Secure Distributed Computation of Anonymized Views of Shared Databases

Tamir Tassa
Department of Mathematics and Computer Science
The Open University
Ra'anana, Israel

Ehud Gudes
Department of Computer Science
Ben Gurion University
Be'er Sheva, Israel

Abstract

We consider the problem of computing efficient anonymizations of partitioned databases. Given a database that is partitioned between several sites, either horizontally or vertically, we devise secure distributed algorithms that allow the different sites to obtain a k-anonymized and ℓ-diverse view of the union of their databases, without disclosing sensitive information. Our algorithms are based on the sequential algorithm [20] that offers anonymizations with utility that is significantly better than other anonymization algorithms, and in particular those that were implemented so far in the distributed setting. Our algorithms can apply to different generalization techniques and utility measures and to any number of sites. While previous distributed algorithms depend on costly cryptographic primitives, the cryptographic assumptions of our solution are surprisingly minimal.

Categories and subject descriptors.

- H. Information Systems, H.2 Database management, H.2.4 Systems, Distributed databases
- H. Information Systems, H.2 Database management, H.2.8 Database Applications, Data mining
- K. Computing Milieux, K.4 Computers and society, K.4.1 Public Policy Issues, Privacy

General Terms. Algorithms, Design, Experimentation, Security

Keywords. k-anonymization, privacy preserving data mining, secure multi-party computation, distributed data mining

1 Introduction

Recent advances in information technology enable more organizations to collect, store, and use various types of information on individuals. Such large repositories of data carry valuable information that may be extracted using data mining tools. In such settings, protecting the privacy of the individuals whose private data appear in those repositories is of paramount importance. Although identifying attributes such as names and ID numbers are always removed before releasing the table for data mining purposes, sensitive information might still leak due to *linking attacks*; such attacks may join the public attributes, a.k.a *quasi-identifiers*, of the published table with a publicly accessible table like the voters registry, and thus disclose private information of specific individuals. Privacy-preserving data mining [5] has been proposed as a paradigm of exercising data mining while protecting the privacy of individuals. One of the most well-studied models of privacy preserving data mining is k-anonymization [39, 40]. The usual practice in k-anonymization is to generalize or suppress the values of the public attributes, so that each of the released records becomes indistinguishable from at least $k-1$ other records, when projected on the subset of public attributes, thus hiding its relationship with the values of the sensitive attribute. As a consequence, each individual may be linked to sets of records of size at least k in the released anonymized table, whence privacy is protected to some extent.

k-Anonymity on its own is not sufficiently secure since the distribution of the sensitive data within a given block of indistinguishable records may reveal information on the sensitive data of a given target individual. Hence, k-anonymity must be enforced in conjunction with other measures, such as ℓ-diversity [32, 45, 46], t-closeness [31], or p-sensitivity [43], that limit the amount of information that is leaked by the distribution of the sensitive data in each block of indistinguishable records. All of those notions offer essential *enhancements* to k-anonymity in the sense that one must require them *in addition* to k-anonymity. In accord with this, Truta et al. [43] propose algorithms that generate tables that are both k-anonymous and p-sensitive, and Wong et al. [44] consider the conjunction of k-anonymity with ℓ-diversity (they call this conjunction of conditions $(1/\ell, k)$-anonymity). In this study we discuss algorithms for achieving k-anonymity together with ℓ-diversity.

Given a measure of information loss that associates a penalty to each operation of generalizing a table entry, the k-anonymity problem seeks a k-anonymized view of a given table with minimal loss of information (namely, maximal utility). That problem was shown to be NP-hard [3, 33] and several approximation algorithms [3, 19, 26, 33, 38] and heuristics, e.g. [17, 18, 20, 22, 30, 42], were proposed for obtaining k-anonymizations with high utility.

In some settings, the data is split between several data holders. For example (the case of horizontal partitioning), several hospitals may wish to combine their databases of patients in order to conduct a medical research on their unified population of patients. However, as each hospital is committed to the privacy of its own patients, it is impossible to simply unify the databases and then apply on the unified database an anonymization algorithm. As another example (the case of vertical partitioning), the data may be split into different sites according to its type. For instance, one site may hold the medical information of an individual, another site may hold his credit history, and a third site may hold relevant demographic information. In either of those cases, it is needed to devise a protocol that would allow each hospital to perform anonymization of its own database, without revealing its original content to the collaborating hospitals, so that the union (or join) of all anonymized tables will respect the required privacy measure.

In this paper we discuss the problem of computing high-utility anonymizations of distributed databases. In an ideal setting, in which there is a trusted third party, each site could surrender to that third party his part of the database and trust the third party to compute an anonymization of the unified database. Without such a trusted third party, the goal is to devise distributed protocols, for the horizontal and vertical settings, that allow the data holders to simulate the operation of a trusted third party and obtain a k-anonymized and ℓ-diverse view of the union of their databases, without disclosing unnecessary information to any of the other parties, or to any eavesdropping adversary.

1.1 About the relevance of clustering-based anonymity models

In recent years, other measures of privacy were proposed as more secure alternatives to the above mentioned clustering-based models, the most prominent of which is differential privacy [12]. However, albeit theoretically more secure, differential privacy and similar theoretical notions (such as (ε, δ)-indistinguishability [37]) are still far from obsoleting the clustering-based approaches, due to several problems. First, it is much harder to achieve privacy according to those notions in practice for non-trivial queries. Second, as opposed to the clustering-based approaches, such perturbation-based methods need to know the query in advance in order to calibrate the level of noise to the global sensitivity of the query and the targeted differential privacy parameter ε [13]. Third, the theoretical study of those notions did not address, so far, the practically essential question of how to set the privacy parameter ε in order to provide a satisfactory level of privacy in a given practical setting. Moreover, in some disciplines, such as biostatistics or biomedical research, it is imperative to work on sanitized data whose correlation with the original data is known [15]; this is the case with clustering-based anonymization models, in which data is generalized according to accepted generalization rules; this is not the case with perturbation models in which the correlation between the original and perturbed data is probabilistic. Because of those reasons, clustering-based privacy models are still perceived by practitioners as sufficient for mitigating risk in the real world while maximizing utility, and real life applications still utilize them for sanitizing data (see [14, 15]).

Another reinforcement to the relevance of clustering-based anonymity models was proposed recently by Cormode in [10]. He examined the effects of the deFinetti attack [27] on clustering-based methods of anonymization. He then designed a similar inference-based attack on differentially private releases of the same databases and found that the susceptibility of those two classes of privacy notions to such attacks is quite similar. His conclusion was that "rejecting all such (clustering-based) anonymizations because the deFinetti attack exists is erroneous: by the same logic, we should also abandon differential privacy". His final conclusion, with which we concur, is that depending on the perceived threats, and the consequences of a successful attack, it may be appropriate to use deidentification, clustering-based methods, differential privacy, or to withhold release entirely.

1.2 Related work

The problem that we wish to solve is a problem of Secure Multiparty Computation (SMC hereinafter). In the most general setting of SMC, there are m interacting players, $P_1, \ldots, P_m$, each one holding a private input x_i, $1 \leq i \leq m$, and they wish to compute the value of a publicly known function f on their inputs, i.e. $s := f(x_1, \ldots, x_n)$, so that none of the players learns about the inputs of the other players more than what is implied by the final output and his own input. In our case, the private inputs are the private databases, and the required output is a k-anonymization of the joint database.

Generic solutions of the general problem were presented in [7, 21, 48]. Such solutions require to reformulate the problem as a problem of evaluating a boolean circuit. Since such generic solutions are practical only when the size of the input in bits is small and the description of the function as a boolean circuit is compact, our problem of computing an anonymized view of a distributed database calls for other types of protocols.

This problem was the topic of several studies in the past few years. Zhong et al. [50] devise a secure distributed implementation of the approximation algorithm from [33] for horizontally partitioned databases. The approximation algorithm in [33], as well as its distributed implementation in [50], assumes that only suppressions are allowed and their solution requires homomorphic encryption, group exponentiations, and secret sharing.

Jurczyk and Xiong [24] present a distributed implementation of the Mondrian algorithn [30] for the case of horizontally-partitioned databases. Their solution invokes an SMC protocol for computing the kth ranked element in the union of sets that are held by the different players [4], using cryptographic primitives such as oblivious transfer and pseudorandom functions.

In [23], Jiang and Clifton deal with databases that are vertically-partitioned between two sites. In their method, each player begins by k-anonymizing his partial database, with respect to the subset of quasi-identifiers that he holds. Such anonymizations result in a clustering of the records in each of the two databases into clusters of size at least k. The idea is then to check whether the join of the two anonymized databases is k-anonymous on the unified set of quasi-identifiers. To that end, the two players engage in a secure set intersection protocol for computing the size of the intersection of two private sets. Clusters whose intersection is of size less than k are further generalized until all cluster intersection sizes are either zero or above k. Their protocol for the secure computation of set intersection uses homomorphic encryptions.

Mohammed et al. [34] deal also with vertically-partitioned databases. They implement a top-down specialization algorithm, which is essentially the Mondrian algorithm [30]. Their protocol starts with all database entries fully suppressed. Then, in each step, each player examines the quasi-identifiers that he has in order to find the best one to specialize, where each candidate specialization operation is ranked by a score function that balances between the projected information gain and anonymity loss. In order to find the most profitable specialization operation, a secure maximum protocol [48] is invoked between every two players. The authors discuss extensions of their protocol to support additional privacy measures such as ℓ-diversity.

In a slightly different line of work, Nergiz et al. [35] describe a method to save on SMC protocols in privacy-preserving algorithms such as distributed k-annonymizations. The main idea is to use a Look-Ahead mechanism. At each step of the computation they estimate the utility that may be gained by invoking an SMC protocol at that stage in order to decide whether it is worth-while to invest the computational effort that is entailed by such a protocol. They implement it to horizontally-partitioned databases in which each player has a k-anonymization of his database, and the players wish to compute together a better k-anonymization of the entire database. which is better than the one which may be obtained by simply unifying the separate k-anonymizations.

1.3 Our contributions

We devise secure distributed protocols for obtaining k-anonymized and ℓ-diverse views of shared databases, which are based on the sequential anonymization algorithm that was recently introduced in [20]. Our approach offers the following advantages:

• **Generality.** Our approach applies to both horizontal and vertical partitionings, whereas the approaches in [23, 24, 34, 50] are restricted to only one of those settings. Our approach applies to any number of data sites, where the entailed communication cost depends linearly on the number of sites. The sequential algorithm, on which our protocols are based, may be applied with any utility measure and it applies to any generalization technique (unlike [50] which works with generalization by suppression only, or [24, 34] that work only with global recoding generalizations). Our protocols support additional privacy measures such as ℓ-diversity and ℓ-site diversity.

• **Simplicity.** While previous solutions invoke costly cryptographic primitives such as homomorphic encryptions, oblivious transfers, and group exponentiations, the only cryptographic primitives that we need are an SMC protocol for computing sums, and a secure hash function.

• **Efficiency.** Our protocols are practical and efficient. We analyze the communication complexity of our protocols and then conduct experiments that illustrate the dependence of the communication costs on different parameters in several datasets, both in the horizontal and vertical settings.

• **Privacy.** We analyze the privacy of our distributed protocols and show that, even though they are not perfectly secure in the cryptographic sense, they leak very little and benign information. Such a compromise is widely acceptable (see e.g. [24, 25, 50]) when the information leakage is deemed innocuous and the gain in utility, efficiency, and practicality is significant.

• **Utility.** Perhaps the most important advantage offered by our protocols is the utility of the resulting anonymizations. The sequential algorithm is currently one of the leading k-anonymization algorithms in terms of the utility of its output. In particular, it offers anonymizations with significantly smaller information losses than those offered by the algorithms on which the distributed solutions in [23, 24, 34, 50] are based.

Other contributions of this study are two novel generic SMC protocols. The first one is a simple SMC protocol for the computation of the `AND` (or the `OR`) of private bits held by the different players. That protocol gives rise to a simple protocol for secure computation of set intersections and unions. The second SMC protocol computes the least common ancestor of private nodes in a tree. To the best of our knowledge, that problem was not studied before, and it may be of interest in other applications as well.

1.4 Organization of the paper

In Section 2 we provide the necessary terminology and background; in Section 2.4 we describe the sequential algorithm [20] which serves as the basis for our distributed protocols. In Section 3 we describe the SMC protocols that are at the basis of our distributed anonymization protocols. The distributed k-anonymization protocols are given in Sections 4 and 5, in the horizontal and vertical settings, respectively. In Section 6 we describe the necessary modifications that are required in order to ensure that the output k-anonymizations respect also ℓ-diversity and ℓ-site diversity. (A precise definition of those notions appear in Section 6.) The complexity and privacy of the distributed protocols are analyzed in Sections 7 and 8. We present an experimental evaluation in Section 9. We conclude in Section 10.

2 Preliminaries

In Sections 2.1 and 2.2 we provide the necessary terminology and background about anonymization. In Section 2.3 we discuss the relation between clusterings and anonymizations, as the algorithm that we use here for anonymization is basically a clustering algorithm. In Section 2.4 we describe the sequential clustering algorithm, which is the basis for our distributed protocols. Finally, we describe the distributed setting in Section 2.5.

2.1 Anonymization by generalization

Consider a database that holds information on individuals in some population. Each record in the database has several attributes, and we distinguish between identifiers, quasi-identifiers, and sensitive attributes. Identifiers are attributes that uniquely identify the individual, e.g. `name` or `id`. Quasi-identifiers are attributes, such as `age` or `zipcode`, that appear also in publicly-accessible databases and may be used in order to identify a person. The sensitive attributes are those that carry private information like a `medical diagnosis` or the `salary` of the person. k-Anonymity is a model that was proposed in order to prevent the disclosure of sensitive attributes for the purpose of protecting the privacy of individuals that are represented in the database.

When discussing k-anonymizations, the identifiers are suppressed. Hence, assuming that there are d quasi-identifiers and, for the sake of simplicity, one sensitive attribute, we view the database records as elements in $A_1 \times \cdots \times A_d \times A_{d+1}$, where A_j is the set of possible values for the jth attribute; say, if the jth attribute is `gender` then $A_j = \{M, F\}$.

Hereinafter, D denotes the projection of the database on the set of d quasi-identifiers and the records of D are denoted R_i, $1 \leq i \leq n$; namely, $R_i \in A_1 \times \cdots \times A_d$. We denote the jth component of the record R_i by $R_i(j)$. Also, for any set A we let $\mathcal{P}(A)$ denote its power set. Next, we define the notion of generalization.

Definition 2.1. *Let A_j, $1 \leq j \leq d$, be finite sets and let $\overline{A}_j \subseteq \mathcal{P}(A_j)$ be a collection of subsets of A_j. A mapping $g : A_1 \times \cdots \times A_d \to \overline{A}_1 \times \cdots \times \overline{A}_d$ is called a generalization if for every $(b_1, \ldots, b_d) \in A_1 \times \cdots \times A_d$ and $(B_1, \ldots, B_d) = g(b_1, \ldots, b_d)$, it holds that $b_j \in B_j$, $1 \leq j \leq d$.*

As an example, consider a database D with two attributes, `age` (A_1) and `zipcode` (A_2). A valid generalization of the record $R_i = (34, 98003)$ can be

$$g(34, 98023) = (\{30, \ldots, 39\}, \{98000, \ldots, 98099\}) .$$

We assume here that each of the collections $\overline{A}_j$ is a generalization hierarchy tree for A_j, $1 \leq j \leq d$. Such a tree has $|A_j|$ leaves – one for each singleton subset of A_j; the root corresponds to the whole set; and the subset of each node is the union of the subsets that correspond to the direct descendants of that node.

Definition 2.1 refers to generalizations of single records. We now define generalizations of an entire database.

Definition 2.2. *Let $D = \{R_1, \ldots, R_n\}$ be a database having public attributes $A_1, \ldots, A_d$, let $\overline{A}_1, \ldots, \overline{A}_d$ be corresponding collections of subsets, and let $g_i : A_1 \times \cdots \times A_d \to \overline{A}_1 \times \cdots \times \overline{A}_d$ be corresponding generalization operators. Let $\overline{R}_i := g_i(R_i)$ be the generalization of the record R_i, $1 \leq i \leq n$. Then $\overline{D} = \{\overline{R}_1, \ldots, \overline{R}_n\}$ is a generalization of D. If every generalized record in $\overline{D}$ has at least $k - 1$ other generalized records that equal to it, $\overline{D}$ is called a k-anonymization of D.*

As a final note, we distinguish between three main models of generalization:

- In *(single-dimensional) global recoding*, e.g. [6, 22, 29, 45], each collection of subsets $\overline{A_j}$ is a clustering of the set A_j (in the sense that $\overline{A_j}$ includes *disjoint* sets whose union equals A_j). In such cases, every entry in the jth column of the database is mapped to the unique subset in $\overline{A_j}$ that contains it. As a consequence, every single value $a \in A_j$ is always generalized in the same manner.

