CASTLE: A δ-constrained scheme for k_s-anonymizing data streams

Jianneng Cao[1] Barbara Carminati[2] Elena Ferrari[2] Kian Lee Tan[1]

[1]School of Computing, National University of Singapore [2]DICOM, University of Insubria, Italy

Abstract

Most of existing privacy preserving techniques, such as k*-anonymity methods, are designed for static data sets. As such, they cannot be applied to streaming data which are continuous, transient and usually unbounded. Moreover, in streaming applications, there is a need to offer strong guarantees on the maximum allowed delay between an incoming data and its anonymized output. To cope with these requirements, in this paper, we present* CASTLE *(*Continuously Anonymizing STreaming data via adaptive cLustEring*), a cluster-based scheme that anonymizes data streams on-the-fly and, at the same time, ensures the freshness of the anonymized data by satisfying specified delay constraints. We further show how CASTLE can be easily extended to handle l-diversity. Our extensive performance study shows that CASTLE is efficient and effective.*

CID	Name	Sex	Zipcode	Birthday
01	Mike	M	53708	1/21/76
02	Alice	F	53715	4/13/86
03	John	M	53703	21/11/79
04	Bob	M	53706	23/4/81
05	Beth	F	53703	7/5/83
06	Carol	F	53706	8/6/85

Table 1: Customer table

CID	Name	Sex	Zipcode	Birthday
01	Mike	M	[53703-53708]	[76-81]
03	John	M	[53703-53708]	[76-81]
04	Bob	M	[53703-53708]	[76-81]
02	Alice	F	[53703-53715]	[83-86]
05	Beth	F	[53703-53715]	[83-86]
06	Carol	F	[53703-53715]	[83-86]

Table 2: statically 3-anonymized customer table

1 Introduction

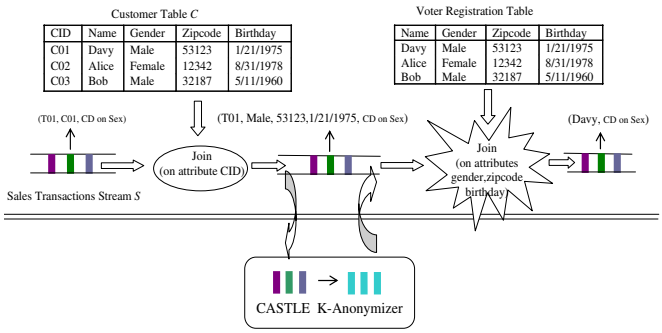


Figure 1: Linking attacks on data streams

In many applications, data arrive in the form of high speed data streams that may contain a lot of private information. Examples of such applications include telecommunication, market-basket analysis, network monitoring, sensor networks, and so on. As such, while streaming data offer an opportunity to analyze and study specific behaviors, user privacy may be compromised either directly or indirectly. As an example, consider Amazon.com. In a single day, it records hundreds of thousands of online sales transactions, which are received in the form of streaming data. Suppose the sales transaction stream has the schema $S(tid, cid, goods)$, where a relation C containing the information about existing customers is stored on disk, with schema $C(cid, name, gender, birthday, zipcode, address, telephone, credit\text{-}card\#)$. Suppose moreover that, to analyze customers' buying behavior, a mining algorithm operates on a data stream obtained by joining S with relation C. To protect users' privacy, attributes that explicitly identify the customers (such as $name$, $address$ and $telephone$) are projected out of the join result. However, in many cases, the remaining data may still be vulnerable to *linking attacks*: data can be used to re-identify individuals by linking or matching the data to other data sources. Figure 1 illustrates an example where customers can be re-identified by joining attributes such as $gender$, $birthday$ and $zipcode$.

Note that the naive solution of joining S with a k-anonymized version of the relation C is not always adequate. This is due to four main reasons. The first is that

the dimensions (i.e., quasi-identifier attributes) to be anonymized may come from both C and S. Thus, the solution of anonymizing only C is not useful since the resultant joined stream must be further anonymized. The second is that each update on the static relation could trigger a re-anonymization of the whole relation and, as consequence the regeneration of joined data stream. The third is due to the fact that once the value of k is changed or the quasi-identifiers are re-defined, a new anonymized version of C should be generated. In addition, the naive solution does not ensure the principle of *l*-diversity [13]. The resultant joined stream only *k*-anonymizes quasi-identifier attributes from C. It should be further processed to make the sensitive attributes from S *l*-diverse. Assume Table 1 is a customer relation, in which quasi-identifier attributes are *sex, zipcode, birthday*. Table 2 is a 3-anonymized version of Table 1. Let $k = 3$ and $l = 2$. Assume Bob bought sex video, John bought vegetable, and Carol bought a book. By joining these three transactions with Table 2 three tuples are generated: (M, [53703-53708], [76-81], sex video), (M, [53703-53708], [76-81], vegetable), and (F, [53703-53715], [83-86], book). The first two together are 2-diverse. But the last one alone cannot be. So the three tuples should be anonymized again to ensure l-diversity and the following three tuples are generated: ([M-F], [53703-53715], [76-86], sex video), ([M-F], [53703-53715], [76-86], vegetable), ([M-F], [53703-53715], [76-86], book). There is a further relevant motivation that makes traditional *k*-anonymity schemes not suitable for data streams: they rely on the assumption that each record in a data set is associated with a different person, that is, that each person appears in the data set only once. Although this assumption is fine in a static setting, this is not realistic for streaming data. For instance, consider again the example of Amazon.com and assume that one customer purchased $n \geq k$ items. After joining these transactions with customer relation C, n tuples with the same quasi-identifier will appear in the resulting stream. By traditional *k*-anonymization schemes they can be output immediately since $n \geq k$. However, the quasi-identifier can be linked with voting list. Thus, the customer is re-identified.

