produced by this computation. without revealing any information to the SSI. Indeed, the SSI cannot infer personal informations by the distribution of same encrypted values. Injecting personnalization in this protocol has no impact on the cryptographic protection. However, this impacts the internal processing of each phase of the protocol, as detailed next.

5.1 Collection phase

The confrontation between the querier's privacy guarantees and the TDS's privacy constraints, along with the potential data generalization resulting from this confrontation, are included in the collection phase. The resulting algorithm is given below (see Algorithm 3). 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 3 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
```

5.2 Aggregation phase

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. The aggregation phase, as implemented in SQL/AA [20,21], is illustrated by the Figure 6.

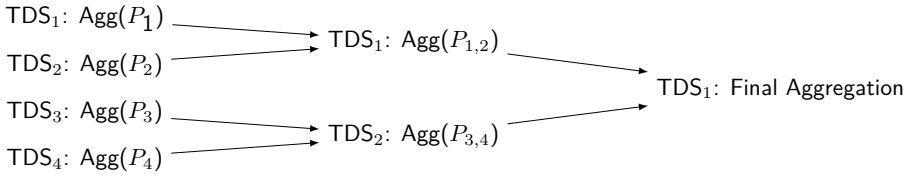

Fig. 6: Aggregation phase with four partitions.

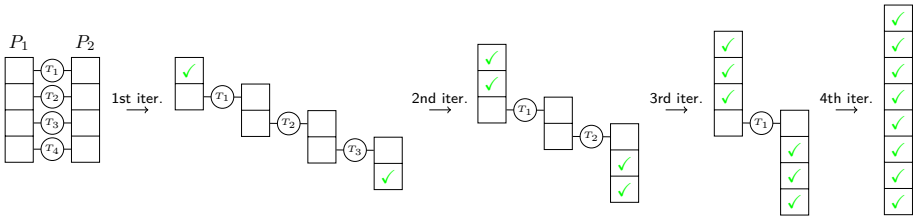

Fig. 7: Merging sorted partitions without NAND access

The weakness of this way of computing the aggregation is the working load put on the unique TDS performing the final aggregation.The limited amount of TDSs RAM reduces the number of different groups (*i.e.* given by the `GroupBy` clause) which can be managed by a single TDS. Moreover, the larger the partitions to merge, the larger the time spent by that TDS to do the computation and the greater the burden put on the TDS owner who cannot easily perform other tasks in parallel. This limitation is magnified in our context since personalized anonymization may lead a wider number of groups since initial groups may appear at different granularity levels.

We propose two algorithms overcoming this limitation. The first algorithm, called the *swap-merge* algorithm, is a direct adaptation of the SQL/AA aggregation algorithm where partitions bigger than the TDS RAM are simply swapped in NAND Flash. When it comes to computes aggregations on two sorted partitions saved on NAND Flash, the TDS will use a simple merge sort to build the result partition. The corresponding algorithm is straightforward and not detailed further due to space limitation. The benefit of this adaptation is to accomadate an unlimited amount of groups while keeping logarithmic the number of aggregation phases. However, this benefit comes at the price of ever increasing the load imposed on TDSs at the root of the aggregation tree due to NAND Flash I/Os. Note that executing the initial SQL/AA aggregation algorithm in streaming to avoid resorting to NAND swapping at each TDS would reveal the tuple ordering to the SSI. This option is then discarded.

The second algorithm, illustrated by the figure 7, does not rely on NAND Flash swapping and is called the *network-merge*. The idea is to use a bubble sort like algorithm to merge two sorted partitions.

The figure 7 illustrates the merge of two sorted partitions P_1 and P_2. We assume that each partition is in turn decomposed into n frames ($n = 4$ in the figure) in order to accomodate the RAM capacity of a TDS. More precisely, the maximal size of a frame is set to the half of the TDS RAM size (so that two frames of two different partitions can fit in RAM and be merged without swapping), minus a RAM buffer required to produce the merge result. Let P_i^j denote the j^{th} frame of partition P_i and let P_r denote the partition resulting from the merge of P_1 and P_2. Assuming that P_1 and P_2 are internally sorted in ascending order on the grouping attributes, the merge works as follows. For each

$j = 1..n$, the j^{th} frames of P_1 and P_2 are pairwise sent to any merger TDS[11] which produces in return the j^{th} and $(j+1)^{th}$ sorted frames of P_r. All elements in P_r^j are smaller or equal to the elements of P_r^{j+1}. By construction, after the first iteration of the protocol, frame P_r^1 contains the smallest elements of P_r and frame P_r^n the greatest ones. Hence, these two frames do not need to participate to the next iterations. Hence, after k iterations, $2 \cdot k$ frames of P_r are totally sorted, the smallest elements being stored in frames $P_r^{1..k}$ and the greatest ones being stored in frames $P_r^{n-k..n}$. The algorithm, presented in 4 is guaranteed to converge in a linear number of iterations. For readability concern and without loss of generality, we suppose that P_1 and P_2 have the same size.

Algorithm 4 Partition Merger Algorithm