- In *local recoding*, e.g. [18, 19, 33, 38, 42, 45], the collection of subsets $\overline{A_j}$ covers the set A_j but it is not a clustering (namely, the subsets in the collection may intersect). In such cases, each entry in the table's jth column is generalized independently to one of the subsets in $\overline{A_j}$ which includes it. Hence, if the age 34, for example, appears in the table in several records, it may be left unchanged in some, or generalized to 30 - 39, or totally suppressed in other records.

- The third model is an intermediate one and it is called *multi-dimensional global recoding*, e.g. [30]. In that model, like in local recoding, the collection of subsets $\overline{A_j}$ is a cover of the set A_j (namely, each value of A_j may be contained in more than one subset in $\overline{A_j}$). However, it is a global recoding in the sense that there exists a global mapping function $g : A_1 \times \cdots \times A_d \to \overline{A_1} \times \cdots \times \overline{A_d}$ such that $\overline{R_i} = g(R_i)$ for all $1 \leq i \leq n$.

In this paper we concentrate on local recoding which is the model that allows anonymization with the smallest losses of information.

2.2 Measures of information loss and the problem of k-anonymization

Many measures were suggested in the literature for the amount of information that is lost in the process of generalizing table entries towards anonymizing the table. Examples include the tree measure [3], the LM measure [22, 36], the CM and DM measures [22], and entropy-based measures such as the EM (the entropy measure), the non-uniform entropy measure [19] and the PMI (Private Mutual Information) measure [20].

Given a database D over the set of attributes $A_1, \ldots, A_d$, hierarchical generalization trees $\overline{A_1}, \ldots, \overline{A_d}$, and a measure of information loss $F(\cdot)$, the problem of k-anonymization is to find a corresponding generalization $\overline{D}$ of D which is k-anonymous and minimizes $F(\overline{D})$. That problem is NP-hard, e.g. [3], and therefore either approximation algorithms or heuristic algorithms are invoked in order to find k-anonymizations with small information losses.

The algorithms that we present herein are independent of the choice of the underlying measure of information loss. In the experiment section, we tested our algorithms using the LM and EM measures, which we proceed to describe. Both measures associate a penalty $F(\cdot)$ with each of the nodes in each of the d taxonomies; then, the information loss that is associated with the generalized record $\overline{R} = (\overline{R}(1), \ldots, \overline{R}(d))$ is the average penalty of the nodes, i.e., $F(\overline{R}) = \frac{1}{d}\sum_{j=1}^{d} F(\overline{R}(j))$; finally, the information loss in the entire generalized table, $\overline{D} = \{\overline{R}_1, \ldots, \overline{R}_n\}$, is $F(\overline{D}) = \frac{1}{n}\sum_{i=1}^{n} F(\overline{R}_i)$.

The LM measure is defined as follows: Let $B \in \overline{A_j}$ be a subset of A_j from the hierarchical generalization tree. Then

$$F(B) := \frac{|B| - 1}{|A_j| - 1}\,. \qquad (1)$$

Therefore, the overall LM cost associated with the generalization $\overline{D} = \{\overline{R}_1, \ldots, \overline{R}_n\}$ is

$$F_{LM}(\overline{D}) = \frac{1}{nd} \sum_{i=1}^{n} \sum_{j=1}^{d} \frac{|\overline{R}_i(j)| - 1}{|A_j| - 1}\,. \tag{2}$$

The definition of the EM measure is a bit more involved. The public database $D = \{R_1, \ldots, R_n\}$ induces a probability distribution for each of the public attributes. Let X_j, $1 \leq j \leq d$, denote the value of the attribute A_j in a randomly selected record from D. Then

$$\Pr(X_j = a) = \frac{\#\{1 \leq i \leq n : R_i(j) = a\}}{n}\,.$$

Let B be a subset of A_j. Then $F(B)$ is defined as the corresponding conditional entropy,

$$F(B) := H(X_j|B) = -\sum_{b \in B} \Pr(X_j = b|X_j \in B) \log_2 \Pr(X_j = b|X_j \in B)\,, \tag{3}$$

where

$$\Pr(X_j = b|X_j \in B) = \frac{\#\{1 \leq i \leq n : R_i(j) = b\}}{\#\{1 \leq i \leq n : R_i(j) \in B\}}\,, \qquad b \in B. \tag{4}$$

Therefore, the overall EM cost associated with the generalization $\overline{D} = \{\overline{R}_1, \ldots, \overline{R}_n\}$ is

$$F_{EM}(\overline{D}) = \frac{1}{nd} \sum_{i=1}^{n} \sum_{j=1}^{d} H(X_j|\overline{R}_i(j))\,. \tag{5}$$

2.3 Clusterings and anonymizations

Each k-anonymization defines a clustering of the data records into clusters of size at least k. Specifically, every k-anonymization induces an equivalence relation on D where $R_i \sim R_j$ iff $\overline{R}_i = \overline{R}_j$; then, the induced clustering is the corresponding set of equivalence classes.

Similarly, any clustering of the records of D defines a corresponding anonymization. Assume that $\mathcal{C} = \{C_1, \ldots, C_t\}$ is a clustering of the records in D to t disjoint clusters of size at least k. Then each record will be replaced by the *closure* of the cluster to which it belongs:

Definition 2.3. *Given a cluster of records $C = \{R_{i_1}, \ldots, R_{i_c}\}$ over $A_1 \times \cdots \times A_d$, and a collection of hierarchical generalization trees, $\overline{A}_j$, $1 \leq j \leq d$, the closure of C is the minimal record in $\overline{A}_1 \times \cdots \times \overline{A}_d$ that generalizes all records in C.*

The cost of C is defined as the information loss that is associated with its closure, $F(\overline{C})$, as defined in Section 2.2. The information loss of the anonymization $\overline{D}$ that corresponds to the clustering $\mathcal{C} = \{C_1, \ldots, C_t\}$ is then $F(\overline{D}) = \frac{1}{n} \sum_{i=1}^{t} |C_i| \cdot F(\overline{C}_i)$.

2.4 The sequential algorithm

In [24] it is argued that top-down algorithms are more suitable for distributed computation than bottom-up ones since the intermediate views that are revealed during a bottom-up algorithm may disclose sensitive information. For that reason, the agglomerative algorithm [18, 36] and the k-member algorithm [9] are not suitable for the distributed setting. Since in their preliminary stages it is necessary to compute distances between pairs of records held by two players, and some of those distances may be small or even zero, one player may learn information about a record held by another player.

The sequential algorithm [20] works neither top-down, nor bottom-up; it works "sideways". In that algorithm, one starts with an initial random clustering of the records. Such a clustering induces an anonymization of the database, as explained in Section 2.3. The algorithm then attempts to improve the quality of the clustering (namely, the quality of the induced anonymization) by examining the current location of each record and moving it to the cluster in which it fits best. It does so repeatedly until it reaches a stage in which it can find no single-record moves that reduce the information loss in the induced anonymization. At that point, if there exist clusters of size smaller than k, it applies on them the agglomerative algorithm until all clusters are of the required minimum size. As this algorithm performs a local search procedure, it will find a local minimum that depends on the starting point of the search (namely, the initial clustering). Hence, it can be repeated several times and then output the best local minimum that it found.

The sequential algorithm, as we show in Section 9, produces anonymizations with utility that is significantly greater than that of anonymizations produced by the Mondrian [30] algorithm, that was implemented in the distributed setting in [24]. The utility gap with respect to the Mondrian algorithm stems from the fact that the sequential algorithm achieves anonymizations by means of local recoding, an anonymization model which is broader than the global multidimensional recoding that is implemented by the Mondrian algorithm. In addition to that, the sequential algorithm does not need to assume an order on each of its attributes, unlike the Mondrian algorithm.

In fact, as shown in [20], the sequential algorithm is currently one of the leading k-anonymization algorithms in terms of the utility of its output. It was compared in [20] against the Mondrian algorithm, an improved version of the Hilbert-curve algorithm [17], the agglomerative algorithm and the k-member algorithm. The information losses in the outputs of the sequential and the agglomerative algorithms were very close (but the runtime of the sequential algorithm is much better), and they are slightly smaller than the information loss of the output of the k-member algorithm. However, the agglomerative and the k-member algorithms are not suitable for the distributed setting, as discussed above. The next best algorithm which is not bottom-up is the improved Hilbert-curve algorithm. As shown in [20] and as we show again here in Section 9, the sequential algorithm is significantly better, in terms of the information losses that it yields, than the improved Hilbert-curve algorithm.

(We note that there exist a number of other algorithms, e.g. [1, 2, 11], that use techniques other than generalization to achieve k-anonymity. To the best of our knowledge, those algorithms were not implemented in the distributed setting.)

2.5 The distributed setting

In the case of horizontal partitioning, the database $D = \{R_1, \ldots, R_n\}$ is distributed among m sites, or players. Player i holds a table with n_i of the records, say $D^i = \{R^i_1, \ldots, R^i_{n_i}\}$, $1 \le i \le m$, where $n = \sum_{i=1}^m n_i$ and $D = \bigcup_{i=1}^m D^i$. The goal is to design a distributed algorithm so that each player computes a generalization of his table, $\overline{D}^i$, such that the union of all those tables, $\overline{D} = \bigcup_{i=1}^m \overline{D}^i$, satisfies k-anonymity.

In the case of vertical partitioning, the set of attributes $\mathcal{A} = \{A_1, \ldots, A_d\}$ is partitioned into m

disjoint sets, $\mathcal{A}_1, \ldots, \mathcal{A}_m$, and Player i holds the partial database $D^i = \{R^i_1, \ldots, R^i_n\}$, where R^i_j is the projection of R_j onto the subset of quasi-identifiers $\mathcal{A}_i$. The goal is to design a distributed algorithm so that each player computes a generalization of his table, $\overline{D}^i$, such that the join of all those tables, $\overline{D} = \overline{D}^1 || \cdots || \overline{D}^m$, satisfies k-anonymity.

3 SMC protocols

Here we describe secure multi-party protocols that will be used by our distributed anonymization algorithms in the subsequent sections. The sum protocol (Section 3.1) will be used to compute sizes of clusters of records that are distributed among the various players in the horizontal setting, or their generalization costs in the vertical setting. The `And` protocol (Section 3.2) will be used to compute closures of clusters in the horizontal setting. While the sum protocol is part of the folklore for many years now and was used in previous studies (e.g. [25]), the `And` protocol is a novel one.

3.1 Computing the sum of private integers

The first and most basic secure multi-party protocol that we will need is a protocol to compute sums of private integers. Algorithm 1 implements such a computation. N is any sufficiently large integer that is agreed among the players in advance.

Algorithm 1 Secure computation of the sum

```
Input: Player $i$, $1 \le i \le m$, has an input integer $a_i \in \mathbb{N}$.
Output: $a = \sum_{i=1}^m a_i$.
1: Player 1 sets $a = 0$.
2: for $i = 1, \ldots, m$ do
3:     Player $i$ generates a random element $r_i \in \mathbb{Z}_N$ and sends to Player $i+1$ (or Player 1 when
       $i = m$) the value $a = a + a_i + r_i$, where all operations are made in $\mathbb{Z}_N$.
4: end for
5: for $i = 1, \ldots, m$ do
6:     Player $i$ sends to Player $i+1$ (or Player 1 when $i = m$) the value $a = a - r_i$.
7: end for
8: The value of $a$ at this stage is $a = \sum_{i=1}^m a_i$.
```

The above algorithm can be easily extended to compute sums of vectors. So, whenever it is needed to compute several sums which are independent of each other, it is preferable to invoke Algorithm 1 once for adding vectors, than invoking it several times for adding scalars, in order to reduce the communication cost. The communication complexity of Algorithm 1, for adding inputs of any size, is $2m$.

Our protocols are based on two basic protocols — the sum protocol that was described above, and the `AND` protocol, that we present in Section 3.2 below. For both of these protocols we present a version that is secure against collusions.

3.2 Simplified SMC protocols for computing the `AND`

We suggest here a novel SMC protocol for computing the `AND` of the private bits held by the players, $b = \prod_{i=1}^m b_i$. (The case $m = 2$ is usually referred to as the Match-Making problem.) Our protocol is much simpler than the generic solutions suggested in [7, 8, 21]. (We would like to stress that those generic solutions are practical, and especially the one that was recently introduced in [8], called FairplayMP, which is based on the solution in [7] and improves it. However, the solutions that we present herein are much simpler to understand and program, and they employ less cryptographic primitives.)

Our protocol, given in Algorithm 2, is presented for computing `And` of single bits; the extension to binary vectors is straightforward. It is based on the fact that the `And` of all bits equals 1 iff their sum equals m. Hence, the players execute the secure sum protocol up to one step before its completion (Steps 1-8). Then, in Steps 9-10, the last two players check whether the sum equals m or not. To do that, they need to securely compare two values (u and v), without disclosing them. To that end they may invoke a secure oblivious string comparison algorithm (e.g. [16]). We note that if $m > 2$ the two values may be compared more easily as follows: Players $m-1$ and m will send to Player 1 the values $h(r+u)$ and $h(r+v)$, where r is a large random number that the two players choose jointly, and h is a secure hash function, in the sense that $h(r+u)$ and $h(r+v)$ do not reveal information on $u-v$. Then, Player 1 will output $b=1$ if the two hashed values equal, and $b=0$ otherwise.

Algorithm 2 Secure computation of the `AND`

```
Input: Player $i$, $1 \leq i \leq m$, has an input bit $b_i \in \{0,1\}$.
Output: $b = \prod_{i=1}^{m} b_i$.
1: Player 1 sets $a = 0$.
2: for $i = 1, \ldots, m$ do
3:     Player $i$ generates a random element $r_i \in \mathbb{Z}_{m+1}$ and sends to Player $i+1$ (or Player 1 when
       $i = m$) the value $a = a + b_i + r_i$, where all operations are made in $\mathbb{Z}_{m+1}$.
4: end for
5: for $i = 1, \ldots, m-2$ do
6:     Player $i$ sends to Player $i+1$ the value $a = a - r_i$.
7: end for
8: Player $m-1$ computes $u := a - r_{m-1} = \sum_{i=1}^{m} b_i + r_m$.
9: Players $m$ computes $v = m + r_m$.
10: Players $m-1$ and $m$ output $b = 1$ if $u = v$, and $b = 0$ otherwise.
```

Before moving on, we observe that the above described solution may be easily modified in order to compute other Boolean functions. The first function is `OR`: It is easy to see that if we set $v = r_m$ in Step 9, and modify Step 10 to output $b = 0$ if $u = v$ and $b = 1$ if $u \neq v$, the protocol will compute the `OR` of the input bits. The second function is the following threshold function,

$$T_t(b_1, \ldots, b_m) = \begin{cases} 1 & \sum_{i=1}^{m} b_i \geq t \\ 0 & \text{otherwise} \end{cases} ,$$

where $0 < t \leq m$. To that end, Players $m-1$ and m agree on a secure hash function and a large random integer r. Then, in case $m > 2$, Player m sends to Player 1 in Step 10 a random permutation of the set of values $H = \{h(r + i + r_m) : t \leq i \leq m\}$, while Player $m-1$ sends to Player 1 the value $h(u+r)$. Player 1 then checks whether the value that he received from Player $m-1$ appears in the set H received from Player m; if it does, he outputs $b = 1$; otherwise he outputs $b = 0$.

4 A distributed sequential algorithm for horizontally partitioned databases

Here we describe a protocol that implements the sequential algorithm in a distributed manner for horizontally partitioned databases. In the next section we shall describe a protocol for vertically partitioned databases.

Before the protocol starts, the players need to compute the information loss $F(\cdot)$ that is associated with each node in each of the hierarchical generalization trees $\overline{A}_i$, $1 \le i \le d$ (see Section 2.2). In the case of the LM measure, (2), those values are given by (1) and they can be computed publicly. In the case of the EM measure, (5), those values are given by (3)+(4). Their computation requires the players to compute for each attribute, A_j, $1 \le j \le d$, its distribution in the unified database D. To that end, Player i, $1 \le i \le m$, constructs a vector $(f_1, f_2, \ldots, f_{|A_j|})$ where f_s is the number of records in D^i with the sth value in A_j. Then, using Algorithm 1, they add those vectors from all players. Dividing the result by n gives the probability distribution of A_j in D.

The distributed sequential clustering protocol in the horizontal partitioning setting is given in Algorithm 3. Some of the operations in it require an SMC protocol (those operations are denoted by the comment "SMC protocol"). All other operations can be carried out solely by one of the participating players.

The algorithm uses two parameters that control the size of the clusters: k_0, for the size of the initial clusters, and k_1, for the upper limit on cluster size. As the distributed version simulates the non-distributed version, we used in our experiments values that were found in [20] to yield good performance — $k_0 = 0.5k$ and $k_1 = 1.5k$.