From the above analysis, it is clear that a mechanism is needed to ensure that streaming data can be anonymized without violating user privacy. While numerous *k*-anonymization techniques have been developed for protecting privacy in traditional (relational) databases (see Section 2 for a survey), these methods cannot be directly applied on streaming data, because they are designed for static data sets. Even for the approaches [5] that anonymize incremental data sets, memory is typically too small relative to the stream data size. Another important characteristic of data streams is their *temporal dimension*. Streaming data arrive at a certain rate, they are dynamically processed, and the result is output with a certain delay. In some applications, the output data are immediately used to trigger appropriate procedures. For example, in a sensor network applications the output stream can be used to real-time react to some anomalous situations, and the time to react is very crucial. Therefore, the application receiving the output stream should have strong guarantees on the maximum delay of the output data. To cope with these requirements, in this paper, we present *CASTLE* (*Continuously Anonymizing STreaming data via adaptive cLustEring*), a cluster-based scheme that k-anonymizes streams on the fly and, at the same time, ensures the freshness of anonymized data by satisfying specified delay constraints. Moreover, we propose an extension of CASTLE to apply the *l*-diversity principle [13] to data streams. To the best of our knowledge, this is the first reported work that considers *k*-anonymity and *l*-diversity on data streams.

The basic idea of the proposed approach is to exploit quasi-identifier attributes to define a metric space: tuples are modeled as points in this space. CASTLE groups incoming tuples in clusters and releases all tuples belonging to the same cluster with the same generalization. CASTLE supports both numerical and categorical attributes anonymization, by generalizing the latter through domain generalization hierarchies, and the first through intervals. Clustering of tuples is further constrained by the need to have fresh anonymized data. To cope with this requirement, CASTLE ensures that the delay between a tuple's input and its output is at most equal to a given input parameter δ. We refer to this constraint as δ *constraint*. When a tuple is going to expire (i.e., its delay is equal to $\delta - 1$), CASTLE immediately releases it. Obviously, it could be the case that an expiring tuple does not belong to a cluster with size at least *k*. To manage this case, CASTLE implements a merge and split technique to obtain a cluster with size at least *k* and whose generalization minimizes the information loss. Additionally, to reduce information loss, CASTLE exploits a strategy that allows the reuse of clusters. When a cluster is anonymized and output all its tuples, CASTLE still keeps it (a.k.a. the corresponding generalization) in memory to anonymize newly arriving tuples if necessary. However, we found that adopting a naive reuse strategy is flawed *even if it strictly follows the definition of k-anonymity on static data sets*, since it is vulnerable to inference attacks similar to those arising in the anonymization of incremental data sets [5]. In the paper, we present the reuse strategy employed by CASTLE to avoid such privacy breaches.

The rest of this paper is organized as follows. In the next section, we review related work. Section 3 defines *k*-anonymity and δ constraint for data streams. In Section 4, we present CASTLE scheme, the algorithms and some formal results. Section 5 reports experimental results. Finally, Section 6 concludes the paper.

2 Related work

Our work attempts to integrate two fields: data stream management and privacy preserving data management. Data stream processing issues (e.g., continuous query processing, data stream mining) have received much attention in recent years, and we refer readers to [8] for a survey. Since we can consider these issues orthogonal to our work, in this section we focus on work on privacy preserving data management.

In recent years, issues related to safeguarding the privacy of traditional databases have been well studied. A variety of techniques have been proposed to represent and mine the data without loss of privacy, including methods such as perturbation [1, 3, 10] and k-anonymity [4, 9, 12]. Perturbation-based methods perturb individual data values or the results of queries by swapping [10], condensation [1], or adding noise [3]. The drawback of perturbation-based methods is that they impair data integrity. Unlike the outcome of perturbation-based methods, the records within a *k*-anonymized data set remain true. A data set T is said to satisfy the *k*-anonymity property with respect to attribute set *Q* if each combination of values of *Q* in *T* occurs at least *k* times; here, the attribute set Q is termed as *quasi-identifier* attributes and can be used to link with external information.

The concept of *k*-anonymity has been proposed by Samarati and Sweeney in [14], where the notion of generalization has been presented to ensure *k*-anonymity. In particular, generalization implies that a quasi-identifier attribute value is replaced by a less specific but semantically related value. For instance, the Sex domain $\{male, female\}$ can be generalized to $\{person\}$. This operation is formally defined as value generalization. Since the work in [14], a lot of different schemes have been proposed to achieve *k*-anonymity. These work can be roughly divided into two groups: domain generalization hierarchy based approaches (DGH based approaches for short) and domain partition based approaches (DP based approaches for short).

DGH based approaches achieve *k*-anonymity using the idea of generalization and suppression according to pre--defined domain generalization hierarchies [7, 11, 17, 16]. The Datafly approach [17] of Sweeney uses greedy heuristic to iteratively generalize the combinations whose frequency is below *k*. Datafly cannot guarantee good information quality. Samarati [16] proposes a binary search algorithm for finding a minimal generalization on the DGH lattice. Though it avoids exhaustively searching the entire generalization space, it is not efficient because the number of generations in one level of DGH lattice may be large. Incognito [11] achieves *k*-anonymity more efficiently, by exploiting a bottom-up bread-first search. Finally, Fung et al. [7] propose a top-down approach for producing anonymized data.

DP based approaches do not need user-defined domain generalization hierarchies. They divide quasi-identifier attributes domains into sets of partitions, each partition could be represented by a symbolic value that denotes the partition. Iyengar [9] proposes a single-dimensional partitioning model, in this model the domain of each quasi-identifier attribute is totally ordered, and each domain is divided into a set of disjoint intervals. Iyengar proposes a genetic algorithm to find *k*-anonymity, however, the solution is not guaranteed to be optimal. Bayardo and Agrawal [4] present an exhaustive tree-search method to find the optimal *k*-anonymity, which exploits both cost-based pruning and dynamic search rearrangement. LeFevre et al. extend the single-dimensional partitioning model to the multidimensional partitioning model [12], in which each partition is defined as multidimensional space region. They prove that the *k*-anony-mous strict multidimensional partitioning decision problem is NP-complete, and give a greedy algorithm that produces anonymizations which are a constant factor approximation of the optimal.

In general, DGH based approaches are more suitable for categorical attributes while DP based approaches are more suitable for continuous attributes. The common goal of the above discussed work is to pursue the best quality of anonymous data. However, it is interesting to note that except [1], they all need to scan the data multiple times, and output the anonymous data set in the final step. However, for streaming data the algorithm can only scan the data in one pass and executes in a pipeline manner.