```
procedure NETWORK-MERGE(Partition P1, Partition P2, Command cmd)
    Let top, bot, mid, q1, q2 empty partitions
    for i from 1 to NumberOfFrame(P1) do
        Send(P1^i, P2^i, cmd) to TDS_i
    end for
    for i from 1 to NumberOfFrame(P1) do
        P_r ← receive() from TDS_i
        if i == 1 then
            top.append(P_r^1)
        else
            q1.append(P_r^1)
        end if
        if i == NumberOfFrame(P1) then
            bot.append(P_r^2)
        else
            q2.append(P_r^2)
        end if
    end for
    if NotEmpty(q1) then
        mid ← NETWORK-MERGE(q1, q2, cmd)
    end if
    return top + mid + bot
end procedure
```

Hence, algorithm *network-merge* can accomodate any number of groups without resorting to NAND Flash swapping. The price to pay is a linear (instead of logarithmic for *swap-merge* algorithm) number of iterations. The negative impact on the latency of the global protocol is low compared to the latency of the collection phase itself. But the benefit is high in terms of TDS availability.

[11] each TDS can contribute to any phase of the protocol, depending on its availability, independently of the fact that it participated to the collection phase.

Thanks to this algorithm, each participating TDS contributes to a very small part of the global computation and this part can additionally be bounded by selecting the appropriate frame size. This property may be decisive in situations where TDSs are seldom connected.

5.3 Filtering Phase

The filtering phase algorithm is given by Algorithm 5.

Algorithm 5 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
```

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.

6 Experimental Evaluation

We have implemented the ^{k_i}SQL/AA protocol on equivalent open-source hardware used in [20,21]. 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 [15]

6.1 Experiments Platform

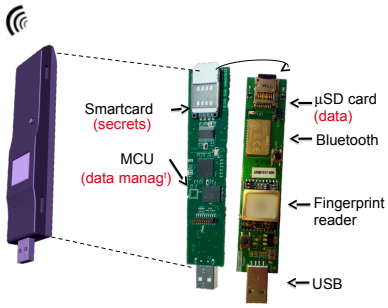

Fig. 8: TDS characteristics [12].

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 8. 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[12], 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 [15]. This dataset is an extraction from the American census bureau database. We modified the dataset the same way of [10,4]. 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 9. The native country attribute is the largest and requires 49 bits to be represented.

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

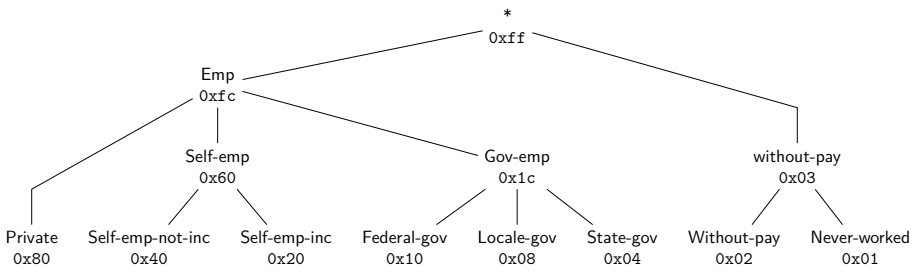

Fig. 9: Generalization tree of `workclass` attribute [17].

6.2 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 anonymization 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 [21] 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 10. The Paillier curve shows the performance to compute aggregation in a secure centralized server using homomorphic encryption, presented in [8]. 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.

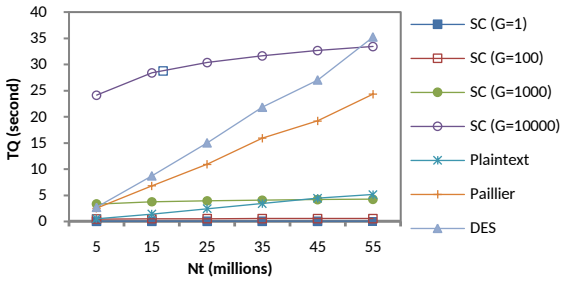

Fig. 10: Performance measurements of SQL/AA and state of the art [21].