The algorithm starts (Step 1) by computing the total number of records in the unified database, using Algorithm 1. (We assume that also n_i need to remain private; if those numbers are not sensitive, then n can be computed in a public manner.) After n was computed, the players create a random clustering of the records of the unified database D into t clusters, where the size of each cluster is roughly k_0 (Step 2). To that end, each player privately generates a random labeling of his own records by labels from $\{1, \ldots, t := \lfloor n/k_0 \rfloor\}$, where the number of records in D^i with any given label is either $\lfloor n_i/t \rfloor$ or $\lceil n_i/t \rceil$. The cluster C_s, $1 \le s \le t$, is the union of all records from all m tables that have the label s.

Then, in Step 3, Player 1 starts SMC protocols to compute the sizes and closures of all clusters. Computing the size of a given cluster may be carried out by invoking Algorithm 1. The protocol for computing the closure of a cluster is described in Section 4.2. Once the closures were computed, the corresponding generalization costs of the clusters may be computed publicly, since the players have computed earlier the generalization cost $F(\cdot)$ of all nodes in $\overline{A}_1, \ldots, \overline{A}_d$.

The core of the algorithm is the main loop (Steps 4-20) on all players, and for Player i on all of the records in his table D^i. The idea is to greedily improve the allocation of each of the records in D to the currently best cluster for it, until we reach an iteration during which all records in D were found to be in the best place for them. To that end, the current player examines each of his records and checks whether a better cluster may be found for that record, in the sense that if we re-allocate that record, the overall generalization cost of the clustering would reduce. At this stage, we get rid of clusters that are singletons (Steps 12-14). After completing the whole loop, we look for clusters that are too large (namely, of size greater than k_1), and randomly split each such cluster to two equally-sized clusters (Steps 21-24). If at least one record moved during the last loop, the main loop is iterated. Otherwise, the protocol starts the final stage, Steps 26-30.

In the final clustering stage we take care of clusters that are smaller than k, by applying on them

Algorithm 3 Sequential clustering for k-anonymization in horizontally-partitioned databases

```
Input: $m$ tables $D^i = \{R^i_1, \dots, R^i_{n_i}\}$, integer $k$.
Output: A $k$-anonymized table, $\overline{D} = \{\overline{R}_1, \dots, \overline{R}_n\}$ of $\bigcup_{i=1}^m D^i$, where $n = \sum_{i=1}^m n_i$.
 1: Compute $n = \sum_{i=1}^m n_i$. {SMC protocol}
 2: Choose a random partition of the data records into $t := \lfloor n/k_0 \rfloor$ clusters, $C_1, \dots, C_t$.
 3: Player 1 computes the size, closure, and generalization cost of all clusters, $|C_s|$, $\overline{C}_s$, and $F(\overline{C}_s)$,
    $1 \le s \le t$. {SMC protocol}
 4: for $i = 1, \dots, m$ do
 5:   for $j = 1, \dots, n_i$ do
 6:     Let $C_s$ be the cluster to which record $R^i_j$ currently belongs. Compute the closure and
        generalization cost of $C'_s := C_s \setminus \{R^i_j\}$. {SMC protocol}
 7:     for $r = 1, \dots, t$, $r \neq s$ do
 8:       Compute the closure and generalization cost of $C'_r := C_r \bigcup \{R^i_j\}$.
 9:       Compute the change in the overall information loss if $R^i_j$ would move from $C_s$ to $C_r$:
          $$\Delta_{(i,j):s\to r} := \Big(|C'_s| \cdot F(\overline{C}'_s) + |C'_r| \cdot F(\overline{C}'_r)\Big) - \big(|C_s| \cdot F(\overline{C}_s) + |C_r| \cdot F(\overline{C}_r)\big) \ .$$
10:     end for
11:     Let $C_{r_0}$ be the cluster for which $\Delta_{(i,j):s\to r}$ is minimal.
12:     if $|C_s| = 1$ then
13:       Move $R^i_j$ from $C_s$ to $C_{r_0}$ and update the size, closure and generalization cost of $C_{r_0}$.
14:       Remove $C_s$ from the list of clusters.
15:     else
16:       If $\Delta_{(i,j):s\to r_0} < 0$, move $R^i_j$ from $C_s$ to $C_{r_0}$ and update the size, closure and general-
          ization costs of both $C_s$ and $C_{r_0}$.
17:     end if
18:   end for
19:   Transfer to the next player the updated sizes and closures of all clusters.
20: end for
21: for each $C_s$ of size $|C_s| > k_1$ do
22:   Player 1 creates a new cluster and sends a message to all players to move a random half of
      the records in $C_s$ to the new cluster.
23:   Player 1 computes the size, closure, and generalization cost of $C_s$ and the new cluster. {SMC
      protocol}
24: end for
25: If at least one record was moved during the last loop (Steps 4-20), go to Step 4.
26: while the number of clusters of size smaller than $k$ is greater than 1 do
27:   Compute the distance between every pair of small clusters,
      $$\mathrm{dist}(C_s, C_r) := \big(|C_s \cup C_r| \cdot F(\overline{C_s \cup C_r})\big) - \big(|C_s| \cdot F(\overline{C}_s) + |C_r| \cdot F(\overline{C}_r)\big) \ . \qquad (6)$$
28:   Unify the two closest small clusters.
29: end while
30: If there exists a single cluster of size less than $k$, unify it with the cluster to which it is closest.
31: Compute the $k$-anonymization that corresponds to the final clustering. {SMC protocol}
```

the agglomerative algorithm. That algorithm greedily unifies each time the two closest clusters, until the size of all clusters is at least k. In this context, the distance between clusters C_s and C_r, given in Eq. (6), is the amount of information that we would lose if we unify them. If clusters C_s and C_r were found to be the closest ones, all that is needed to do is to relabel all records in C_r as records in C_s, remove cluster C_r, and store the closure and the generalization cost of the new C_s.

The last stage is to translate the final clustering to the corresponding k-anonymization (Step 31). This is the first stage where the sensitive values are incorporated. Assume that the clusters in the final clustering are $C_1, \ldots, C_t$. For each such cluster, it is necessary to create $|C_i|$ generalized records that equal the corresponding closure $\overline{C}_i$. In addition, it is necessary to add the corresponding sensitive values of those $|C_i|$ records. The players cannot reveal the sensitive values of the records that they have in that cluster since then other players would learn about sensitive values in records of other players. In order to perform that computation in a secure manner, the players agree upfront on an ordering of all possible values of the sensitive attribute. Assuming that there are q possible values, each player computes a vector of counts, $(f_1^i, \ldots, f_q^i)$, where f_j^i is the number of his records having the jth sensitive value in cluster C_i. By adding up those vectors, using Algorithm 1, the players can find out the number of records in C_i, from all players, that have the jth sensitive value for each $1 \leq j \leq q$. With that information, they can construct the block of identical generalized records that corresponds to cluster C_i.

4.1 Distributed and centralized implementations of sequential clustering

The implementation of the sequential clustering anonymization algorithm in the distributed setting is essentially the same as its implementation in the centralized setting. The sequential clustering anonymization algorithm and its distributed versions that we describe herein are randomized. For a given input (a database D and an anonymity parameter k) there could be several outputs, depending on the random choices made during the execution of the algorithm. Our claim is that the set of possible anonymizations of the unified database that the collaborating players may get at the conclusion of the distributed protocol (in both the horizontal and vertical settings) is independent of the number of players and the manner in which the database is split among the players. Hence, the distributed protocols simulate the centralized one. We proceed to state that claim formally and prove it.

By applying Algorithm 3 on a database D that is partitioned between m players, we obtain a sequence of clusterings of D's records, $\sigma = (\mathcal{C}_1, \ldots, \mathcal{C}_p)$, where $\mathcal{C}_1$ is the initial random clustering and $\mathcal{C}_p$ is the final one. That sequence depends on the random selections made by the players in Steps 2 and 22 (or on random breaking of ties in stages when there are more than one optimal action). Let $\mathcal{P} = \langle D^1, \ldots, D^m \rangle$ be a partitioning of D among m players and let $\Sigma_{\mathcal{P}}(D, k)$ denote the set of all possible sequences σ that may be realized during an implementation of Algorithm 3 on inputs D and k, when D is partitioned between m players according to $\mathcal{P}$. Then:

Theorem 4.1. *The set $\Sigma_{\mathcal{P}}(D, k)$ is independent of $\mathcal{P}$.*

Namely, each sequence of clusterings that can be realized during a $\mathcal{P}$-distributed implementation of Algorithm 3 on given inputs, is a possible sequence also in a centralized implementation ($m = 1$ and $\mathcal{P} = \langle D \rangle$), and vice-versa.

Proof. Let $\sigma = (\mathcal{C}_1, \ldots, \mathcal{C}_p)$ be a sequence of clusterings in a $\mathcal{P}$-distributed implementation of Algorithm 3. We prove by induction that every prefix of length i in σ can be realized in any other distribution setting, $\mathcal{P}'$, with any number of players, m'. When $i = 1$, any initial random clustering $\mathcal{C}_1$ can clearly be realized in any distribution setting. If $\mathcal{C}_{i+1}$ is obtained from $\mathcal{C}_i$ without randomization, then $\mathcal{C}_{i+1}$ will follow $\mathcal{C}_i$ in any distribution setting since it is implied by the basic operation of the sequential algorithm. If $\mathcal{C}_{i+1}$ is obtained from $\mathcal{C}_i$ by randomization (e.g., $\mathcal{C}_{i+1}$ results from applying Step 22 on $\mathcal{C}_i$), then it can be obtained from $\mathcal{C}_i$ in any distribution setting, since the same random choices are possible in any distribution setting. Since the termination condition is independent of $\mathcal{P}$, the sequence σ can be realized in any distribution setting. $\square$

(Theorem 4.1 and its proof hold also in the vertical partitioning setting.)

4.2 Computing the closure of a cluster

The main SMC protocol in Algorithm 3 is the one needed for computing the closure of a distributed cluster. That computation is needed in the beginning of the algorithm (Step 3), each time we examine the potential utility gain by removing a record from a cluster (Step 6), and whenever a large cluster is split (Step 23). (There are also closure computations in Steps 8 and 27, but those do not require an SMC protocol. In Step 8, the current player knows the closure of C_r and, therefore, he may compute on his own what would be the closure of $C_r \cup \{R^i_j\}$. In Step 27, all players know the closures of C_s and C_r and, therefore, the closure of $C_s \cup C_r$ is simply the minimal record that generalizes those two closures.)

Let C_s denote the sth cluster and let $\overline{C}_s = (X(1), \ldots, X(d)) \in \overline{A}_1 \times \cdots \times \overline{A}_d$ be its closure. In order to compute $\overline{C}_s$, each player starts by computing the local closure of the records in his table that belong to the sth cluster; let us denote the ith local closure by $\overline{C}^i_s = (X^i(1), \ldots, X^i(d)) \in \overline{A}_1 \times \cdots \times \overline{A}_d$, $1 \le i \le m$. The goal is then to compute the minimal generalized record that generalizes all of those m generalized records.

The computation can be done in each dimension separately. Given a taxonomy, say $\overline{A}_j$, the ith player has a node in that taxonomy, $X^i(j)$, and the goal is to securely compute the least common ancestor of all the nodes $X^i(j)$, $1 \le i \le m$. (If Player i does not have at the moment s-labeled records, he sets $X^i(j) = \emptyset$ and then every node in $\overline{A}_j$ may be viewed as an ancestor of $X^i(j)$).

Below, we describe a simple SMC protocol that enables such a computation. Given such a basic protocol, the players may run it for each of the d attributes and thus construct the closure $(X(1), \ldots, X(d))$. (The latter computation may be parallelized in order to reduce the communication overhead.) Once the entire closure, $\overline{C}_s$, is computed, it is possible to compute publicly the corresponding cost $F(\overline{C}_s)$.

We now turn to describe the SMC protocol to compute the minimal ancestor in a taxonomy $\overline{A}$ of nodes $X^1, \ldots, X^m \in \overline{A}$ that are held by the m players. One of the players is designated as the pivot of the computation. He then performs a DFS scanning of the taxonomy tree $\overline{A}$ in search of the minimal node that is an ancestor of all m nodes $X^1, \ldots, X^m$. Once the search hits a node that is not a common ancestor of all input nodes, it goes up to the father node and then tries to go down another branch. The output will be the first node which is a common ancestor that has no direct descendant with the same property.

The basic check in the above described DFS is to decide whether a given public node in the taxonomy, say $X \in \overline{A}$, is a common ancestor of X^i for all $1 \le i \le m$, where X^i is known only to the ith player. To that end, the players need to compute $\prod_{i=1}^m \delta(X, X^i)$, where $\delta(X, X^i) = 1$ if X is an ancestor of X^i, and $\delta(X, X^i) = 0$ otherwise. Hence, we are looking at one of the most basic problems of SMC – computing the AND of m private bits. That computation may be carried out using Algorithm 2 that was presented in Section 3.2.

We would like to point out that in Step 6 we may compute the closure more efficiently than in the initial computation in Step 3. Assume that R^i_j belongs to C_s, that the closure of C_s is $\overline{C}_s = (X(1), \ldots, X(d))$, and the local closure of the current Player i is $\overline{C}^i_s = (X^i(1), \ldots, X^i(d))$. Then Player i executes a loop over all attributes a, $1 \le a \le d$, and checks whether the removal of R^i_j from C_s has an effect on $X^i(a)$. If not, then $X(a)$ will not change either; if, however, $X^i(a)$ changes in wake of a possible removal of R^i_j from C_s, Player i has to start an SMC protocol to compute the new $X(a)$. At this stage, though, he may start the DFS search of the ath taxonomy $\overline{A}_a$ in the current node $X(a)$, rather than in the root.

5 A distributed sequential algorithm for vertically-partitioned databases

The vertical setting is somewhat simpler than the horizontal one, since all records are shared by all players. Hence, the allocation of records to clusters is public, and a single player may act as a pivot throughout the entire protocol. Another significant implication of the fact that all records are shared by all players, is that given a clustering of the records, each player may compute the projections of the closures of all clusters onto the subset of attributes that he governs, without interaction with other players. Those local closures need not be shared with the rest of the players. The only interaction between players is needed when they wish to compute the change in the overall information loss in wake of changes in the clustering. In such cases, each player computes separately the change in the generalization cost of each such contemplated action (be it a record transition or a unification of two small clusters) in his attributes, and then the players invoke Algorithm 1 in order to compute the sum of those changes.

Algorithm 4 implements the distributed sequential algorithm for vertically-partitioned databases. The steps in the algorithm that call for an SMC protocol are marked by the comment "SMC protocol". All other operations can be carried out solely by one of the participating players. During the algorithm, one of the players is designated the pivot, who acts as the master that coordinates the operation of the algorithm, whereas all other players act as slaves.

At the beginning, the pivot selects a random partitioning of the database records into t clusters, where the size of each cluster is k_0 or $k_0 + 1$. (As before, k_0 is a free parameter that depends on k; we selected $k_0 = k/2$.) Since the data is vertically-partitioned, the number of records and the cluster sizes are known to all and, hence, there is no need to perform cluster size computation as in the case of horizontal partitioning. After the pivot informs all players of the initial clustering that he selected, all players compute the local closure and generalization cost of their clusters.

In the main loop of the algorithm (Steps 3-17) the players try to greedily improve the allocation of each of the records in D to a cluster, until they reach an iteration during which all records in D were found to be in the best cluster for them. Here, all players need to collaborate in order to compute the change in the overall generalization cost if a records moves from its current cluster to another cluster (Steps 6-7).