Cluster-based algorithms for *k*-anonymizing static data sets have been proposed [2]. In particular, [2] presents a theoretical framework for grouping tuples in clusters of at least *k* tuples and publishing tuples by replacing the quasi-identifier attributes with the center of the corresponding cluster. However, this approach does not support domain generalization hierarchies and manages categorical attributes like simple discrete values, thus losing relevant semantic information that a domain generalization hierarchy can provide. Another work related to ours is [1], where a cluster-based scheme for k-anonymizing data is presented. Like the scheme presented in this paper, the approach presented in [1] requires a single pass over the data, by generating clusters on-the-fly. More precisely, it creates some clusters using historical data and pushes a new tuple to the nearest cluster (i.e., the cluster's center having the minimum distance from the tuple). Then, after all the tuples from the data set have been processed, tuples in each cluster are generalized and output. Since a data stream is an unbounded flow of tuples, this approach can hardly be applied, due to memory limitation. Moreover, it does not support categorical attributes nor time constraints.

Recently, Machanavajjhala et al. [13] proposed the *l*-diversity principle, which has been introduced to avoid possible inference attacks on k-anonymized data. Ensuring *l*-

diversity implies that all tuples with the same generalization values should have at least *l* distinct values for each different sensitive attribute. Along the same line is the *t*-closeness [6], which requires that the distribution of sensitive attributes inside tuples with the same generalization values is analogous to the distribution of the whole data set (i.e., the distance between the two distributions should be no more that *t*).

Recently, the problem of anonymizing dynamic data sets has started to be investigated. Byun et al. [5] proposed some algorithms to manage insertions of new tuples into an l-diverse table (i.e., a table that satisfies the l-diversity principle), by, at the same time, keeping the l-diversity property and avoiding inferences. Xiao and Tao [18] proposed *m-Invariance* to support the re-publication of a table after it has been updated with both insertion and deletion. However, none of these approaches is easily applicable to data streams for two main reasons. First, since streams are unbounded flows of tuples, the limited memory would be easily swarmed by the potentially large number of waiting tuples. Second, they doe not support time constraints.

3 k_s-anonymity of data streams

A data stream can be modeled as an infinite append-only sequence of tuples with an incremental order that stores, together with standard attributes, also information about when the data have been collected. This is usually modeled as an additional attribute storing the time of origin of the corresponding tuple, or the position of the tuple inside the stream. Without loss of generality, we will consider the tuple position throughout the paper. Thus, given a tuple t in a stream S, we denote with $t.p$ the attribute of t storing the position of tuple t. As discussed traditional *k*-anonymity definition is not adequate for data stream. Therefore, we revise the standard *k*-anonymity principle to take into account the continuity of data streams, and to relax the assumption that each record in the data set is associated with a different person.

Definition 3.1 (Data stream k_s-anonymity)**.** Let S(p, pid, a_1,...,a_j, q_1,...,q_n) be a stream, where Q=$\{q_1,\ldots,q_n\}$ are quasi-identifiers, pid is the person's identity, p is tuple's position, and $a_1, \ldots, a_j$ are the remaining attributes. Let S_{out} be the anonymized stream generated from S where p and pid have been pruned. Given a position P, we say that S_{out} is k_s*-anonymized up to* P, if:

- Any tuple $t \in S$ with position $t.p < P$ is output.
- Let $\Pi_Q(S_{out})$ be the projection of S_{out} on Q. Let s be an element in $\Pi_Q(S_{out})$, and $T_s \subseteq S_{out}$ be the set of tuples such that $T_s = \{t \in S_{out} | t.q_j = s.q_j, j \in [1,n]\}$. Let $SP(s)$ be the set of distinct individuals to which tuples in T_s refer to. For any possible $s \in \Pi_Q(S_{out})$, $|SP(s)| \geq k$.

A relevant property of k_s-anonymized data is their *freshness*. This can be considered as the maximum allowed time of a tuple staying in the memory before it is output, which can be formally defined as follows.

Definition 3.2 (δ delay constraint)**.** Let A be a *k*-anonymization scheme that takes as input a data stream S_{in} and generates in output a data stream S_{out}, and let δ be a positive integer. We say that A satisfies the δ *delay constraint* if and only if for each new tuple $t \in S_{in}$ with position $t.p$, all tuples with position less than $t.p-\delta$ have already been output by A.

By the definition, when a new tuple t arrives, the tuple t' with position $t'.p = t.p - \delta$ can still stay in the memory. However, when the next new tuple with position $t.p + 1$ comes, t' should already have been output. Therefore, once t has arrived, t' is *expiring* and need to be output. In the following, we say that a tuple *is going to expire* if it must be output before the next new tuple arrives. Note that the δ parameter can be tuned on the basis of the application domain and its temporal requirements, making it possible to trade off between the allowed delay and the obtained information quality.

4 The CASTLE framework

In the following sections, before presenting and discussing CASTLE in more details, we introduce the adopted information loss metric and some needed definitions.

4.1 Information loss metrics

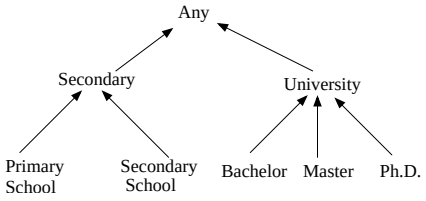


Figure 2: Domain generalization hierarchy of Edu

As we are dealing with streaming data, we need an information loss metric that can be calculated in an incremental manner. We adapt the general information loss metric presented by Iyengar [9] to streaming data. Let *Q*=$\{q_1, \ldots, q_n\}$ be a set of quasi-identifier attributes. Consider first a categorical attribute q_i and let DGH_i be the domain generalization hierarchy for q_i. Given a node *v* in DGH_i, the information loss of *v* is defined as follows:

$$VInfoLoss(v) = \frac{|S_v|-1}{|S|-1}$$

where S_v is the set of leaf nodes of the subtree rooted at *v* in DGH_i and *S* is the set of all the leaf nodes in DGH_i. Intuitively the information loss of a leaf node is 0 according to the formula above. For example, considering the domain generalization hierarchy in Figure 2, the information loss of value "University" is 2/4. In contrast, given a continuous attribute a and an interval *I* = $[l, u]$ from the domain $[L, U]$ of a, used to generalize a's value, the information loss associated with *I* is defined as follows:

$$VInfoLoss(I) = \frac{u-l}{U-L}$$

Hence, we define the information loss of a tuple generalization $g = (v_1, ..., v_n)$ as follows:

$$InfoLoss(g) = \frac{1}{n} \sum_{i=1}^{n} VInfoLoss(v_i)$$

Example 1. Let us consider the stream Customer(P, a_1,..., a_j, Age, Edu), where Age and Edu are the quasi-identifier attributes. With respect to the domain generalization hierarchy in Figure 2 and assuming the interval [0,100] as the domain of the Age attribute, the tuple generalization g=($[1, 40], University$) has the following information loss: $InfoLoss(g)$ = 1/2*(2/4 + 39/100)=0.445.