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 11 and 12 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 4, 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 ^{k_i}SQL/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 11 shows that our protocol introduces about a low overhead (between 0.29ms and 0.40ms). The variation is due to the number of generalization to be made. Most of the time is taken by the DBMS running in the TDS since it takes privacy preference on the NAND memory which is slow. The time taken by the SQL/AA system is almost due to the query to get data on the NAND memory which is *constant*. The same query could have been kept but we used the minimal query size to show this variation. This performances are a bit different from the article [17] since we used more complicated queries on the TDS DBMS. Our contribution introduces an overhead under 10% of the overall time which we consider as a very reasonable cost.

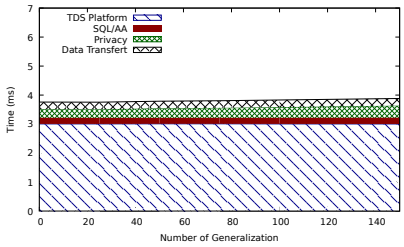

Fig. 11: Collection Phase Execution Time Breakdown.

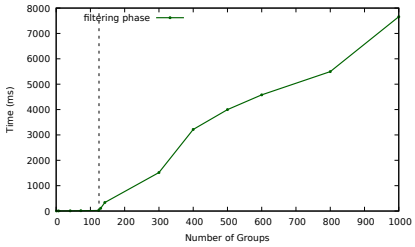

Fig. 12: Filtering Phase Execution Time Breakdown.

Filtering Phases Analysis. Figure 12 shows breakdown of the filtering phase execution time. The filtering phase takes place 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 aggregation phase, then it will be possible to compute the filtering phase with our improved protocol. To make performance measures, the cortex-M4 has a 24bit system timer giving the possibility to measure time lower than 100ms. Since computations now use NAND access, the time is too high to give a comparison between the SQL/AA system and the ^{k_i}SQL/AA system. The dashed line shows the limit where resorting to NAND is mandatory.

Aggregation phase Analysis. Figure 13 shows the performance measures of the sorting algorithm used in the aggregation phase. After the aggregation phase, the result contains as much tuples as different groups made by the `GroupBy` clause. We vary the number of groups to increase or decrease the overall time taken by each step of the merge algorithm. The limit of the SQL/AA system is shown by the dashed vertical line. The system was unable to compute a query with more groups than this limit. The merge algorithm permits to overcome this limit. The performance measure was done on one TDS. Since each step of the merge sort can be distributed, the maximum time of each partition merge was kept and summed between the different steps to get the time showed by the figure 13. The performance of the swap merge drastically decreases when the data to aggregate cannot fit in the RAM of the TDS and needs to be swapped on the NAND memory.

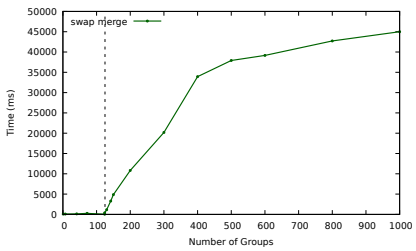

Fig. 13: Aggregation Phase Execution Time Breakdown.

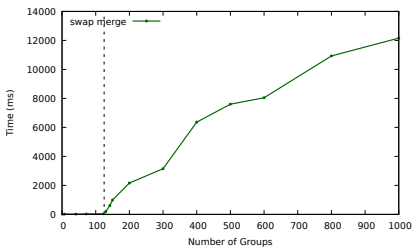

Fig. 14: TDS Availability Required By The Aggregation Phase.

One of the weakness of the swap merge algorithm is the time a TDS must be available to merge two partitions. This time is directly related to the number of distinct groups generated by the `GroupBy` clause. It is important to measure this time. The maximum time a TDS must be available to compute the aggregation phase is illustrated by the figure 14. The time increases drastically when the TDS needs to use the NAND memory.

7 Conclusion

While privacy-by-design and user empowerment are being consecrated in the legislations regulating the use of personal data (e.g., EU General Data Protection Regulation), time has come to put these principles into practice. This paper tackles this objective by proposing a novel approach to define personalized anonymity constraints on database queries. The benefit is twofold: let the individuals control how their personal data is exposed ever since collection time and, by this way, provide the querier with a greater set of consenting participants and more accurate results for their surveys. Moreover, we propose a fully decentralized and secure execution protocol enforcing these privacy constraints, which avoids the risk of centralizing all personal data on a single site to perform anonymization. Our experiments shows the practicality of the approach in terms of performance. Finally, this paper shows that different semantics can be attached to the personalized anonymization principle, opening exciting research perspectives for the data management community.