After the sequential phase ends, the algorithm executes agglomerative clustering on the remaining small clusters (Steps 23-29). The distance between each pair of those clusters is computed as the overall change in the generalization cost in case those two clusters were to be unified. As before, each player computes the difference in the local generalization cost if such a union would take place, and then the pivot adds up those differences in an SMC manner in order to find out the

Algorithm 4 Sequential clustering for k-anonymization in vertically-partitioned databases

```
Input: A set $\mathcal{A}$ of attributes $\{A_1, \dots, A_d\}$ partitioned into $m$ subsets $\mathcal{A}_1, \dots, \mathcal{A}_m$ of lengths
    $d_1, \dots, d_m$; $m$ tables $D^i = \{R^i_1, \dots, R^i_n\}$, where $R^i_j$ contains attributes from $\mathcal{A}_i$; integer $k$.
Output: A $k$-anonymization of $D = D^1 || \cdots || D^m = \{R_1, \dots, R_n\}$.
 1: Pivot randomly partitions the data records into $t := \lfloor n/k_0 \rfloor$ equal-sized clusters, $C_1, \dots, C_t$.
 2: All players compute the local closure and generalization cost of each of the clusters in their
    local database.
 3: for $i = 1, \dots, n$ do
 4:   Let $C_s$ be the cluster to which record $R_i$ currently belongs.
 5:   for $r = 1, \dots, t, r \neq s$ do
 6:     Pivot sends a message to all players to compute the change in the local generalization cost
        if $R_i$ would move from $C_s$ to $C_r$.
 7:     Pivot computes the total change in the generalization cost if $R_i$ would move from $C_s$ to
        $C_r$ (we denote that value by $\Delta_{(i,j):s \to r}$). {SMC protocol}
 8:   end for
 9:   Let $C_{r_0}$ be the cluster for which $\Delta_{(i,j):s \to r}$ is minimal.
10:   if $|C_s| = 1$ then
11:     Pivot moves $R^i_j$ from $C_s$ to $C_{r_0}$.
12:     Pivot sends a message to all players to update the local closure and generalization cost of
        $C_{r_0}$.
13:   else
14:     If $\Delta_{(i,j):s \to r_0} < 0$, move $R_i$ from $C_s$ to $C_{r_0}$.
15:     Pivot sends a message to all players to update the local closure and generalization costs of
        both $C_s$ and $C_{r_0}$.
16:   end if
17: end for
18: for each $C_s$ of size $|C_s| > k_1$ do
19:   Pivot creates a new cluster and moves a random half of the records in $C_s$ to the new cluster.
20:   All players compute the local closure and generalization cost of $C_s$ and of the new cluster.
21: end for
22: If at least one record was moved during the last loop (Steps 3-17), go to Step 3.
23: while the number of clusters of size smaller than $k$ is greater than 1 do
24:   For every pair of small clusters, $C_s$ and $C_r$, the Pivot instructs all players to compute the
      change in the local generalization cost if those clusters were to be unified.
25:   Pivot computes the total change in the generalization cost if $C_s$ and $C_r$ were to be unified.
      {SMC protocol}
26:   Pivot identifies the two small clusters whose unification results in the smallest change in the
      overall generalization cost.
27:   Pivot unifies those two clusters; namely, if those are $C_s$ and $C_r$, he relabels all records in $C_r$
      as records in $C_s$, removes cluster $C_r$, and then notifies all players of the new clustering and
      instructs them to compute the local closure and generalization cost of the new $C_s$.
28: end while
29: If there exists a single cluster of size less than $k$, unify it with the cluster to which it is closest.
    {SMC protocol}
30: Output the resulting anonymization.
```

overall difference and, consequently, pick the pair whose union results in the smallest addition to the generalization cost. Once the final clustering has been reached, the players compute the corresponding k-anonymization (Step 30). As opposed to the horizontal setting, here there is no need in an SMC protocol, since the player that maintains the sensitive attribute may compute on his own the list of sensitive values in each block of identical records.

We would like to stress the difference of our approach compared to that of [23]. In their approach, each site computes independently a k-anonymization of his part of the database; but then, in order to validate the k-anonymity of the join of those parts, a secure computation of the intersection of clusters is needed. This costly protocol is avoided by us, since the clustering is being orchestrated by the pivot player.

6 Ensuring diversity

The notion of ℓ-diversity was introduced in [32] as an *enhancement* to k-anonymity. In that model, each cluster of size at least k of indistinguishable records must have at least ℓ "well-represented" distinct values in the sensitive attribute. One of the ways in which ℓ-diversity is usually enforced is by demanding that the frequency of each of the private values within each cluster of indistinguishable records does not exceed $1/\ell$ [45, 46].

The related notion of ℓ-site diversity was introduced in [24]. While ℓ-diversity aims at protecting the privacy of the data subjects, ℓ-site diversity aims at protecting the privacy of the data providers, in the case of horizontal partitioning. In order to illustrate that notion, assume that one of the quasi-identifiers may be used in order to identify the site that provided the data record. For example, if the different sites are hospitals and only one of those hospitals is from the northwest, then a record of a patient from Seattle, WA, can be immediately connected to that hospital. The requirement is then that the generalized values that appear in such quasi-identifiers may not be linked to sets of sites of size less than ℓ.

Our algorithms may be modified in order to support both of those security requirements, in addition to k-anonymity, as we proceed to describe.

6.1 Supporting ℓ-diversity

Sequential clustering may be modified so that it issues k-anonymizations that are also ℓ-diverse, as described in [20]. Let ℓ be the input parameter that indicates the required level of diversity of the final anonymization. A necessary condition is that ℓ would be no larger than $\ell_{\max}$ — the diversity of the sensitive value in the entire table. But as explained and illustrated in [20], if one chooses ℓ to be too close to $\ell_{\max}$, it is possible that the only clustering that is ℓ-diverse is the trivial clustering (where all of the table is one cluster). Hence, in order to allow meaningful clusterings, the input target diversity parameter ℓ should not be too close to $\ell_{\max}$.

The first step is choosing an initial clustering of the database records such that the distribution of the sensitive attribute in each cluster is close to the distribution in the entire table. By doing so, the minimal diversity of the initial clusters would be close to $\ell_{\max}$. Let q be the number of possible values of the sensitive attribute. Hereinafter we refer to records for which the sensitive value is the jth possible sensitive value as j-records, $1 \leq j \leq q$. Let f_j be the number of j-records in D. Hence, $\ell_{\max} = \frac{n}{\max_{1 \leq j \leq q} f_j}$. The goal is to create t initial clusters $C_1, \ldots, C_t$ for which the count of j-records in each cluster is either $\lfloor \frac{f_j}{t} \rfloor$ or $\lceil \frac{f_j}{t} \rceil$. Such an initial clustering will have a diversity

ℓ' which is close to $\ell_{\max}$ (the diversity of that clustering might be smaller than $\ell_{\max}$ as a result of rounding the non-integral values $\frac{f_j}{t}$). If the achieved initial diversity ℓ' is greater than or equal to the input target diversity ℓ, then the initial clustering is ℓ-diverse and we may proceed with the sequential clustering. Otherwise, the input parameter ℓ is too high and needs to be reset to ℓ'.

The goal now is to maintain the ℓ-diversity of the initial clustering. Therefore, during its operation, the algorithm examines each contemplated change of clustering and performs it only if it does not lead to a violation of the ℓ-diversity condition. By doing so, we guarantee that all intermediate clusterings are also ℓ-diverse, and, consequently, so is the final output. There are three types of changes of clusterings that the algorithm performs:

1. Transitions of records from one cluster to another.
2. Splitting of large clusters.
3. Unification of small clusters.

As for record transitions, the modified algorithm performs such an action only if it does not violate ℓ-diversity in neither the originating cluster, nor in the destiny cluster. Specifically, we consider the option of moving a record from a cluster C_i only if such a removal would not decrease the diversity of C_i to below ℓ; if it would, we move on to the next record. Otherwise, we look for a better cluster for that record only among those clusters that can receive that record without violating ℓ-diversity.

Splitting a large cluster to two clusters is done in similarity to the initial splitting of the entire database; i.e., instead of splitting all of D to t equal sized clusters, we split a given large cluster to $t = 2$ equal sized smaller clusters.

As for the unification of small clusters, it needs no checking since the union of two ℓ-diverse clusters is also ℓ-diverse.

Next, we discuss the implementation of those modifications in the two distributed settings. The case of vertical partitioning is simple since the site that holds the sensitive attribute may perform all the above described checks without interacting with other players. The case of horizontal partitioning, on the other hand, requires the players to interact in order to check compliance with the diversity requirement. We proceed to describe the necessary modifications to the protocol in order to perform the initial splitting of the entire database to t clusters, the splitting of a large cluster to two smaller clusters, and moving a record from one cluster to another.

6.1.1 Splitting the database in a manner that maximizes the minimal diversity

Recall that f_j denotes the number of j-records in D, and we wish to find an initial clustering to t clusters, $\mathcal{C} = \{C_1, \ldots, C_t\}$, where the value of t is given, so that the number of j-records in each cluster is either $\lfloor \frac{f_j}{t} \rfloor$ or $\lceil \frac{f_j}{t} \rceil$, $1 \leq j \leq q$. To that end, each player splits his records to t clusters so that the sensitive values in those records are spread evenly. Namely, if that player has g_j j-records, he will make sure that the number of j-records that he has in each of the t clusters is either $\lfloor \frac{g_j}{t} \rfloor$ or $\lceil \frac{g_j}{t} \rceil$. However, such a strategy does not guarantee that the global goal of uniform distribution of each of the sensitive values is achieved.

As an example, assume that there are $m = 2$ players, and that they wish to split the records in the unified database to $t = 2$ clusters with an even distribution of the sensitive values. Assume next that the first player has 11 j-records and the other one has 9 j-records, for some j. The first player splits his 11 j-records to 5 in one cluster and 6 in the other, while the second player splits his 9 j-records to 4 and 5. If both players elected to place the smaller number of j-records in the same cluster, say C_1, then C_1 will have 9 j-records while C_2 will have 11 j-records, as opposed to the desired partition of 10 j-records in each of the two clusters.

In order to rectify this, the players proceed as follows. After each player splits his records to clusters, the players invoke Algorithm 1 to compute the counts of each of the q sensitive values in each of the t clusters. At the end of Algorithm 1, it is Player 1 who recovers first those counts. Let f_j^s, $1 \le j \le q$, $1 \le s \le t$, denote those counts. The goal is to have

$$\max_{1\le s\le t} f_j^s - \min_{1\le s\le t} f_j^s \le 1\,, \quad 1 \le j \le q\,; \tag{7}$$

namely, that for each of the q sensitive values, its counts in all t clusters are as even as possible. Player 1 checks condition (7). If he finds out that it is violated for some of the sensitive values, he will try to rectify this, or at least improve the situation towards meeting that condition. Instead of formalizing this simple process, we illustrate it by example: Assume that there are $t = 4$ clusters and that the count vector of the jth sensitive value is

$$(f_j^1 = 5, f_j^2 = 10, f_j^3 = 7, f_j^4 = 9)\,.$$

The target distribution for such a value would be any permutation of the count vector $(7, 8, 8, 8)$. Player 1 will attempt to approach this target distribution as much as possible by moving some of his own j-records between clusters. After doing so with all the sensitive values, he will transfer to the next player those count vectors so that he will continue to "flatten" the sensitive value distribution among clusters by moving his own records from one cluster to another. By doing so, it is guaranteed that the target "flat" distribution will be achieved eventually. Only then, the players may continue to Step 3 in Algorithm 3 in order to compute the closure of the clusters and proceed with the usual operation of the algorithm.

6.1.2 Splitting a large cluster

Splitting a large cluster to two clusters goes along the same lines as described above. Namely, given a large cluster C in which the number of j-records equals g_j, we split it to two equal sized clusters, C_1 and C_2, in which the number of j-records is $\lfloor \frac{g_j}{2} \rfloor$ and $\lceil \frac{g_j}{2} \rceil$.

It is possible that even if we follow that plan, one of the smaller clusters will not be ℓ-diverse. In such cases, cluster C is retained and not split. As an example, assume that a and b are two integers such that $a > b$, and that the diversity parameter is $\ell = \frac{2a+1}{2b+1}$. If $|C| = 2a + 1$ and the most frequent sensitive value in C appears $2b + 1$ times, then the diversity of that cluster is exactly ℓ. But it is impossible to split C non-trivially to C_1 and C_2 such that both will still be ℓ-diverse. The best non-trivial split is to clusters of sizes a and $a + 1$ where the most frequent sensitive value appears b and $b + 1$ times respectively. But then the larger cluster will not be ℓ-diverse. Hence, in such cases we retain the large cluster and do not split it.

6.1.3 Moving a record from one cluster to another

Consider the stage where Player i is the pivot in the main loop. In Step 6 in Algorithm 3 he should examine whether the removal of R_j^i from C_s would decrease the diversity of C_s to below the allowed threshold of ℓ. Player 1 can check that on his own, without interacting with other players, since he has the counts f_j^s, $1 \leq j \leq q$, and he knows the sensitive value of R_j^i. If the diversity of C_s would become less than ℓ, he would not move it. Then, in the loop over all other clusters, Steps 7-10, he would look for an optimal cluster for R_j^i only among those clusters C_r for which $C_r \bigcup \left\{ R_j^i \right\}$ has a diversity of at least ℓ. Also that check may be performed solely by the pivot without interacting with other players. When Player i finishes his loop over all of his records, Steps 5-18, he will transfer to the next player (in Step 19) the new counts f_j^s, $1 \leq j \leq q$, $1 \leq s \leq t$.

6.2 Supporting ℓ-site diversity

We outline here a modification to our solution in order to support ℓ-site diversity. Assume that sites may be identified by b attributes, say $A_1, \ldots, A_b$. First (in a preliminary stage, before the distributed protocol starts), the corresponding hierarchical generalization trees of those attributes are pruned so that they do not include nodes that are supported by less than ℓ sites. Specifically, if X_i is a leaf of $\overline{A}_i$, the hierarchical generalization tree of A_i, that is supported by less than ℓ sites, we remove X_i as well as its siblings from $\overline{A}_i$, so that the father of X_i in $\overline{A}_i$ becomes a leaf. That process is repeated until all leaves in $\overline{A}_i$ are ℓ-site diverse.

In case that $b = 1$ (namely, sites may be identified by only one quasi-identifier, say `location`) that is it; no further precaution steps are needed. If, however, $b > 1$, it is not enough. Assume that $(X_1, \ldots, X_b)$ is a b-tuple of nodes in the Cartesian product of the corresponding pruned taxonomies $(\overline{A}_1, \ldots, \overline{A}_b)$; namely, X_i is a node in $\overline{A}_i$ that supports a set of sites S_i of size $|S_i| \geq \ell$, $1 \leq i \leq b$. Even though each of those b nodes can be linked to no less than ℓ sites, it is possible that $|\bigcap_{1 \leq i \leq b} S_i| < \ell$, whence their combination can be linked to less than ℓ sites. Such combinations can be identified upfront in the preliminary stage. Then, in the course of the distributed protocol, the computation of the closure of a given cluster must not find out that its closure in the first b attributes is $(X_1, \ldots, X_b)$ since then the closure can be linked to less than ℓ sites. Therefore, as opposed to the usual operation of the algorithm in which the computation of the closure in all attributes may be performed in parallel, here it should be done in a sequential manner. First, the players should compute the closure X_1 in $\overline{A}_1$. Then, they will compute the closure in $\overline{A}_2$, but the search should not go below a node in $\overline{A}_2$ that has a son whose conjunction with X_1 is not ℓ-site diverse. Once the closure in $\overline{A}_2$ is found, say X_2, the next taxonomy $\overline{A}_3$ is searched, bearing in mind that only nodes whose conjunction with both X_1 and X_2 is ℓ-site diverse are allowed. Clearly, the order in which the taxonomies are searched may have an effect on the computed closure. Therefore, the players must start with the attribute that is deemed as the most valuable one for the data mining purpose in mind, and then move on to the second most valuable attribute and so forth.

Assume, for example, that sites may be identified by `location` and `income`. Table 1 shows how the location of an individual or his annual income level can be linked to possible hospitals in the Bay Area. All values in those tables are 2-site diverse. However, in order to respect 2-site diversity, it is forbidden to reveal that all records in some cluster are of patients from Palo Alto with an income level of 300K-400K, since such patients are most probably from hospital H7. Hence, in such cases, we either generalize Palo Alto to its ancestor in the `location` taxonomy (say, The Bay Area in this example), or we generalize 300K-400K to 100K-500K. If `location` is the more important attribute, then we first compute the closure in that attribute. If the closure turns out to be San Francisco, then we can compute the closure in the `income` attribute without any restrictions. However, if the closure turns out to be Palo Alto, and when computing the closure in the `income` attribute we found out that 100K-500K is consistent with all records in the cluster, we will not continue the computation of the closure down the `income` taxonomy since a more specified value might lead to a violation of 2-site diversity.

`location`	Hospitals
San Francisco	H1,H2,H3
Oakland	H3,H4,H5
Palo Alto	H6,H7
San Jose	H7,H8

`income`	Hospitals
100K-200K	H1,H2,H3,H4,H5,H6
200K-300K	H2,H3,H6,H7
300K-400K	H7,H8
400K-500K	H7,H8

Table 1. Associating location and income of patients to hospitals

7 Complexity analysis

In this section we analyze the worst-case complexity of both algorithms – the horizontal and the vertical. In both algorithms there are computations that are done locally, and computations that require the players to communicate. As the contribution of this paper is distributed protocols (that are based on a centralized algorithm that was already studied in [20]), we focus on the communication complexity. Communication complexity can be measured by either message complexity (i.e., the number of messages that are transmitted between players, regardless of their actual length) or bit complexity (the volume of the transmitted data). As discussed in Section 4, the algorithm in the horizontal setting, Algorithm 3, uses two types of SMC protocols. One for the computation of sums of private integers and one for the computation of `And` of private bits. On the other hand, in the vertical setting it uses only the sum protocol. The message complexity of both of those protocols is $2m$, due to their ring nature.