Given a data stream S k_s-anonymized up to position P, we can define the average information loss of S up to P as follows:

$$AvgLoss(S, P) = \frac{1}{P} \sum_{t_i \in S, t_i.p \leq P} InfoLoss(t_i)$$

4.2 k_s-anonymized clusters

In CASTLE, clusters are defined as n-dimensional intervals, where n is the number of quasi-identifier attributes. The formal definition of cluster is given below.

Definition 4.1 (Cluster over a data stream)**.** Let S(p, pid, a_1,..., a_j, q_1, ..., q_n) be a stream where Q={q_1, q_2, ..., q_n} are the quasi-identifier attributes. Let S' be a set of tuples in S. A cluster C over S' is defined as a set of intervals, called *range intervals*, in the quasi-identifier attribute domains. More precisely, for each $q_i \in$ Q, the corresponding range interval r_i is defined as follows:

- if q_i is a continuous attribute, r_i is the minimal subinterval of q_i's domain that contains all values of q_i of tuples in S';
- if q_i is a categorical attribute, let $Leaves(DGH_i)$ be the set of leaves of DGH_i generated by a left-most traversal of all the leaves in DGH_i. Let L_{q_i} be the smallest subset of $Leaves(DGH_i)$ containing all values of q_i of tuples in S', r_i's bounds are the left-most and the right-most values of L_{q_i}, respectively.

Given a cluster C we denote with $C.size$ the number of distinct persons to which the tuples in C refer to. This can be easy calculated by considering the number of distinct values of the pid attribute. Moreover, we denote with $C.r_i$ the i-th range interval of C, and with C(r_1,...,r_n) the cluster together with its range intervals.

Definition 4.2 (k_s-anonymized cluster)**.** Let C(r_1,...,r_n) be a cluster with size at least *k*. We associate with C the tuple generalization (g_1,...,g_n), where for each $r_i, i \in [1, n]$, g_i is defined as follows:

- if r_i is defined on a continuous attribute q_i, $g_i = r_i$;
- if r_i is defined on a categorical attribute q_i, g_i is set equal to the lowest common ancestor wrt DGH_i of the bounds of r_i.

Moreover, we say that a tuple t is output with C's generalization, if each quasi-identifier attribute $q_i, i \in [1, n]$, of t is replaced by the corresponding value in the tuple generalization associated with C. If all C's tuples are output with C's generalization at time instant j, we say that, starting from j, C is a k_s-anonymized cluster.

Example 2. Consider the stream Customer(p, pid, $a_1, \ldots,$ a_j, Age, Edu) and the following tuples:[1] (pid_1,25, Bachelor), (pid_2,26,Master), and (pid_3,30,Ph.D.). According to Definition 4.1, cluster C defined over these three tuples has [25,30] as Age range interval and [Bachelor, Ph.D.] as Edu range interval. Therefore, it is denoted as C([25,30], [Bachelor,Ph.D]) with size equal to 3. If we further add the tuple (pid_1,25,Bachelor)[2] to C, it does not change its range intervals nor its size (which is still equal to 3). If $k \leq 3$, this cluster can be k_s-anonymized. According to the domain generalization hierarchy presented in Figure 2, tuples in C are anonymized as ([25,30],University), since University is the lowest common ancestor of Bachelor and Ph.D.

4.3 The CASTLE scheme

In this section we present CASTLE. We start by giving a general overview of the approach. Detailed algorithms are presented in Section 4.4.

4.3.1 Scheme overview

Initially, no clusters are in memory. When CASTLE receives the first tuple, it generates a cluster over it. Then, for every newly arriving tuple t, among all the existing clusters CASTLE selects one to which t is assigned. CASTLE

[1] For simplicity, here and in the following we only consider the pid and quasi-identifier attributes.

[2] In data stream scenario two or more tuples may refer to transactions of the same person.

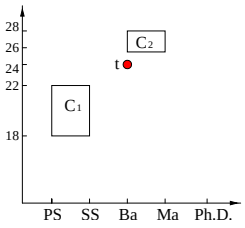


Figure 3: Cluster selection

always selects the cluster which requires the *minimal enlargement* to enclose t, since this reduces information loss. Cluster enlargement is formally defined as follows.

Definition 4.3 (Cluster enlargement)**.** Let S(p, pid, a_1,..., a_j, q_1, ..., q_n) be a stream where Q=$\{q_1,\ldots,q_n\}$ are the quasi-identifier attributes. Let $C(r_1, r_2, ..., r_n)$ be a cluster defined over tuples in $S' \subset S$, where $(g_1, g_2, ..., g_n)$ is the associated generalization, and let t be a tuple in $S \setminus S'$. The enlargement of C w.r.t. t is defined as follows:

$$Enlargement(C,t) = \sum_{i=1}^{n}(InfoLoss(\tilde{g}_i) - InfoLoss(g_i))$$

where $(\tilde{g}_1, \tilde{g}_2, ..., \tilde{g}_n)$ is the generalization associated with C calculated with the new range intervals $(\tilde{r}_1, \tilde{r}_2, ..., \tilde{r}_n)$, computed such that C contains tuples in $S' \cup \{t\}$.

Example 3. Consider clusters C_1 and C_2 in Figure 3. To enclose tuple t into cluster C_1 its range intervals should be enlarged to [18,24] and [Primary School,Bachelor], respectively, which implies that the new generalization associated with C_1 is ([18,24],Any). $Enlargement(C_1, t)$= $1/2 * (4/4 + 6/100) - 1/2 * (1/4 + 4/100) = 0.509$. In contrast, the range intervals of C_2 generated to enclose t are [24,28] and [Bachelor,Master], respectively, which correspond to the new generalization associated with C_2: ([24,28], University). $Enlargement(C_2, t)$=$1/2 * (2/4 + 6/100) - 1/2 * (2/4 + 2/100) = 0.02$.

To prevent clusters from becoming too big, which implies generalizations with poor information quality, if pushing a new tuple to any existing cluster makes the information loss of the cluster greater than a predefined threshold τ, CASTLE generates a new cluster over the new tuple (see Section 4.3.2 for more details).