References

1. Abiteboul, S., André, B., Kaplan, D.: Managing your digital life. Commun. ACM 58(5), 32–35 (Apr 2015), http://doi.acm.org/10.1145/2670528
2. Anciaux, N., Bonnet, P., Bouganim, L., Nguyen, B., Popa, I.S., Pucheral, P.: 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 (2013)
3. Bamba, B., Liu, L., Pesti, P., Wang, T.: Supporting anonymous location queries in mobile environments with privacygrid. In: Proceedings of the 17th International Conference on World Wide Web. pp. 237–246. WWW '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1367497.1367531
4. Bayardo, R.J., Agrawal, R.: Data privacy through optimal k-anonymization. In: Proceedings of the 21st International Conference on Data Engineering. pp. 217–228. ICDE '05, IEEE Computer Society, Washington, DC, USA (2005), http://dx.doi.org/10.1109/ICDE.2005.42
5. 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) (Text with EEA relevance), http://data.europa.eu/eli/reg/2016/679/oj
6. Dwork, C.: Differential privacy. In: Proceeding of the 39th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 4052, pp. 1–12. Springer Berlin / Heidelberg (2006)
7. Flajolet, P., Fusy, ï., Gandouet, O., Meunier, F.: Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In: Proceedings of the 2007 International conference on Analysis of Algorithms (AOFA'07) (2007)
8. Ge, T., Zdonik, S.: Answering aggregation queries in a secure system model. In: Proceedings of the 33rd International Conference on Very Large Data Bases. pp. 519–530. VLDB '07, VLDB Endowment (2007), http://dl.acm.org/citation.cfm?id=1325851.1325912

9. Gedik, B., Liu, L.: Location privacy in mobile systems: A personalized anonymization model. In: 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05). pp. 620–629 (June 2005)
10. Iyengar, V.S.: Transforming data to satisfy privacy constraints. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 279–288. KDD '02, ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/775047.775089
11. Jorgensen, Z., Yu, T., Cormode, G.: Conservative or liberal? personalized differential privacy. In: 2015 IEEE 31st International Conference on Data Engineering. pp. 1023–1034 (April 2015)
12. Lallali, S., Anciaux, N., Sandu Popa, I., Pucheral, P.: A secure search engine for the personal cloud. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. pp. 1445–1450. SIGMOD '15, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2723372.2735376
13. Li, H., Xiong, L., Ji, Z., Jiang, X.: Partitioning-Based Mechanisms Under Personalized Differential Privacy, pp. 615–627. Springer International Publishing, Cham (2017), https://doi.org/10.1007/978-3-319-57454-7_48
14. Li, N., Li, T., Venkatasubramanian, S.: Closeness: A new privacy measure for data publishing. IEEE Trans. Knowl. Data Eng. 22(7), 943–956 (2010), http://dx.doi.org/10.1109/TKDE.2009.139
15. Lichman, M.: UCI machine learning repository (2013), http://archive.ics.uci.edu/ml
16. Machanavajjhala, A., Gehrke, J., Kifer, D., Venkitasubramaniam, M.: 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. p. 24 (2006), http://dx.doi.org/10.1109/ICDE.2006.1
17. Michel, A., Nguyen, B., Pucheral, P.: Managing distributed queries under personalized anonymity constraints. In: DATA 2017, Sixth International Conference on Data Science, Technology and Applications (2017)
18. Mokbel, M.F., Chow, C.Y., Aref, W.G.: The new casper: Query processing for location services without compromising privacy. In: Proceedings of the 32Nd International Conference on Very Large Data Bases. pp. 763–774. VLDB '06, VLDB Endowment (2006), http://dl.acm.org/citation.cfm?id=1182635.1164193
19. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10(5), 557–570 (2002)
20. To, Q., Nguyen, B., Pucheral, P.: SQL/AA: executing SQL on an asymmetric architecture. PVLDB 7(13), 1625–1628 (2014), http://www.vldb.org/pvldb/vol7/p1625-to.pdf
21. To, Q.C., Nguyen, B., Pucheral, P.: Private and scalable execution of sql aggregates on a secure decentralized architecture. ACM Trans. Database Syst. 41(3), 16:1–16:43 (Aug 2016), http://doi.acm.org/10.1145/2894750
22. Trabelsi, S., Neven, G., Raggett, D., Ardagna, C., et al.: Report on design and implementation. Tech. rep., PrimeLife Deliverable (May 2011)
23. Wu, Z., Palmer, M.: Verbs semantics and lexical selection. In: Proceedings of the 32Nd Annual Meeting on Association for Computational Linguistics. pp. 133–138. ACL '94, Association for Computational Linguistics, Stroudsburg, PA, USA (1994), https://doi.org/10.3115/981732.981751
24. Xiao, X., Tao, Y.: Personalized privacy preservation. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. pp. 229–240. SIGMOD '06, ACM, New York, NY, USA (2006), http://doi.acm.org/10.1145/1142473.1142500