In order to reduce the communication overhead, we aggregate several computations that may be executed in parallel. For example, if Player i, $1 \leq i \leq m$, holds a sequence of numbers $x_1^i, \ldots, x_\ell^i$, and we need to compute $x_j = \sum_{i=1}^m x_j^i$ for all $1 \leq j \leq \ell$, then instead of invoking the sum protocol ℓ times for adding integers, we invoke it once for adding vectors of length ℓ, as that reduces the message complexity from $2m\ell$ to $2m$.

We proceed to discuss all invocations of SMC protocols by either of our two algorithms. For each of those invocations we analyze its message complexity and the length of the transmitted message; the corresponding bit complexity is the product of the message complexity and the message length.

7.1 The horizontal setting

When Algorithm 3 invokes the sum protocol (Algorithm 1), it does so in order to compute sums of integers that do not exceed n. As the number of bits to represent integers is logarithmic, the length of the messages in that protocol is $O(\log n)$. Therefore, each invocation of the sum protocol, which we denote hereinafter by $[S]$, has message complexity $2m$ and bit complexity $O(m \log n)$.

When Algorithm 3 invokes the `And` protocol (Algorithm 2), it does so in order to compute the closure of a given cluster simultaneously in all d attributes. That means that in each invocation of that protocol, which we denote hereinafter by $[A]$, we compute the `And` of m vectors, where each vector consists of d bits. To do that, we set the length of the modulus N in Algorithm 2 to $\lceil \log m \rceil$, so that the resulting integer vectors are of length $d\lceil\log m\rceil$ bits. Hence, each invocation of the `And` protocol has message complexity $2m$ and bit complexity $O(md\log m)$.

In what follows, we only count the number of invocations of either $[S]$ or $[A]$. The resulting message and bit complexities can be obtained by multiplying those numbers by the corresponding complexities of those two basic protocols, as specified above.

- Step 1 invokes $[S]$ for computing n.
- Step 3 invokes $[S]$ for computing $|C_s|$, $1 \leq s \leq t$. Since $t = O(n/k)$, the corresponding contribution is $O(n/k) \times [S]$.
- In Step 3 we compute also the closure of all clusters. For any given cluster, the number of invocations of $[A]$ is at most $T = \max_{1\leq j\leq d} |\overline{A}_j|$ — the size of the largest taxonomy. As the number of clusters is $t = O(n/k)$, the overall contribution to the complexity is $O(nT/k) \times [A]$.
- The execution of Step 6 requires no more than T invocations of the `And` protocol (but due to the efficient implementation of that step which is discussed in Section 4.2, it will be usually much less). Hence, as that step is visited nL times, where L is the number of iterations in the sequential algorithm, its contribution to the overall complexity is $O(nLT) \times [A]$.
- Each invocation of Step 23 requires one invocation of the sum protocol and at most T invocations of the `And` protocol, for each of the two clusters. Since there may be no more than $O(n/k)$ large clusters and this step may be executed in each iteration, it incurs a cost of $O(nL/k) \times [S]$ and $O(nLT/k) \times [A]$.

Hence, the communication complexity of Algorithm 3 is bounded by $O(nL/k)\times[S]+O(nLT)\times[A]$. Consequently, the message complexity is $O(nLTm)$ and the bit complexity is $O(nL\log n/k)+O(nLTd\log m)$.

Next, we examine the communication cost that is incurred by modifying Algorithm 3 in order to support ℓ-diversity. The added SMC invocations are in the initial splitting (as described in Section 6.1.1) and whenever a large cluster is split (see Section 6.1.2). Each such split requires one invocation the sum protocol in order to add qt numbers (q being the number of sensitive values and t is the number of clusters). Hence, each time a cluster is split there are $O(m)$ messages ($2m$ messages in the sum protocol and up to m additional messages in the procedure of "flattening" the distribution of each of the sensitive values among all clusters). Since there are $O(nL/k)$ large cluster splits altogether, the added number of messages is $O(nmL/k)$.

7.2 The vertical setting

Algorithm 4 invokes only the sum protocol to compute sums of generalization costs. Generalization costs are typically fractional numbers, but once we decide on the required precision, we can reduce those computations to adding integers of some given length. As the length of messages in this case is independent of m, we follow the usual practice in the analysis of communication complexity and focus on the message complexity.

Step 7 in Algorithm 4 occurs nL times, and in each time it triggers one $[S]$, in order to compute the change in the generalization cost for all clusters at the same time. The agglomerative phase requires to compute distances between the small clusters. In each step we may compute the distances between all existing pairs of small clusters in one invocation of the sum protocol. As the number of small clusters is at most n, and the number of small clusters reduces by at least one in each iteration, the number of iterations in that phase is no more than n. Hence, the overall message complexity of that phase is no more than $n \times [S]$. Therefore, the contribution of Step 7 is the dominating one, whence the overall message complexity is $O(nLm)$.

We note that we do not have a theoretical bound on the number of iterations, L. It depends on n and k, but, as explained in Section 4, it is independent of m. Our experiments indicate that L is a small constant that does not exhibit a monotone dependence on neither n nor k; more details on the experiments are given in Section 9.

8 Privacy analysis

In this section we discuss the privacy provided by our two distributed protocols. Namely, we are interested in information that may be inferred by the interacting players from their view during the execution of the distributed protocol. Like in all previous studies, we too assume that the players are semi-honest, i.e., that they respect the protocol, but try to learn as much as they can from their own view of the protocol about records that are held by other players.[1] The data that needs to be protected is not only the sensitive values but also the quasi-identifiers. For example, if the interacting players are hospitals, then a hospital that treated Bob needs to protect not only Bob's medical condition but also the fact that Bob was hospitalized in it.

It is important to note that we do not discuss here privacy issues of the regular (i.e., non-distributed) k-anonymity and ℓ-diversity models. Namely, we are not interested in possible privacy problems due to the publication of the final output of the distributed protocol, since such problems exist also in the non-distributed setting. (Recall that the final anonymization that is computed in a distributed manner is also a possible output in a centralized execution of the anonymization algorithm, see Section 4.1.) As we describe briefly in Section 1.1, all existing privacy models, like k-anonymity, ℓ-diversity, or differential privacy, have their advantages and disadvantages and the selection of the appropriate model for a given setting depends on the characteristics of that setting. As stated above, it is the secure *transformation* of the anonymization algorithm from the centralized setting to the distributed setting which interests us herein.

We proceed as follows: In Section 8.1 we discuss the security of our basic general purpose protocols. The protocol in the horizontal setting is discussed in Section 8.2 and the vertical one is discussed in Section 8.3.

8.1 The basic protocols

The distributed versions of the sequential clustering algorithms use two basic protocols. One for computing the sum of private integers (Algorithm 1 in Section 3.1), and one for computing the least common ancestor (LCA hereinafter) of nodes in a tree (Section 4.2). The LCA protocol invokes Algorithm 2 for computing the `AND` of private bits.

All those protocols are secure in the sense that they reveal to each of the participating players no more than what is implied by his private input and the final output. We proceed to prove those claims.

[1]See [23] and [50] for a discussion and justification of that assumption.

Proposition 8.1. *Algorithm 1 is a secure protocol for computing the sum.*

Proof. (The proof of this well-known claim, see [25], is included for the sake of completeness.) Let $a_i \in \mathbb{N}$, $1 \le i \le m$, be the inputs of the players and $a = \sum_{i=1}^m a_i$ be the output. Also, let N be an integer larger than a. The view of Player j consists of the following values:

$$\sum_{i=1}^{j-1}(a_i + r_i)\,, \quad a + \sum_{i=j+1}^{m} r_i\,, \quad a\,.$$

For any sequence of inputs $(a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_m)$ whose sum equals $a - a_j$, there exist N^{j-2} selections of $r_1, \ldots, r_{j-1}$ and N^{m-j-1} selections of $r_{j+1}, \ldots, r_m$ that are consistent with it and with the given view. Since the shifts are selected uniformly and independently at random, each such sequence is equally probable. Hence, Player j learns about the inputs of the other players no more than what is implied by his own input and the final output. $\square$

Proposition 8.2. *Algorithm 2 is a secure protocol for computing the and of private bits, provided that the oblivious string comparison algorithm is secure.*

Proof. Let $b_i \in \mathbb{N}$, $1 \le i \le m$, be the input bits, $a = \sum_{i=1}^m b_i$ and $b = \prod_{i=1}^m b_i$. First, we concentrate on the main part of the algorithm, up to Step 10 (excluding). The view of Player j where $j < m$ consists of the following values:

$$\sum_{i=1}^{j-1}(b_i + r_i)\,, \quad a + \sum_{i=j+1}^{m} r_i\,, \quad b\,.$$

The view of Player $j = m$ consists of the values:

$$\sum_{i=1}^{m-1}(b_i + r_i)\,, \quad b\,.$$

Let $(b_1, \ldots, b_{j-1}, b_{j+1}, \ldots, b_m)$ be any sequence of input bits that is consistent with b_j and b. There exist N^{j-2} selections of $r_1, \ldots, r_{j-1}$ and N^{m-j-1} selections of $r_{j+1}, \ldots, r_m$ that are consistent with that sequence and with the given view. Since the shifts are selected uniformly and independently at random, each such sequence is equally probable. Hence, Player j learns about the inputs of the other players no more than what is implied by his own input and the final output.

At the end of the protocol, players $m-1$ and m need to compare u and v (Step 10). We described two options: Either using oblivious string comparison or masking and hashing and sending to Player 1 (in case $m > 2$). That part of the protocol is secure in case the oblivious string comparison protocol is, or the hash function is. $\square$

Proposition 8.3. *The LCA algorithm for computing the least common ancestor of private nodes in a tree is secure.*

Proof. Let T be a public tree and assume that each player has a private node in T. The output of the protocol is the minimal node in T that is an ancestor of all input nodes. The protocol consists of a sequence of invocations of Algorithm 2, where in each invocation, one of the nodes in the tree is tested whether it is a common ancestor of all input nodes or not. In view of Proposition 8.2, we may assume that the view of each player consists only of the final outputs of all of those invocations of Algorithm 2. But such a view immediately follows from the actual output; indeed, given that the output is u, then the result of applying Algorithm 2 for u or any of its ancestors is 1, while for all other nodes it is 0. □

8.2 The horizontal setting

The only interaction between the players in the horizontal setting is for computing the size and closure of clusters (as described in Section 4), and computing the distribution of the sensitive values in each cluster (Section 6.1). During the protocol, the players may learn information on records held by other players, which is not implied by their own input and the final output. Therefore, the protocol is not perfectly secure in the cryptographic sense. Such a compromise is widely acceptable since, as written in [25], "allowing innocuous information leakage allows an algorithm that is sufficiently secure with much lower cost than a fully secure approach". Indeed, many distributed protocols accept innocuous information leakage (e.g. [24, 25, 50]) for gaining efficiency, utility and practicality. We proceed to characterize herein the excessive information that is leaked, compare it to information leakage in other protocols, and argue that such a leakage of information is benign from practical point of view.

We separate our discussion to three types of information that the players may learn on the private data of other players. Assume that the different players are hospitals and the partial database of Hospital i, $1 \leq i \leq m$, holds information on the patients in that hospital. One of the participating hospitals may be interested to know whether a particular individual, Alice, was hospitalized in one of the other hospitals. Using Alice's publicly accessible quasi-identifier values, that hospital may try to examine his view of the protocol in order to deduce the answer. More generally, the hospital may wish to learn how many people from a given age range and location took part in the other databases. In Section 8.2.1 we explain why such inferences are hard and sometimes even impossible to extract from the protocol's views. Alternatively, it is possible that one hospital knows that Alice was hospitalized in another participating hospital, but it wishes to know her sensitive value. In Section 8.2.2 we explain why it is impossible to extract such information beyond what is implied by the final k-anonymized and ℓ-diversified anonymization. Finally, it is possible that hospitals will aim at learning information on the number of patients in the other hospitals. In Section 8.2.3 we explain how to hide also that information. (To the best of our knowledge, no other study dealt with the question of hiding the size of the partial databases.)

8.2.1 Information on the quasi-identifiers of records of other players

In this section we discuss possible inferences that the players may make on the quasi-identifier values of records of other players. In the first part of this section we show that any attempt to infer information about the inclusion of a given quasi-identifier record, $R = (R(1), \ldots, R(d))$, in the unified database D is useless. Then, we proceed to characterize the significantly weaker type of information leakage on the quasi-identifier values of records in D that does occur. We conclude this section by comparing the information leakage on quasi-identifiers in our protocol to other protocols in the horizontal setting.

Let $\sigma := (\mathcal{C}_1, \ldots, \mathcal{C}_p)$ denote the sequence of clusterings of the database records as produced by the distributed sequential clustering algorithm, where $\mathcal{C}_1$ is the initial clustering, $\mathcal{C}_p$ is the final one, and $\mathcal{C}_j$, $2 \leq j \leq p-1$ are the intermediate ones. Then, in view of the security proofs in Section 8.1, the only information that is revealed to the players during the algorithm is the size and the closure of the clusters in each clustering $\mathcal{C}_j$, $1 \leq j \leq p$. Stated differently, the players are exposed to multiple anonymized views of D, since the size and the closure of the clusters in the clustering $\mathcal{C}_j$ is equivalent to an anonymized view, $\overline{D}_j$, of D, $1 \leq j \leq p$. In that sequence of anonymized views, only the final one, $\overline{D}_p$, has the sensitive data; all preceding views include only generalized quasi-identifier records without the sensitive data.

Let us begin by defining the hypergraph corresponding to the multiple anonymized view $\langle \overline{D}_1, \ldots, \overline{D}_p \rangle$.

Definition 8.4. *Let $\mathcal{D} := \langle \overline{D}_1, \ldots, \overline{D}_p \rangle$ be a sequence of anonymized views of some basic table $D = \{R_1, \ldots, R_n\}$. Let $\overline{R}_i^j$ denote the ith generalized record in $\overline{D}_j$, $1 \leq i \leq n$, $1 \leq j \leq p$. A set of p generalized records, one from each view — $\{\overline{R}_{i_1}^1, \ldots, \overline{R}_{i_p}^p\}$, is called a clique, if there exists at least one record $R \in A_1 \times \cdots \times A_d$ which is generalized by each of those records. The hypergraph $G_{\mathcal{D}}$ that corresponds to the sequence of anonymized views $\mathcal{D}$ is the hypergraph with the set of nodes $\{\overline{R}_i^j : 1 \leq i \leq n, 1 \leq j \leq p\}$, and the set of hyperedges is the set of all cliques.*

The hypergraph $G_{\mathcal{D}}$ is p-uniform, namely, all hyperedges in it consist of exactly p nodes, one from each anonymized view. An hyperedge connects the records $\overline{R}_{i_1}^1, \ldots, \overline{R}_{i_p}^p$ if they all generalize some specific record R. Hence, those records are connected by an hyperedge if they could all be the generalized view of the same original record in D.

Example 1. Consider the table D in Table 2 that has $d = 3$ quasi-identifier attributes, $A_1 = \{a, b\}$, $A_2 = \{x, y\}$ and $A_3 = \{1, 2\}$. Assume that during the distributed protocol, the players constructed $p = 3$ anonymized views of D as shown in Table 2. The corresponding hypergraph $G_{\mathcal{D}}$ is shown in Figure 1. It has four hyperedges: The three hyperedges that correspond to the three real records in D, and a fourth artifact hyperedge. The first hyperedge is $\{\overline{R}_1^1, \overline{R}_1^2, \overline{R}_1^3\}$, since all those generalized records generalize the record $\overline{R}_1 \in D$. The sets $\{\overline{R}_2^1, \overline{R}_2^2, \overline{R}_2^3\}$ and $\{\overline{R}_3^1, \overline{R}_3^2, \overline{R}_3^3\}$ are two additional hyperedges, corresponding to $\overline{R}_2, \overline{R}_3 \in D$. The fourth hyperedge is $\{\overline{R}_1^1, \overline{R}_2^2, \overline{R}_3^3\}$. All three records in that hyperedge indeed generalize the same record — $(a, x, 2)$. However, as opposed to the first three hyperedges (which generalize a true record in D), that latter record is an artifact one that does not appear in D. $\square$

D	$\overline{D}_1$	$\overline{D}_2$	$\overline{D}_3$
$R_1 = (a, x, 1)$	$\overline{R}_1^1 = (a, x, *)$	$\overline{R}_1^2 = (*, x, 1)$	$\overline{R}_1^3 = (a, *, 1)$
$R_2 = (b, x, 2)$	$\overline{R}_2^1 = (b, x, *)$	$\overline{R}_2^2 = (*, x, 2)$	$\overline{R}_2^3 = (b, *, 2)$
$R_3 = (a, y, 2)$	$\overline{R}_3^1 = (a, y, *)$	$\overline{R}_3^2 = (*, y, 2)$	$\overline{R}_3^3 = (a, *, 2)$

Table 2. A table D and three anonymized views

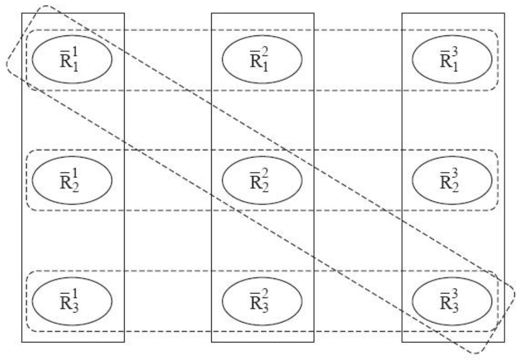


Figure 1. The hypergraph corresponding to the three anonymized views in Table 2

Each of the players may construct the hypergraph $G_{\mathcal{D}}$. Such a graph, as we explain below, induces a collection of “possible worlds” regarding the content of the original table D.