To satisfy δ constraints, when a new tuple arrives, CASTLE checks whether a tuple in some cluster is going to expire. There are two main cases. The first is when C's size is already greater than or equal to *k*. In this case CASTLE simply outputs all the tuples in C with C's generalization. Starting from the instant of outputting all C's tuples, C is a k_s-anonymized cluster. The second case is when the cluster C hosting the expiring tuple has size less than *k*. Before outputting the expiring tuple CASTLE performs some merge operations. In particular, for every *non-*k_s-anonymized cluster C_i, excluding C itself, CASTLE calculates the enlargement of C due to the possible merge with C_i. Then, it selects the cluster, which brings the minimum enlargement to C, and merges C with it. This process continues until C's size is at least *k*. After the merge operations, all tuples in C are output with C's generalization. However, according to the adopted information loss metric, the smaller the cluster is (i.e., its range intervals), the smaller its information loss will be. Therefore, if C's size is at least *2k*, CASTLE splits it into two or more sub-clusters, each with size at least k, before outputting the tuples. The splitting technique we use is adapted from the KNN algorithm in [1]. It works as follows: (a) Select a tuple t randomly from C, and form a sub-cluster C_i over t. (b) For each tuple t_i in C excluding those with the same pid as t, calculate C_i's enlargement due to possible absorption of t_i. *Since* C_i *is composed of only* t*, the enlargement can be seen as the distance between* t *and* t_i. Each group of tuples with the same pid has the same distance to t. Sort the groups in the ascending order of their distance to t. From each of the first $k-1$ groups, randomly select one tuple. These *k*-1 selected tuples, each having a different pid, are pushed to C_i, and are deleted together with t from C. (c) The above steps are repeated until C's size is less than *k*. (d) For the remaining tuples in C, for each group of tuples with the same pid, push them into the sub-cluster, which requires the minimum enlargement to enclose them.

After a cluster becomes k_s-anonymized, it is not deleted from memory rather its generalization is kept. The main advantage of such *cluster reuse* is that the generalizations of k_s-anonymized clusters can be reused for anonymizing expiring tuples that are enclosed in a *non-*k_s-anonymized cluster with size less than *k*. Therefore, when a new tuple arrives CASTLE considers only the set of *non-*k_s-anonymized clusters as possible candidates to accomodate it. Then, every time a tuple t inside a *non-*k_s-anonymized cluster is going to expire, CASTLE verifies whether t falls in a k_s-anonymized cluster KC. If this happens, t is immediately output with the generalization of KC. This avoids some clusters merges and therefore improves information quality.

The reuse strategy allows cluster overlapping, which, if not carefully managed, can be exploited by a potential attacker to infer additional knowledge about the tuple value or even guess its exact value, by analyzing clusters' intersections, as the following example shows.

Example 4. With reference to Figure 4, suppose that at a given instant j both clusters C_2([26,28], [Bachelor, Master]) and C_3([24,27], [Bachelor,Master]) are k_s-anonymized. Suppose moreover that a tuple $\tilde{t}$(pid_7, 25,Master) arrives after instant j and that, after some time,

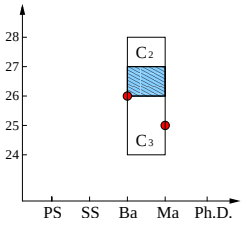


Figure 4: Overlapping clusters

it is going to expire. CASTLE outputs this tuple with C_3's generalization, i.e., ([24,27], University). However, by tracing the output stream, an attacker is able to infer that $\tilde{t}.Age$ belongs to $[24, 25]$, instead of $[24, 27]$. Indeed, an attacker knows the existence of C_2 by tracing the anonymized output stream. Moreover, s/he knows that $\tilde{t}.Age$ is not in $[26, 27]$, otherwise it would have been given in output with C_2's generalization, since this generalization preserves more information wrt C_3 generalization. Thus, an attacker can infer that $\tilde{t}.Age$ belongs to $[24, 25]$.

To overcome this problem, a first naive solution is to avoid overlaps among k_s-anonymized clusters. However, this solution has the drawback of diminishing the possibility of anonymizing expiring tuples by reusing the generalization of k_s-anonymized clusters. Therefore, we adopt a different solution: if an expiring tuple falls into the overlap of two or more k_s-anonymized clusters, CASTLE randomly selects one k_s-anonymized cluster and anonymizes the tuple with its generalization. This avoids the security flaws previously discussed (see Section 4.5 for a formal proof).

Example 5. Consider again Example 4 and assume that the strategy discussed above is adopted. When a potential attacker sees the generalization of tuple $\tilde{t}$, he/she cannot infer any additional knowledge about $\tilde{t}.Age$. Indeed, he/she may infer that $\tilde{t}.Age$ value could be 24, 25 or any value in the intersection between C_2 and C_3 Age range intervals, that is, 26 and 27. Thus, what he/she can infer is that the value of the Age attribute is in the range [24,27], which is exactly the generalization by which $\tilde{t}$ has been output, i.e., C_3's generalization.

4.3.2 Adaptability to data stream distribution

We recall that CASTLE exploits a threshold τ to decide whether pushing a new tuple into any existing cluster or generating a new cluster over the new tuple. In order to adapt to the data stream distribution, we do not consider a predefined and fixed τ. Instead, τ is set to the average information loss of the μ most recent k_s-anonymized clusters, where μ is a parameter given in input, that can be set according to the expected variance of data distribution. Let us see the benefits of this adaptive τ. When a data stream contains well clustered tuples, it is possible to generate over them small clusters with small information loss. Therefore, τ will assume a small value. As a consequence, a new tuple is pushed into a cluster only if it is very close to it. This ensures that if tuples in a stream are well clustered, only clusters with small range intervals are formed. On the contrary, if a data stream contains sparsely distributed tuples, clusters with big range intervals are generated. This implies that τ will have a high value. The benefit of this is that the chance of a new tuple to be pushed into an existing cluster is increased, since the enlargement allowed by τ is increased too. As a consequence, each cluster has increased possibility of reaching the size of k and the number of cluster merging operations is reduced.