Lemma 8.5. *Let $e = \{\overline{R}^1_{i_1}, \ldots, \overline{R}^p_{i_p}\}$ be a hyperedge in $G_{\mathcal{D}}$. Define, for each quasi-identifier A_h, $1 \leq h \leq d$, the subset $B_h = \cap_{1\leq j\leq p}\overline{R}^j_{i_j}(h)$. Then the Cartesian product $\Omega_e := B_1 \times \cdots \times B_d$ encompases the subset of all records in $A_1 \times \cdots \times A_d$ that are generalized by each of the generalized records in e.*

Proof. It is clear that each record in Ω_e is generalized by all records in e. Conversely, if R is some record in $A_1 \times \cdots \times A_d$ which is not in Ω_e, there exists at least one index $1 \leq h \leq d$ such that $R(h)$ is not included in $\overline{R}^j_{i_j}(h)$ for at least one $1 \leq j \leq p$. Hence, R is not generalized by $\overline{R}^j_{i_j}$. $\square$

Definition 8.6. *A collection of n hyperedges $e_1, \ldots, e_n$ in $G_{\mathcal{D}}$ is called a perfect matching if every two of them are non-adjacent.*

In other words, a perfect matching in $G_{\mathcal{D}}$ is a collection of n hyperedges that covers all nodes in the graph.

Proposition 8.7. *Let $\mathcal{D} := \langle\overline{D}_1, \ldots, \overline{D}_p\rangle$ be a sequence of anonymized views of some basic table $D = \{R_1, \ldots, R_n\}$. For every $R_i \in D$ there exists an hyperedge in $G_{\mathcal{D}}$ that generalizes R_i, and furthermore, it is part of some perfect matching in $G_{\mathcal{D}}$.*

Proof. Assume, without loss of generality, that all records in $\overline{D}_j$, $1 \leq j \leq p$, are ordered like the records in D; i.e., $\overline{R}^j_i$ is the generalized view of R_i in $\overline{D}_j$, $1 \leq i \leq n$, $1 \leq j \leq p$. Then for all $1 \leq i \leq n$, $e_i := \{\overline{R}^1_i, \ldots, \overline{R}^p_i\}$ is a clique, since all those records generalize R_i. That clique is part of the perfect matching $\{e_1, \ldots, e_n\}$. $\square$

We shall refer to the p generalized records in $\mathcal{D} = \langle\overline{D}_1, \ldots, \overline{D}_p\rangle$ that generalize the same original record in D as *a family*. $G_{\mathcal{D}}$ includes exactly n families, $f_1, \ldots, f_n$, where f_i is the family that consists of all generalized records in $\mathcal{D}$ that generalize $R_i \in D$. The collection $\Pi_f := \{f_1, \ldots, f_n\}$ is a perfect matching in $G_{\mathcal{D}}$. The hypergraph $G_{\mathcal{D}}$ may have many other perfect matchings. Any such perfect matching, say $\Pi = \{e_1, \ldots, e_n\}$, could be the perfect matching Π_f that consists of the n families. Since the order of the records in each anonymized view is random, the players have no way to distinguish between the true perfect matching and artifact perfect matchings. Hence, each such perfect matching induces a possible world for D.

Definition 8.8. *Let* $\Pi = \{e_1, \ldots, e_n\}$ *be a perfect matching in* $G_{\mathcal{D}}$*. Then the possible world induced by* Π *is the set* $\{\Omega_{e_1}, \ldots, \Omega_{e_n}\}$*, where* Ω_{e_i} *is the collection of all records that are generalized by* e_i *(as defined in Lemma 8.5).*

Example 2. Consider the table D in Table 3 that has $d = 2$ quasi-identifier attributes — `age` and `gender`. Assume that during the distributed protocol, the players constructed $p = 2$ anonymized views of D as shown in Table 3. It may be easily verified that the corresponding hypergraph in this case is the complete bipartite graph on $\overline{D}_1$ and $\overline{D}_2$, since any selection of two records, one from $\overline{D}_1$ and the other from $\overline{D}_2$, has at least one record that is generalized by both. Hence, there exist in this toy example $4! = 24$ perfect matchings in the graph. Figure 2 shows two of them. The perfect matching that is denoted by solid lines is Π_f, namely, the true perfect matching that consists of all families. (The family f_2 of R_2, for example, is the edge between $\overline{R}^1_2$ and $\overline{R}^2_2$.) The perfect matching that is denote by dashed lines is an artifact perfect matching, denoted Π. The possible worlds that are induced by them are given in Table 4. (For example, the second record in the possible world for Π is $\Omega_{e_2} = (10 - 15, \text{male})$, since the second edge in Π is the edge $e_2 = \{\overline{R}^1_2, \overline{R}^2_1\}$ and the intersection of the two generalized records in that edge is indeed $\Omega_{e_2} = (10 - 15, \text{male})$.) Note that those two possible worlds are written in a concise manner; each of them is in fact a collection of many possible worlds, since each of the records in them is a generalized record. For example, $\{(12, \text{male}), (14, \text{male}), (21, \text{female}), (22, \text{female})\}$ is one of the $324 = 6 \times 6 \times 3 \times 3$ possible worlds induced by Π. $\Box$

D	$\overline{D}_1$	$\overline{D}_2$
$R_1 = (10, \text{male})$	$\overline{R}^1_1 = (10 - 15, *)$	$\overline{R}^2_1 = (10 - 20, \text{male})$
$R_2 = (15, \text{female})$	$\overline{R}^1_2 = (10 - 15, *)$	$\overline{R}^2_2 = (15 - 22, \text{female})$
$R_3 = (20, \text{male})$	$\overline{R}^1_3 = (20 - 22, *)$	$\overline{R}^2_3 = (10 - 20, \text{male})$
$R_4 = (22, \text{female})$	$\overline{R}^1_4 = (20 - 22, *)$	$\overline{R}^2_4 = (15 - 22, \text{female})$

Table 3. A table D and two anonymized views

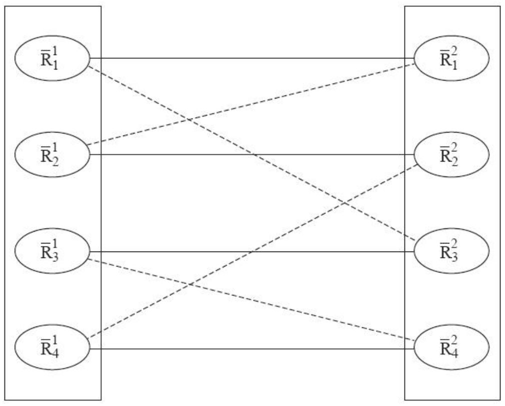


Figure 2. The (hyper)graph corresponding to the two anonymized views in Table 3

$\Omega_{e_1} = (10 - 15, \text{male})$
$\Omega_{e_2} = (15, \text{female})$
$\Omega_{e_3} = (20, \text{male})$
$\Omega_{e_4} = (20 - 22, \text{female})$

$\Omega_{e_1} = (10 - 15, \text{male})$
$\Omega_{e_2} = (10 - 15, \text{male})$
$\Omega_{e_3} = (20 - 22, \text{female})$
$\Omega_{e_4} = (20 - 22, \text{female})$

Table 4. The possible worlds corresponding to Π_f (left) and Π (right)

Assume that one of the players wishes to tell whether a given quasi-identifier record, $R = (R(1), \ldots, R(d))$ appears in D. He will be able to infer with certainty that $R \in D$ if and only if all possible worlds include the record R. He will be able to deduce a probability for R to be in D by computing the percentage of possible worlds that include R. However, the computation of a single possible world (let alone all of them) requires finding a perfect matching in the p-uniform hypergraph $G_{\mathcal{D}}$. Such a computation is known to be NP-hard (see e.g. [33]). Hence, any attempt to infer information about the inclusion of a given quasi-identifier record, $R = (R(1), \ldots, R(d))$, in D would be hard in practice for large datasets. Similarly, any attempt to make similar inferences, such as how many records in D belong to some sub-domain of $A_1 \times \cdots \times A_d$ (representing, say, some location and age range), would be infeasible.

Having said that, a significantly weaker type of information leakage on the quasi-identifier values of records in D does occur. Let C be a cluster in one of the clusterings in $\sigma := (\mathcal{C}_1, \ldots, \mathcal{C}_p)$ and let $C^i = C \cap D^i$ be the set of records in D^i that belong to that cluster. Denote by $c_i = |C^i|$ and $\overline{C^i}$ the size and closure of C^i, and by $c = |C|$ and $\overline{C}$ the global size and closure of C.

Proposition 8.9. *The only information that the ith player learns in wake of computing the size and closure of the cluster C is as follows:*

1. *The union $\bigcup_{i' \neq i} D^{i'}$ includes at least $c - c_i$ records that are consistent with the closure $\overline{C}$.*
2. *For each $1 \leq j \leq d$, $\overline{C}(j)$ is the LCA of $\overline{C^i}(j)$ and the values in the jth attribute of the said $c - c_i$ records.*

Consider for example the case of $m = 4$ players and $d = 2$ attributes – `age` and `location`. Assume that C^1 includes $c_1 = 30$ records, the closure of which equals $\overline{C^1} = ([20-30], LA)$, while the global size of C is $c = 100$ and the global closure is $\overline{C} = ([20-30], CA)$. Then Player 1 may deduce that the union of all other databases, $D^2 \cup D^3 \cup D^4$, includes at least 70 records that are consistent with $\overline{C} = ([20-30], CA)$. Player 1 may deduce no further information about the `age` in those 70 records; as for their `location`, since the local closure (LA) is a proper subset of the global closure (CA), Player 1 may deduce that those 70 records are spread in CA locations that enforce the global closure to be CA. (For example, if Southern CA is a possible generalized value, he may rule out the possibility that all of those 70 records are in Southern CA since then $\overline{C}(2)$ would be Southern CA and not CA.) Finally, the view of Player 1 does not enable him to distinguish between D^2, D^3, and D^4.

Proof. From c_i and c, Player i deduces that $\bigcup_{i' \neq i} D^{i'}$ has $c - c_i$ records in the cluster C. He cannot deduce any further information regarding the number of records in C in any of the other sites since that would contradict Proposition 8.1. From the correctness of the LCA algorithm, Player i deduces that all those $c - c_i$ records are consistent with the computed closure $\overline{C}$.

If j is an attribute index in which $\overline{C}(j) = \overline{C^i}(j)$, Player i may infer no further information about the value of the said $c - c_i$ records in that attribute, as implied by Proposition 8.3. Indeed, if we replace the value of each of those records in the jth attribute with any of the values in the tree underneath the node $\overline{C}(j)$, the output of the LCA algorithm in that attribute would remain the same.

If, on the other hand, $\overline{C}(j) \supsetneq \overline{C^i}(j)$, then by the correctness of the LCA protocol, the jth components in the said $c - c_i$ records must be such that the LCA of them together with $\overline{C^i}(j)$ equals $\overline{C}(j)$. Proposition 8.3 implies that Player i cannot deduce any further information about those records. $\square$

Proposition 8.9 characterizes the information that each player may learn about the quasi-identifier part of the records in the other databases, in wake of computing the size and closure of a given cluster C; we shall denote that piece of information by $I(C)$. Then, if $\mathcal{C} = \{C_1, \ldots, C_t\}$ is a clustering of all records, we let $I(\mathcal{C}) := \{I(C_1), \ldots, I(C_t)\}$ denote the information that is revealed to a given player by the sizes and closures of all clusters in $\mathcal{C}$. We shall refer to it hereinafter as the *information leakage* of $\mathcal{C}$.

Let us assume that the main loop in Algorithm 3 (Steps 4-20) was repeated L times. In the ℓth repetition, at the beginning of the ith cycle of the main loop, Player i obtains from Player $i-1$ the current clustering $\mathcal{C}$ and the corresponding cluster sizes and closures. Hence, he may compute the corresponding information leakage $I(\mathcal{C})$. Therefore, while an ideal solution would expose each player only to the information leakage of the final clustering, Algorithm 3 exposes each player to the information leakage of L clusterings. On one hand, this is more than what an ideal protocol would leak. On the other hand, the excess information that is leaked is benign and it is exactly of the same type as that which is leaked by the final output. Three things worth noting in that regard:

- As mentioned in Section 7, L is a small constant.
- The intermediate clusterings in the L repetitions of the main loop typically reveal less information than the final clustering does, since their utility (which is inversely related to the information that cluster closures leak on the original records in those clusters) is smaller than that of the final clustering.

- The pieces of information that the protocol leaks are determined by the protocol and cannot be selected by the players.

We conclude this section by comparing the above described information leakage to that in other distributed protocols for the horizontal setting. The protocol in [24] discloses a different type of information about the distribution of the quasi-identifiers in the non-final clusters, and, in addition, it leaks information on the distribution of records among the various data holders, whereas our protocol does not. In particular, it discloses for each of the non-final clusters of records the following information: The range and median of values in each of the quasi-identifiers (it is assumed there that each quasi-identifier is fully ordered); the entropy of the distribution of records among the different data holders; and the number of data holders that own records in each such cluster.

As for the protocol in [50], it discloses the distance between specific pairs of records, a disclosure that allows one site to learn information about records held by another site. That is a more severe information leakage than the one described in Proposition 8.9: Even in the worst case in Proposition 8.9, when $c = 2$ and $c_i = 1$, the ith player can only learn about the distance between the record under his control in C and the other record in C (in similarity to the information leakage in [50]), but he will not be able to infer who is the owner of that other record. In addition, such information leakage on the distance between two records will occur only in events when $c = 2$ and $c_i = 1$, in contrast to the protocol in [50] that reveals the distances between all pairs of records.

8.2.2 Information on the sensitive attributes of records of other players

Next, we consider the information that is leaked by the enhancement of the algorithm to support ℓ-diversity, as described in Section 6.1. In view of Proposition 8.1, the only information that is disclosed in wake of each SMC protocol for computing the distribution of the sensitive attribute in each cluster, is precisely that distribution and nothing further. As before, this is the same type of information that the final k-anonymization reveals too, regarding the clusters in the final clustering $\mathcal{C}_p$. Therefore, in similarity to the above discussion, even though the protocol exposes the players to the distribution of the sensitive attribute also in intermediate clusters, no new type of information can be extracted from the protocol views.

It is possible to prevent any leakage of information regarding the distribution of the sensitive values in intermediate clusters, at the cost of executing additional SMC protocols. Instead of computing the actual distribution of the sensitive attribute in each cluster, the players may just verify whether the maximal frequency in the cluster does not exceed the allowed upper limit of $1/\ell$. To do that, the players still use Algorithm 1 in order to add up the frequency vectors from each player. But instead of running Algorithm 1 until the end in order to recover the frequency vector $(f_1, \ldots, f_q)$, they run it until one step before the end, when Player $m-1$ gets $(f_1 + r_1^m, \ldots, f_q + r_q^m)$, where r_j^m is the random noise added by Player m in the jth component. At this stage there are two vectors that need to be compared: The vector $(f_1 + r_1^m, \ldots, f_q + r_q^m)$, which is known only to Player $m-1$, and the vector $(\frac{1}{\ell} + r_1^m, \ldots, \frac{1}{\ell} + r_q^m)$, which only Player m knows. The two players need to verify that the second vector dominates the first one, componentwise. This problem is called the vector dominance problem, and it is closely related to the well-studied problem for securely comparing two numbers (Yao's millionaires' problem [48]). Several SMC protocols for solving this problem were proposed recently in [49].

As discussed in Section 8.2.1, each player is exposed to a sequence of anonymized views of D, which we denoted there by $\overline{D}_1, \ldots, \overline{D}_p$. But since $\overline{D}_i$, for all $1 \leq i \leq p-1$, does not include the sensitive data, that sequence of anonymized views does not allow, say, Player 1, to learn sensitive information of an individual that appears in, say, the partial database D^2 of Player 2. Indeed, the combination of all anonymized views $\overline{D}_1, \ldots, \overline{D}_{p-1}$ does not reveal anything beyond the projection of D onto the quasi-identifiers. (In fact, as explained in Section 8.2.1, it does not reveal even that information.) Hence, for the sake of the argument, we may replace all of those anonymized views with the projection of D onto $A_1 \times \cdots \times A_d$. As implied by the definition of k-anonymity and ℓ-diversity, even if Player 1 has that projection together with the final k-anonymized and ℓ-diversified view, $\overline{D}_p$, he would not be able to locate his target record within less than k generalized records in $\overline{D}_p$, and, in addition, those generalized records will be ℓ-diverse. Hence, the sequence of intermediate anonymized views does not contribute anything towards inferring sensitive data of target records beyond what the final anonymization reveals.

We note that the protocols in [23, 24, 50] do not have a similar information leakage since they do not examine ℓ-diversity.

8.2.3 Information on the number of records in databases of other players

Another place where information may be leaked is in Step 6 of Algorithm 3. Whenever Player i triggers an SMC protocol to examine the potential gain from removing a record R^i_j from its current host cluster, C_s, all other players can know that Player i has (or at least had until that stage) a record in cluster C_s. Hence, at the end of the loop in Steps 5-18, they can deduce his overall number of records, n_i. (Note that they cannot know how those records are split between the clusters since they do not know what were the transitions that the site decided to perform.) However, Player i does not trigger such an SMC protocol for all of his n_i records. If the removal R^i_j from C_s does not change the local closure of C_s, he would not start such an SMC since it is not necessary (as we explain in Section 4.2); also, if the removal of R^i_j from C_s would decrease the diversity of C_s to below the allowed threshold, he would skip to the next record. Therefore, the other sites may deduce only a lower bound on n_i.

If the leakage of such partial information (namely, a lower bound on n_i) is deemed as damaging to the data providers' privacy, we propose a simple randomization of Algorithm 3 in which that information is blurred. In each iteration, Player i will choose to examine $\alpha_i n_i$ of its records, where α_i is a random real number withdrawn from a private distribution which Player i selects. If $\alpha_i < 1$, Player i will examine in that iteration only a fraction of its records. He will start his current series of examinations from the record that comes right after the last record which he examined in the previous iteration, and will proceed in a cyclic manner. If $\alpha_i > 1$, Player i will examine his records in a cyclic manner where some records will be visited more than once. Note that after the completion of a whole cycle, it is relevant to check again the current location of a given record, since the reallocation of the other records might change the previous decision which was made about the location of that record. By setting all selections of α_i to equal 1, we recover the basic non-randomized Algorithm 3.

In the original Algorithm 3, if no record was moved during a complete iteration over all sites and all records, there is no point in performing another iteration, since all n records in D were visited during the last iteration. However, in the above described randomized version of Algorithm 3, it is possible that during one iteration no record was moved, but then in the next iteration other records will be examined and moved. Hence, the stopping criterion should be modified. A natural modification would be as follows. Let h_r be the overall number of records in D that were tested during iteration r; that number is publicly known. The algorithm will stop after iteration r if there exists an integer p such that no record was moved during the last p iterations, and $h_{r-p+1}+\cdots+h_r \geq n$.

8.3 The vertical setting

In Algorithm 4, each site keeps private the local closure and generalization cost. Hence, as implied by Proposition 8.1, the only information that is leaked during the execution of the protocol is the overall change in the generalization cost if a record moves from one cluster to another (Step 7), or two clusters are unified (Step 25 and Step 29). Since the secure sum protocol ends with one of the players learning the sum and all other players learning nothing, our protocol reveals the overall change in the generalization cost only to the pivot, while all other players learn nothing. We proceed to examine what conclusions the pivot may derive from those values.

Assume that in Step 7, the pivot learned that if record R_i would move from cluster C_s to cluster C_r, the generalization cost would change by Δ. Assume, further, that he knows all taxonomies of all attributes (also those that are governed by other sites), and the amount of information loss that is associated with each node in those taxonomies. Then he may perform a search over all quadruples of generalized records, $X, X', Y, Y' \in \overline{A}_1 \times \cdots \times \overline{A}_d$, such that X generalizes X', Y' generalizes Y, and

$$\left(|C'_s| \cdot F(X') + |C'_r| \cdot F(Y')\right) - \left(|C_s| \cdot F(X) + |C_r| \cdot F(Y)\right) = \Delta\,.$$

If there exists a single quadruple that satisfies those conditions, then X and Y are the closures of C_s and C_r prior to the move, while X' and Y' are the closures after the move.

Having said that, such an attack is far from being a real breach of security for the following reasons:

1. Usually, the value of Δ will be consistent with many possible quadruples. The most common information loss measure is the LM [22, 36]; it sometimes appears under different names or variants, e.g., the normalized certainty penalty (NCP) [47], or the global certainty penalty (GCP) [17]. Those measures associate an information loss penalty to a generalized value according to the *number* of specific values that it generalizes, regardless of the actual values. Thus, the above described attack is completely useless when using such measures, since the value of Δ cannot be used to distinguish between an age range of [10-15] and an age range of [80-85].

2. Even information loss measures that might distinguish between generalized values of equal sizes, always associate zero information loss to exact values. Hence, when the closure has entries that are non-generalized values, such an attack might reveal that fact, but it cannot tell what are the actual non-generalized values in those entries.

3. Even in cases where the pivot player is capable of recovering the closures X, X', Y, Y', he may reveal very limited information on the quasi-identifiers which are controlled by other players. At the very worst scenario, he may deduce a generalization of the quasi-identifiers of some record. To illustrate that, assume that `location` is a quasi-identifier that is not controlled by the pivot, and its generalization hierarchy has the exact zipcode as the specific location, cities in the first generalization level, and states in the second generalization level. If by moving a record R to a cluster C the closure of that cluster was changed from LA (city) to

CA (state), the adversary may deduce that R is located in CA but not in LA. If in further steps the adversary was so lucky to be able to deduce that R's location is also not in other cities in CA, he may be able eventually to infer the correct city of that record. However, as explained in point 2 above, the pivot will never be able to infer the exact zipcode within that city.

4. The information which is disclosed relates only to the quasi-identifiers and not to the sensitive values which are revealed only at the end.

5. Even when using information loss measures that do not depend just on the size of the generalized value, and even in cases where the value of Δ identifies a unique quadruple of closures, the attack is not cost effective, as the search space may be too large (even for modest values of d and small taxonomies) for such a negligible and non-guaranteed gain of information.

We would like to add that it is possible to diminish the already negligible leakage of information due to the above described attack by having each player add a small random noise to his input to the sum protocol, in order to prevent the pivot from identifying the a-priori and a-posteriori pairs of cluster closures. Such random noise might imply non-optimal decisions in the sequential clustering, but by properly calibrating the added noise, the effects on the final output will be small and the attack will become even less realistic.[2]

In comparison, the protocol of [23] offers perfect privacy, but it is a very restricted protocol (vertical setting only, and two players only), which is highly inefficient (see Section 9.2) and was not properly tested with respect to utility (see Sections 9.1). The protocol of [34] discloses no information because they utilize a top-down anonymization algorithm in which the intermediate views reveal less information than the final view. While a top-down approach offers privacy advantages, it results in very poor utility results, as we discuss in the experimental section.

9 Experiments

We conducted two sets of experiments. In the first set of experiments we tested the performance of the centralized version of sequential clustering [20] in order to illustrate its main features. In the second set of experiments we simulated the distributed versions of that algorithm which were proposed and analyzed in this study.

Our experiments were conducted on two datasets: The first one is ADULT from the UCI Machine Learning Repository[3]. That dataset was extracted from the US Census Bureau Data Extraction System. It holds demographic information of a small sample of US population with 14 quasi-identifiers such as `age`, `education level`, `marital status`, `occupation`, and `native country` and contains 45,222 records. The second dataset is CENSUS[4]. It has 100,000 records that include 7 quasi-identifiers (`age`, `gender`, `education level`, `marital status`, `race`, `work class`, `country`). We used generalizations taxonomies for the attributes in those datasets of heights ranging from 2 to 4 (where a taxonomy of height 2 means generalization by suppression only). All experiments were implemented in C, and ran on a 1.86 GHz Intel Core Duo processor T2350 personal computer with 2 GB of RAM.

[2]We note that also the closely related simulated annealing clustering algorithm [28] makes on purpose random non-optimal decisions in order to avoid the attraction of local minima.

[3]http://mlearn.ics.uci.edu/MLSummary.html

[4]http://www.ipums.org

9.1 Information loss

In our first experiment, we tested the non-distributed version of the sequential algorithm in order to demonstrate its superiority, in terms of information loss, against other algorithms, and especially those that were implemented also in the distributed setting.

Figures 3 and 4 give the information losses of the sequential algorithm and two other algorithms in terms of the LM and EM measures, as measured on the ADULT and CENSUS datasets, respectively. The first algorithm is the Mondrian (on which the protocols of [24] and [34] are based); the other one is the Hilbert-curve based anonymization algorithm of [17][5]. As mentioned in Section 1.2, also the approximation algorithm of [33] was implemented in the distributed setting [50]; however, we do not include explicit comparison to that algorithm herein since the superiority of the sequential algorithm over that algorithm follows from the set of experiments that were presented in [18] and [20]. (Another reason why we exclude the algorithm of [33] from our experimental evaluation is that it can handle only generalization by suppression.) Note that since our distributed algorithms, as well as those of [24, 50], simulate the operation of the corresponding centralized algorithm on which they are based (Section 4.1), the information losses shown in Figures 3 and 4 reflect the situation also in the distributed setting.

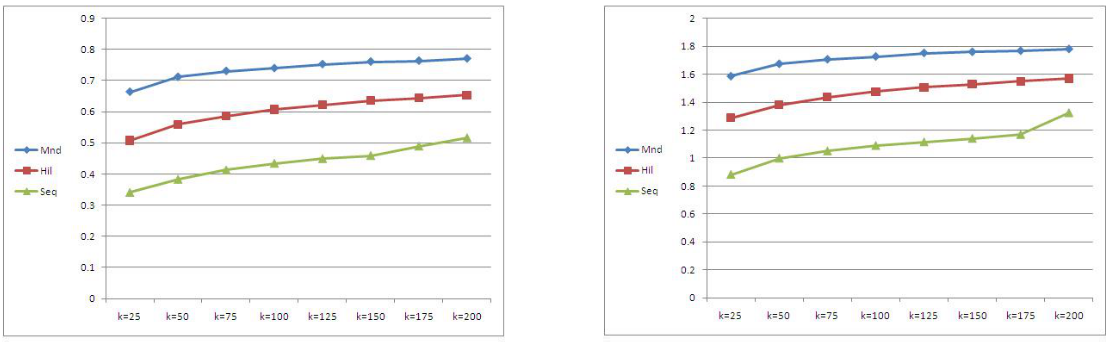
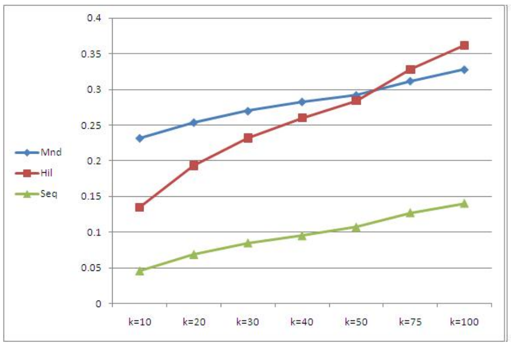


Figure 3. Information Loss (Adult): LM (left) and EM

As for the impact of adding a restriction on the diversity of the final output, such experiments were already conducted in [20]; see Section 8.3 there. (Again, since the distributed version of the sequential algorithm simulates the non-distributed sequential algorithm, Section 4.1, adding a diversity restriction in the distributed version has the same effect as doing so in the non-distributed version.)

Hence, our distributed protocols are based on an anonymization algorithm that is significantly superior, in terms of the utility of the output, to those on which the the distributed protocols of [24, 34, 50] are based. The protocol of [23], unlike the other solutions (including ours), is not a specific protocol but a template of protocols. Each site may run on his database any k-anonymization algorithm; the protocol in [23] only explains how to convert such two independent anonymizations into a k-anonymization of the join of the two databases. Jiang and Clifton implemented

[5]The version that we implement here is an improvement of the version in [17], where we rescale all attributes to fit the same range. That was needed since the original algorithm does not perform well where the various attributes have significantly different range sizes. More on that issue in [20].



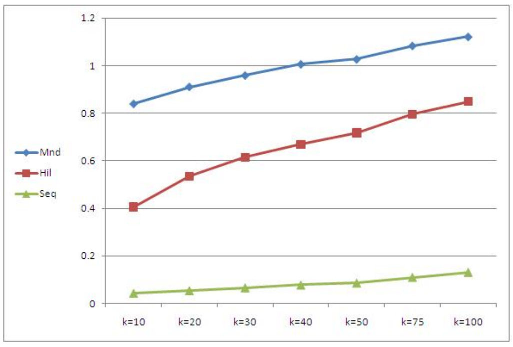


Figure 4. Information Loss (Census): LM (left) and EM

their protocol when both parties run the Datafly anonymization algorithm. The latter is a dated algorithm by Sweeney from 1997 that has very poor utility performance as it is a global recoding algorithm. (Its inventor wrote about it few years later [41, Section 4] that it "can over distort the data.") Moreover, their distributed protocol yields anonymizations with even smaller utility than the centralized algorithm, see [23, Section 6.3]. (This is in contrast to our distributed protocols that offer anonymizations with the same utility as the centralized algorithm.) In order to properly assess the utility offered by the protocol template of [23], more advanced anonymization algorithms, like the sequential clustering algorithm, need to serve as the basic algorithm that each of the parties uses.

9.2 Time performance

Here, we examine the time performance of our algorithms. The time performance depends on the computational cost of the centralized algorithm (since the distributed versions simulate the centralized algorithm) and the added communication cost.

9.2.1 Computational cost

Figure 5 illustrates the high efficiency of the sequential algorithm; it shows the runtime of the centralized algorithm ($m = 1$), in seconds, as a function of n and k. (The different values of n indicate the sizes of the random partial databases of ADULT and CENSUS on which the algorithm was applied.) Those runtimes are of the non-repetitive sequential algorithm. (The repetitive version, that tries several random initial clusterings in order to find the best output, shows negligible variance in the results in terms of information loss.) As for the dependence of the runtime on the dimension d, it is clearly linear, since the cost of computing cluster closures or generalization costs depends linearly on the number of attributes; this linear dependence on the dimension was already validated experimentally in [20].

Next, we turn to measurements of the number of iterations, L. As said in Section 7, it depends on n and k, but not on m. Our experiments indicate that L is a small constant that does not exhibit a monotone dependence on neither n nor k. As can be seen in Figure 6, L always varied between 6 and 10 on various sizes of datasets and for various values of k.

We now turn to examine the computational cost of our algorithms as compared to that of the other

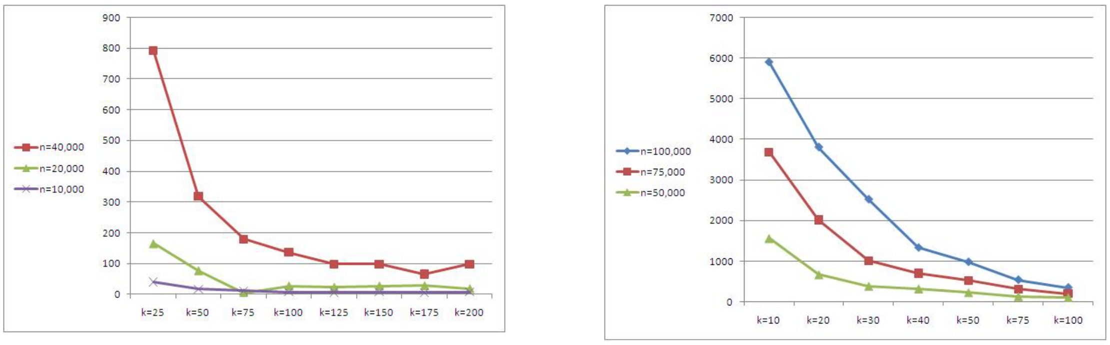


Figure 5. Runtime (seconds): Adult (left) and Census datasets

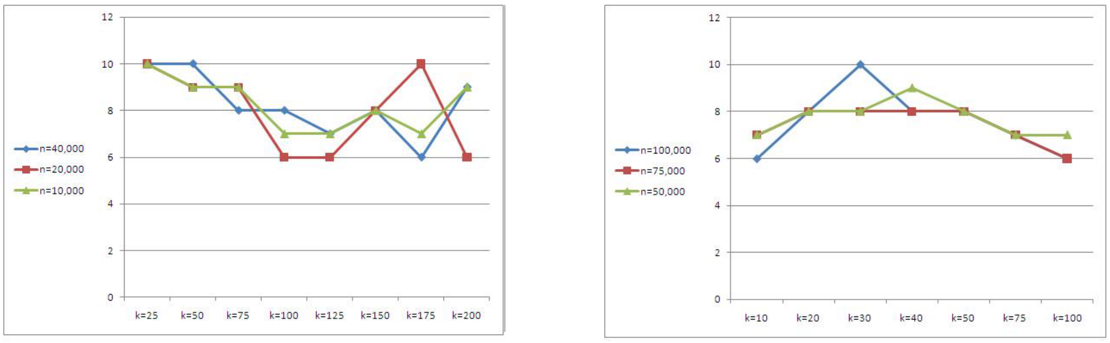
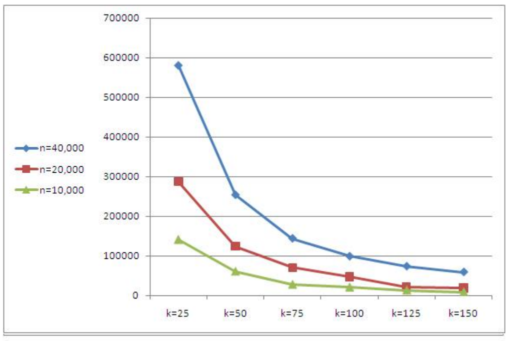


Figure 6. Number of iterations: Adult (left) and Census

distributed k-anonymization protocols. The studies that dealt with horizontal partitionings [24, 50] do not provide any experimental results regarding the time performance of their suggested protocols. The experiments in [34] show that their protocol is very efficient, in terms of runtime, but the value of such a fast running solution is questionable given the poor utility of its output anonymization. A protocol that runs in few seconds or minutes offers no real advantage against a protocol that runs for hours if the latter yields an output with much better utility.