Another adaptivity is obtained by constraining the maximum number of *non-*k_s-anonymized clusters according to a given parameter β. This parameter co-operates with τ to handle the varying of data distribution. More precisely, suppose at the beginning tuples are well clustered, so we have a small τ. If the newly arriving tuples are sparely distributed, many small-size clusters will be formed since the small τ only allows a small cluster enlargement. The large number of clusters increases the overhead of searching for the best cluster into which a new tuple should be pushed, and it also increases the overhead of merging clusters when tuples expire. So we adopt β to prevent the generation of a possibly large number of clusters and thus limit the overhead, it can be set by taking into account the available computational and storage resources. Therefore, a new cluster is generated only if the number of *non-*k_s-anonymized clusters is fewer than β. Otherwise, a tuple is pushed to an existing cluster, which requires minimum enlargement to enclose the tuple.

A further adaptivity to data stream distribution is given by management of cluster outlier. Indeed, the presence of few tuples, which are faraway from other tuples, can drastically increases the information loss of the k_s-anonymized stream. To overcome this drawback, CASTLE suppresses an expiring tuple in a cluster, which is considered as an outlier. More precisely, defining a cluster as an outlier is based on three factors. The first is the life time of a cluster. Intuitively, the longer the cluster exists, the bigger its size should be. The second factor is the size of the cluster. Only cluster with *'small'* size should be considered as an outlier. The third is the size of other clusters. This is the key factor. Based on the analysis of τ and β, it is possible that all clusters are small in size at some moment. If this happens obviously a *'small-size'* cluster should not be judged as an outlier if it is larger than most of the other clusters in size. So when a tuple in a cluster expires, how to decide if the cluster is an outlier? Since this cluster has existed for at least δ time, it has a long life time and is supposed to be 'big' in size. We scan all clusters and remember their sizes. If the cluster's size is less than $\alpha \times count$ existing clusters,

it is regarded as an outlier. Here $count$ is the total number of existing clusters, and α is a threshold. For simplicity we assume $\alpha = 0.5$. Here we only suppress the expiring tuple in the outlier, instead of suppressing the whole outlier, because in the future maybe tuples will be pushed to this outlier, and it will be no longer an outlier any more.

Suppressing a tuple means that the tuple is anonymized with the most generalized values. The information loss of this tuple is obviously 1. However, if the tuple is not suppressed, its expiration will require merge operations. It implies that at least *k* tuples are anonymized according to a possibly big cluster in generalization. The overall information loss of the anonymized stream is more impacted by the information loss of the *k* tuples anonymized by a big cluster, rather than by a single tuple whose information loss is 1.

4.4 CASTLE algorithms

In the following we present the algorithms implemented by CASTLE to ensure k_s-anonymity of data streams, then we illustrate their extension to achieve l-diversity.

4.4.1 Algortihms

The main algorithm is Algorithm 1, which continuously processes the incoming data stream by producing in output a flow of k_s-anonymized tuples.

Algorithm 1: *CASTLE*(S, k, δ, μ, β)

```
1  Let Γ be the set of non-k_s-anonymized clusters, initialized to be empty;
2  Let Ω be the set of k_s-anonymized clusters, initialized to be empty;
3  Let τ be initialized to 0;
4  while S is non-empty do
5      Let t be the next tuple from S;
6      Let C be the cluster returned by best_selection(t);
7      if C = NULL then
8          Create a new cluster on t and insert it into Γ;
9      else
10         Push t to C;
11     Let t' be the tuple with position equal to t.p − δ;
12     if t' has not yet been output then
13         delta_constraint(t');
```

The algorithm takes as input the stream S to be anonymized, and the parameters: k, δ, μ and β. At the beginning, the set of *non*-k_s-anonymized clusters (i.e., Γ) as well as the set of k_s-anonymized clusters (i.e., Ω) are empty. Then, every time a tuple t arrives (step 5), Algorithm 1 calls procedure $best\_selection()$ (step 6), to select from Γ the best cluster into which t should be pushed (step 10). If such a cluster does not exist, Algorithm 1 creates a new cluster for t (steps 7-8). Then, Algorithm 1 verifies whether the arrival of the new tuple t forces a tuple t' with position $t.p - \delta$ to expire (step 11). If this is the case, Algorithm 1 calls procedure $delta\_constraint()$ (step 13).

Function `best_selection`(t)

```
1  Let E be a set initialized empty;
2  foreach C_j ∈ Γ do
3      Let e be Enlargement(C_j, t);
4      Insert e into E;
5  Let min be the minimum element of E;
6  Let SetC_min be the set of clusters C̃ in Γ with Enlargement(C̃, t)
   equal to min;
7  foreach C_j ∈ SetC_min do
8      Let IL_{C_j} be the information loss of C_j after pushing t into it;
9      if IL_{C_j} ≤ τ then
10         Insert C_j into SetC_ok;
11 if SetC_ok is empty then
12     if |Γ| ≥ β then
13         Return any cluster in SetC_min with minimum size;
14     else
15         Return NULL;
16 else
17     Return any cluster in SetC_ok with minimum size;
```

To find out the best *non* k_s-anonymized cluster where a new tuple t should be inserted, $best\_selection()$ calculates the enlargement implied by the insertion of t in each cluster in Γ (steps 1-4). Then, it selects from Γ only the clusters with the minimum enlargement (steps 5-6). According to the adaptability strategy described in Section 4.3.2, $best\_selection()$ selects from the returned clusters only those whose information loss is less than or equal to τ (steps 7-10). Among those, $best\_selection()$ selects the cluster with the minimum size (step 17). By contrast, if no clusters are selected, there could be two different cases. The first case is when the number of *non*-k_s-anonymized clusters is greater than or equal to β (step 12). In such a case, it is not possible to create a new cluster, thus among clusters requiring the minimum enlargement it is returned one among those with the minimum size (step 13). Otherwise, the procedure returns a NULL value (step 15).

When a tuple t is expiring, Algorithm 1 calls procedure $delta\_constraint()$. We recall that tuples are only pushed into *non*-k_s-anonymized cluster. Thus, when a tuple is expiring there could be two cases. If t belongs to a cluster C with size greater or equal to *k*, $delta\_constraint()$ calls procedure $Output\_cluster()$ (step 3). Otherwise, $delta\_constraint()$ verifies whether it is possible to apply on the tuple the reuse strategy illustrated in Section 4.3.1 (steps 5-9). If the reuse strategy cannot be applied, $delta\_constraint()$ verifies whether t is in an outlier and can be suppressed (steps 10-16).[3] Otherwise, it verifies whether a merge among C and the other *non*-k_s-anonymized clusters is possible. Notice that, if the total size of all clusters in Γ is fewer than k (step 18), a merge operation would not generate a cluster with size at least k. Therefore, the only way to output the expiring tuple is sup-

[3] See 4.3.2 for a description of the management of cluster outliers.

Procedure `delta_constraint`(t)

1 Let C be the cluster into which t was pushed;
2 **if** $C.size \geq k$ **then**
3 Output_cluster(C);
4 **else**
5 Let KC_{set} be the k_s-anonymized clusters in Ω containing t;
6 **if** KC_{set} *is not empty* **then**
7 Let $\overline{KC}$ be a cluster randomly selected from KC_{set};
8 Output t with $\overline{KC}$ generalization;
9 Return;
10 Let m be an integer set to 0;
11 **foreach** $C_j \in \Gamma$ **do**
12 **if** $C.size < C_j.size$ **then**
13 $m = m + 1$;
14 **if** $m > 0.5 * n$ **then**
15 Suppress tuple t;
16 Return;
17 Let n be the number of clusters in Γ;
18 **if** $\sum_{i=1}^{n} C_i.size < k$ **then**
19 Suppress t;
20 Return;
21 MC =Merge_clusters(C, $\Gamma \setminus C$);
22 Output_cluster(MC);

pressing it (step 19). Otherwise, the merge can take place (step 21), and after the merge, the merged cluster is output (step 22).

Procedure `Output_cluster`(C)

1 **if** $C.size \geq 2k$ **then**
2 Let SC be the set of clusters returned by $split(C)$;
3 **else**
4 Let SC be $\{C\}$;
5 **foreach** $C_i \in SC$ **do**
6 Output all tuples in C_i with its generalization;
7 Update τ according to the information loss of C_i;
8 **if** *Information loss of* $C_i < \tau$ **then**
9 Insert C_i into Ω;
10 **else**
11 delete C_i;
12 delete C_i from Γ;

When outputting a cluster C, in order to minimize information loss, $Output\_cluster()$ verifies whether C can be split (step 1). If this is the case, it calls procedure $split()$ to split C into sub-clusters, each with size at least k. All tuples of the new created clusters (or of C, respectively) are k_s-anonymized (step 6), and τ is updated to be the average information loss of the most recent μ k_s-anonymized clusters including new ones (step 7). However, $Output\_cluster()$ does not store all these new clusters into Ω, the set of k_s-anonymized clusters. Indeed, to minimize information loss, only clusters with good information quality are reused. For this reason, $Output\_cluster()$ inserts a new k_s-anonymized cluster into Ω only if its information loss is less than τ (steps 8-11). In addition, C is deleted from Γ (step 12).

4.4.2 l-diversity

Function `split`(C, a_s)

1 initialize SC= 0;
2 hash tuples of C by their a_s values into buckets, each having only one a_s value;
3 Let BS be the set of all buckets;
4 **while** $|BS| \geq L$ *and* $sum = \sum_{B_i \in BS} B_i.size \geq k$ **do**
5 generate an empty cluster C_{sub};
6 **foreach** B_j **do**
7 randomly select a tuple t_j from B_j;
8 find t_j's $k \times \frac{B_j.size}{sum} - 1$ nearest neighbors;
9 add t_j and its nearest neighbors to C_{sub};
10 delete t_j and its nearest neighbors from B_j;
11 **if** $B_j.size = 0$ **then**
12 delete B_j from BS;
13 add C_{sub} to SC;
14 **foreach** $B_i \in BS$ **do**
15 **foreach** *tuple* $t \in B_i$ **do**
16 find t's nearest cluster in SC, and add t to it;
17 delete B_i;
18 return SC;

Our scheme for k_s-anonymizing streaming data can be easily extended to a scheme that k_s-anonymizes and *l*-diversifies streaming data. To satisfy the *l*-diversity principle we only need to slightly modify procedure *delta_constraint*() and replace function *split*() while keeping the whole framework of CASTLE unchanged. The modification of Procedure *delta_constraint*() is as follows. Step 1 should add a further condition which verifies that cluster C, with a size not less than *k*, also contains at least *L* diverse sensitive values. Similarly, step 18 must be changed by adding the checking of an alternative condition, that is, the union of all clusters in Γ have less than *l* sensitive values.

Function *split*, different from the previous one, takes into account the requirements of the *l*-diversity principle. The function has two arguments: C, that is, the cluster to be split, and a_s, that is, the sensitive attribute. Before splitting, tuples with a same pid are pushed into a group. Then, it randomly marks one tuple in each group. The splitting function only processes the marked ones; all the tuples in a group are pushed to the same sub-cluster as the marked one. In step 2, tuples are hashed into buckets similar to the *l*-diversity (not *k*-anonymity) scheme reported in [15], each bucket having a single sensitive value (step 2). If the number of buckets is larger than or equal to l and the total size of buckets is at least *k* (step 4), the function selects from each bucket B_j a portion of tuples, the number of which is proportional to the size of B_j (steps 7-8). The selected tuples are added to C_{sub}, and are deleted from B_j (steps 9-10). If B_j is empty after deletion, it is deleted from BS (step 11-12). After that the function has selected tuples from each $B_j \in BS$, C_{sub} has a size of k and is added to SC (step 13). While it

is impossible to construct a new *l*-diverse sub-cluster, every remaining tuple in each bucket is added to its nearest cluster in SC (steps 14-17).

4.5 Formal results

In this section, we analyze the correctness and complexity of CASTLE.

4.5.1 CASTLE correctness

To prove the correctness of CASTLE we first introduce a theorem proving that CASTLE produces k_s-anonymized data streams.