Jiang and Clifton [23] only estimate the runtime of their suggested protocol based on the runtime of its most costly operations, which are homomorphic encryptions (an operation that is based on modular exponentiation with a 1024-bit modulus). The projected number of encryptions needed by their protocol is bounded by $O(n \log q)$ where n is the database size and q is the size of the public key domain. Their resulting runtime estimates, for a subset of the ADULT dataset (30,162 records with 8 quasi-identifiers) are significantly larger than ours, ranging from approximately 10 days, for $k = 20$, to 16 days, for $k = 100$. (Recall that the runtimes that we report in Figure 5 are for the full ADULT dataset — 45,222 records with 14 quasi-identifiers.)

As noted in Section 1.3, our solution is significantly simpler than the previous solutions. While those were based on homomorphic encryptions [23, 50] and oblivious transfers [24, 34] — cryptographic primitives that depend on costly group exponentiations, our solution depends only on



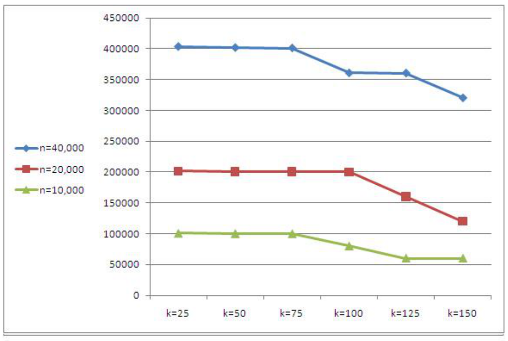


Figure 7. SMC calls on Adult: Horizontal (left) and Vertical (right)

modular addition and secure hash functions. To illustrate the difference between group exponentiations and the latter operations, here are the average runtimes of those operations, as measured on an Intel(R) Core(TM)2 Quad CPU 2.67 GHz personal computer with 8 GB of RAM:

- Modular addition — 0.762 μs (microseconds).
- SHA-1 on an input of 512 bits — 7.8 μs.
- Modular exponentiation with modulus of 1024 bits — 12251 μs.

The staggering differences in the runtimes of those basic operations illustrate the advantage of our protocol as compared to the existing distributed k-anonymization protocols also in terms of computational costs.

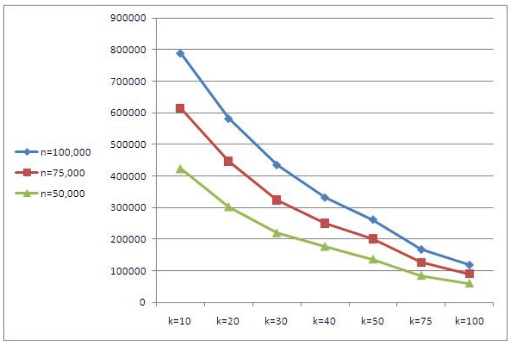
9.2.2 Communication cost

Apart from the computational cost of the centralized algorithm, the runtime of the corresponding distributed protocols is also affected by the cost of communication. The message and bit complexities of the algorithms were analyzed in Section 7. We simulated our distributed protocols and counted the number of SMC calls, namely, invocations of the sum and AND protocols. Figures 7 and 8 show the counts of SMC calls in each of the two settings, for different values of n and k. As can be seen, the number of SMC calls depends linearly on n and decreases with respect to k.

Few comments are in order regarding our evaluation methodology:

1. As the focus of our paper was on algorithmic and privacy aspects, we were not interested in testing the performance of an actual implementation on a network of computers. Instead, we estimate the communication cost by counting the calls for the basic distributed operations. Counting basic steps as a performance measure is an accepted practice. For example, in distributed protocols where multiple agents solve a constraint satisfaction problem, a common measure is the number of *non-concurrent constraint checks*, which provides an estimate to the overall computation time [51].

2. We do not show the SMC counts for different values of m for the following reason: In the vertical case, the number of SMC calls is completely independent of the number of sites, m.



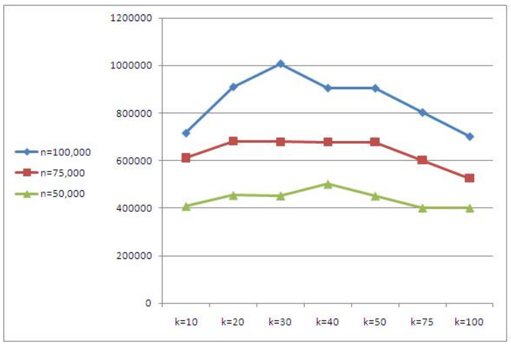


Figure 8. SMC calls on Census: Horizontal (left) and Vertical (right)

In the horizontal case, on the other hand, that number may depend on m and on skewness, because of the efficient implementation of Step 6 that was discussed in the last paragraph of Section 4.2. However, our tests on various values of m (we used $m = 2, 5, 10, 20$) and various patterns of skewness showed that the dependence on these factors is extremely weak and the different plots almost coincided.

3. We do not show the counts of SMC calls for the version of the algorithm that respects ℓ-diversity since, as implied by complexity analysis (Section 7), the added number of messages that are required for the sake of testing the diversity is significantly smaller than the number of messages in the basic version of the algorithm.

We conclude this section by giving some rough estimate of the communication overhead that our SMCs entail. The transmission of a single message is estimated by 1-2 ms (milliseconds) for LAN environments and 30 ms for WAN environments. Since the basic SMC protocols, Algorithms 1 and 2, involve $2m$ messages, their execution time is estimated by $4m$ ms and $60m$ ms in LAN and WAN environments, respectively. Hence, the corresponding estimated runtimes may be derived for each experimental setting from the numbers in Figures 7 and 8.

For example, consider the CENSUS dataset with $100,000$ records. In producing a corresponding anonymization with $k = 50$ in the horizontal setting, Algorithm 3 required roughly 270,000 SMC calls (see Figure 8). If $m = 5$, it translates to 2,700,000 messages. In a LAN environment that would take about 5400 seconds, while in WAN environments it would take about 81,000 seconds (22.5 hours).

In order to compare the time performance of our vertical algorithm to that of the algorithm in [23], we consider the ADULT dataset in the vertical partitioning scenario with $m = 2$ and $k = 100$. As can be seen in Figure 7, Algorithm 4 required roughly 360,000 SMC calls in this case (when executed on a subset of 40,000 records). When $m = 2$, that number translates to 1,440,000 messages. In a LAN environment that would take 2880 seconds, and in a WAN environment about 43,200 seconds (12 hours). The corresponding computational time is roughly 150 seconds (see Figure 5). There is no estimate of the corresponding communication overhead in [23], but the computational overhead alone is over 16 days, as was mentioned above.

10 Conclusions

We begin this concluding discussion with a summary of the comparison between our solution and those offered in [23, 24, 34, 50] with respect to several evaluation criteria:

GENERALITY. The following table compares the different studies with regard to the settings which they considered, number of players, the generalization model, and privacy model. (KA, LD, and LSD stand for k-anonymity, ℓ-diversity, and ℓ-site diversity, respectively.)

Study	Setting	#players	Generalization model	Privacy model
[23]	vertical	two	global and local recoding	KA
[24]	horizontal	any	global recoding	KA, LSD
[34]	vertical	any	global recoding	KA, LD
[50]	horizontal	any	suppressions only	KA
Ours	horizontal and vertical	any	global and local recoding	KA, LD, LSD

CRYPTOGRAPHIC ASSUMPTIONS. The solution in [23] relies upon homomorphic encryptions. [24] uses a secure kth-ranked element protocol (depending on oblivious transfer and pseudorandom functions). [34] assumes a secure maximum protocol (depending on oblivious transfer). The protocol in [50] uses homomorphic encryptions, group exponentiations, and secret sharing. Our solution requires a secure sum protocol and a secure hash function; both are much simpler and have significantly smaller computational costs than the protocols upon which the previous studies rely.

PRIVACY. The protocols of [23, 34] offer perfect privacy. The protocols in [24, 50] and ours leak some information. (See the full discussion in Section 8.)

EFFICIENCY. The studies [24, 50] contain no simulation or evaluation of time performance. The protocol of [23] is inefficient. On the other hand, that of [34] is very efficient. Our protocols are efficient, and are also highly scalable since the number of SMC calls in it depends linearly on the database size. (See the full discussion in Section 9.2.)

UTILITY. The protocols of [24, 34, 50] offer low utility since they are based on global recoding generalizations. The study [23] does not provide sufficient testing in order to assess properly its utility. Our protocols offer very high utility since they are based on a leading state of the art local recoding anonymization algorithm. (See the full discussion in Section 9.1.)

In conclusion, we presented a general approach to secure distributed computations of anonymized views of shared databases. The generality of our approach is with regard to the type of partitioning, the number of sites, the generalization model, and the privacy model. The presented algorithms are highly efficient and simple, as they rely on very basic and few cryptographic primitives. Even though we focused here on distributed versions of one particular algorithm (sequential clustering) and one particular goal (anonymization), the ideas and techniques that were presented here are suitable for any other algorithm that reorganizes clusters (like simulated annealing or k-means) and could be applicable also for other distributed data mining problems.

Acknowledgement. The authors thank Ido Tamir for his help in conducting the experiments, and the anonymous referees for their useful comments.

References

[1] C.C. Aggarwal and P.S. Yu. A condensation approach to privacy preserving data mining. In *EDBT*, pages 183–199, 2004.

[2] G. Aggarwal, T. Feder, K. Kenthapadi, S. Khuller, R. Panigrahy, D. Thomas, and A. Zhu. Achieving anonymity via clustering. In *PODS*, pages 153–162, 2006.

[3] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Anonymizing tables. In *ICDT*, volume 3363 of LNCS, pages 246–258, 2005.

[4] G. Aggarwal, N. Mishra, and B. Pinkas. Secure computation of the kth-ranked element. In *EUROCRYPT*, 2004.

[5] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *ACM-SIGMOD Conference on Management of Data*, pages 439–450, May 2000.

[6] R. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, 2005.

[7] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *STOC*, pages 503–513, 1990.

[8] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP - A system for secure multi-party computation. In *CCS*, pages 257–266, 2008.

[9] J.W. Byun, A. Kamra, E. Bertino, and N.Li. Efficient k-anonymization using clustering techniques. In *DASFAA*, 2007.

[10] G. Cormode. Personal privacy vs population privacy: learning to attack anonymization. In *KDD*, pages 1253–1261, 2011.

[11] J. Domingo-Ferrer, A. Martínez-Ballesté, J.M. Mateo-Sanz, and F. Sebé. Efficient multivariate data-oriented microaggregation. *VLDB J.*, 15:355–369, 2006.

[12] C. Dwork. Differential privacy. In *ICALP (2)*, pages 1–12, 2006.

[13] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[14] K. El Emam and F. Dankar. Protecting privacy using k-anonymity. *Journal of the American Medical Informatics Association*, 15:627 637, 2008.

[15] K. El Emam et al. A globally optimal k-anonymity method for the de-identification of health information. *Journal of the American Medical Informatics Association*, 16:670–682, 2009.

[16] R. Fagin, M. Naor, and P. Winkler. Comparing Information Without Leaking It. *Communications of the ACM*, 39:77–85, 1996.

[17] G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis. A framework for efficient data anonymization under privacy and accuracy constraints. *ACM Trans. Database Syst*, 34, 2009.

[18] A. Gionis, A. Mazza, and T. Tassa. k-Anonymization revisited. In *ICDE*, pages 744–753, 2008.

[19] A. Gionis and T. Tassa. k-Anonymization with minimal loss of information. *TKDE*, 21:206–219, 2009.

[20] J. Goldberger and T. Tassa. Efficient anonymizations with enhanced utility. *TDP*, 3:149–175, 2010.

[21] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[22] V. Iyengar. Transforming data to satisfy privacy constraints. In *ACM-SIGKDD*, pages 279–288, 2002.

[23] W. Jiang and C. Clifton. A secure distributed framework for achieving k-anonymity. *The VLDB Journal*, 15:316–333, 2006.

[24] P. Jurczyk and L. Xiong. Distributed anonymizations: Achieving privacy for both data subjects and data providers. In *Data and Applications Security*, pages 191–207, 2009.

[25] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Trans. Knowl. Data Eng.*, 16:1026–1037, 2004.

[26] B. Kenig and T. Tassa. A practical approximation algorithm for optimal k-anonymity. *Data Mining and Knowledge Discovery*, 2012.

[27] D. Kifer. Attacks on privacy and definetti's theorem. In *SIGMOD Conference*, pages 127–138, 2009.

[28] S. Kirkpatrick, D. Gelatt Jr., and M.P. Vecchi. Optimization by simmulated annealing. *Science*, 220(4598):671–680, 1983.

[29] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Incognito: efficient full-domain k-anonymity. In *ACM-SIGMOD International Conference on Management of Data (SIGMOD)*, pages 49–60, 2005.

[30] K. LeFevre, David J. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *ICDE*, 2006.

[31] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and ℓ-diversity. In *ICDE*, pages 106–115, 2007.

[32] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-Diversity: privacy beyond k-anonymity. In *ICDE*, page 24, 2006.

[33] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *PODS*, pages 223–228, 2004.

[34] N. Mohammed, B.C.M. Fung, K. Wang, and P.C.K. Hung. Privacy-preserving data mashup. In *EDBT*, pages 228–239, 2009.

[35] M.E. Nergiz, E. Çiçek, and Y. Saygm. A look ahead approach to secure multi-party protocols. *TKDE*, 2011.

[36] M.E. Nergiz and C. Clifton. Thoughts on k-anonymization. In *ICDE Workshops*, page 96, 2006.

[37] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *STOC*, pages 75–84, 2007.

[38] H. Park and K. Shim. Approximate algorithms for k-anonymity. In *ACM-SIGMOD*, pages 67–78, 2007.

[39] P. Samarati. Protecting respondent's privacy in microdata release. *IEEE TKDE*, 13:1010–1027, 2001.

[40] P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information. In *PODS*, page 188, 1998.

[41] Latanya Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10:571–588, 2002.

[42] T. Tassa, A. Mazza, and A. Gionis. k-Concealment: An alternative model of k-type anonymity. *Transactions on Data Privacy*, 2012.

[43] T.M. Truta, A. Campan, and P. Meyer. Generating microdata with p-sensitive k-anonymity property. In *SDM*, pages 124–141, 2007.

[44] R.C. Wong, J. Li, A.W. Fu, and K. Wang. (α, k)-anonymity: an enhanced k-anonymity model for privacy preserving data publishing. In *In ACM SIGKDD*, pages 754–759, 2006.

[45] R.C.W. Wong, J. Li, A.W.C. Fu, and K. Wang. (α, k)-anonymity: An enhanced k-anonymity model for privacy preserving data publishing. In *KDD*, pages 754–759, 2006.

[46] X. Xiao and Y. Tao. Anatomy: Simple and Effective Privacy Preservation. In *VLDB*, 2006.

[47] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A.W.-C. Fu. Utility-based anonymization using local recoding. In *KDD*, pages 785–790, 2006.

[48] A.C. Yao. Protocols for secure computation. In *FOCS*, pages 160–164, 1982.

[49] J. Yuan, Q. Ye, H. Wang, and J. Pieprzyk. Secure computation of the vector dominance problem. In *IPSEC*, pages 319–333, 2008.

[50] S. Zhong, Z. Yang, and R.N. Wright. Privacy-enhancing k-anonymization of customer data. In *PODS*, pages 139–147, 2005.

[51] R. Zivan and A. Meisels. Message delay and discsp search algorithms. *Ann. Math. Artif. Intell.*, 46:415–439, 2006.