Theorem 1. *Let S(p,pid, a_1,..., a_j, q_1, ..., q_n) be a stream where Q=$\{q_1,\ldots,q_n\}$ are the quasi-identifier attributes. Let S_{out} be the stream generated by CASTLE. $\forall P \in N$, S_{out} is k_s-anonymized up to P.*

Proving that CASTLE generates k_s-anonymized data streams is not enough. As shown in example 4, cluster overlaps can be exploited by a potential attacker to infer additional knowledge about the tuple value or even guess its exact value, We have therefore to prove that the reuse strategy of CASTLE avoids these security flaws. Before proving this, we need to introduce some notations. Given a k_s-anonymized tuple t, the value of each quasi-identifier attribute in t denotes a set of possible values of the corresponding attribute in the non-k_s-anonymized tuple. For instance, given the domain generalization hierarchy presented in Figure 2, if the value of the quasi-identifier attribute *Edu* of a k_s-anonymized tuple is University, we can infer that the corresponding non-anonymized tuple can have as value for the *Edu* attribute one of the element in the set {Bachelor, Master, Ph.D.}. Similarly, if the value of the quasi-identifier attribute *Age* of a k_s-anonymized tuple is [25-30], we can infer that the *Age* attribute of the corresponding non-anonymized tuple has a value in the interval [25,30]. Therefore, given a k_s-anonymized tuple t, we denote with $values(t.q_i)$ the set of values implied by the value of $t.q_i$. If q_i is a continuous attribute, $values(t.q_i)$ contains all the values in the interval corresponding to $t.q_i$'s value, whereas if q_i is a categorical attribute, $values(t.q_i)$ contains all the leaves of the subtree in DGH_i rooted at $t.q_i$'s value, where DGH_i denotes the domain generalization hierarchy for attribute q_i.

Theorem 2. *Let S(p,pid, a_1,..., a_j, q_1, ..., q_n) be a stream where Q=$\{q_1,\ldots,q_n\}$ are the quasi-identifier attributes. Let S_{out} be the stream generated by CASTLE. By tracing the generalized values of all tuples $t \in S_{out}$, for each $q_i \in Q$ an attacker is not able to infer that the real value of $t.q_i$ belongs to a subset of $values(t.q_i)$.*

Formal proofs of the both the therems above are reported in Appendix A.

4.5.2 CASTLE complexity

Symbol	Description
C_d	computational cost for enlargement at a single dimension
D	number of quasi-identifier attributes (6∼10)
β	maximal number of *non-*k_s-anonymized clusters (50)
δ	temporal constraint (5,000∼30,000)
k	value of k_s in k_s-anonymity (100∼1,000)
N	number of k_s-anonymized clusters
C_p	cost of replacing the root of a heap ($\log^k$)
S_g	space for tuple generalization at one dimension
S_t	space occupied by one tuple

Table 3: Notation

In order to analyse the efficiency of our approach we have estimated the time and space complexity of CASTLE. In doing that we make use of a set of parameters meaningful for the analysis. These parameters are summarized in Table 3, where the numbers in the parentheses, if present, represent the values used in our experiments (see Section 5).

Time Complexity. We have estimated the time complexity of the main operations carried out by CASTLE, that is: (1) pushing a tuple into a cluster; (2) merging clusters; (3) splitting a cluster.

Pushing a tuple in a cluster. To decide which cluster must absorb a tuple t, CASTLE calculates, for each cluster, the enlargement needed to absorb t. The time required by these operations is upper bounded by:

$$C^u_{push}=(D \cdot C_d) \cdot \beta$$

where $(D \cdot C_d)$ is the cost of calculating the enlargement of a cluster for each dimension in D, and β is the maximum number of existing clusters.

Clusters merge. Let us assume C be the cluster to be merged with other clusters. We recall that the the merge operations is performed by scanning the *non-*k_s-anonymized clusters, calculating the enlargement of C due to the possible merge with one of the non-k_s-anonymized clusters, selecting the cluster that brings the minimum enlargement to C, and merging C with it. This process continues until C's size is at least *k*. To have an upper bound the of the time cost, we need to consider the worst case. According to the adopted cluster merge technique this happens when C is merged with all the existing clusters to obtain a cluster with size at least *k*. Thus, required time is upper bounded by:

$$C^u_{merge}=\sum_{i=1}^{\beta-1}(D \cdot C_d)(\beta - i) = \frac{\beta \cdot (\beta-1)}{2} \cdot (D \cdot C_d)$$

Cluster split. Assume that C is the cluster to be split, and that there are $n \geq 2k$ tuples in C.

We recall that the split technique implies to randomly select a tuple t from C, form a sub-cluster C_i over t, and let C_i absorb *k*-1 tuples of C. The above steps are repeated until C's size is less than *k*. Then, the remaining tuples in C are pushed in the sub-cluster that requires the minimum enlargement to enclose them. Let us see the time complexity of these steps. For simplicity, assume that each tuple of C has a distinct pid. To speed up the process of selecting $k-1$ tuples to be pushed in the sub-cluster C_i formed over the randomly selected tuple t, CASTLE maintains a heap, where the nodes are organized on the basis of their distance to t. More precisely, at the beginning, the heap is initialized with $k-1$ nodes marked with a infinite distance from t. Then, for each tuple $\widetilde{t} \in C$, CASTLE calculates the distance between $\widetilde{t}$ and t: if $\widetilde{t}$ is closer to t than the root of the heap, $\widetilde{t}$ replaces the root and the heap is adjusted. Therefore, the cost of pushing a tuple in the sub-cluster C_i is upper bounded by $(D \cdot C_d + C_p)$. All together $\lfloor n/k \rfloor$ sub-clusters are formed. For each of the remaining $n - k \cdot \lfloor n/k \rfloor$ tuples, pushing it to its nearest sub-cluster takes: $\lfloor n/k \rfloor \cdot (D \cdot C_d)$. Therefore, the total cost is upper bounded by:

$$\begin{aligned} C^u_{split} = & [\textstyle\sum_{i=0}^{\lfloor n/k \rfloor - 1}(n - i \cdot k)(D \cdot C_d + C_p)] + \\ & (n - k \cdot \lfloor n/k \rfloor)(\lfloor n/k \rfloor \cdot (D \cdot C_d)) \\ \approx & \tfrac{n^2}{2k}(D \cdot C_d) \end{aligned}$$

Since C_p cannot require more than log^k (the hight of heap) comparisons, C_p is much smaller than $D \cdot C_d$. Therefore, the total cost is roughly $\frac{n^2}{2k}(D \cdot C_d)$. If tuples are uniformly distributed among *non-*k_s-anonymized clusters, then each cluster has δ/k tuples and the cost is: $\frac{\delta^2}{2k^3}(D \cdot C_d)$. If the distribution of tuples is highly skew, then one cluster may contain almost all the δ tuples and the cost is: $\frac{\delta^2}{2k}(D \cdot C_d)$.

Space Complexity. Let us now consider the space complexity of CASTLE. This is mainly affected by storage information related to three components, that is, the tuples in memory (i.e., in the *non-*k_s-anonymized clusters), the *non-*k_s-anonymized clusters, and the k_s-anonymized clusters. Regarding the first component, it is important to note that the δ constraint ensures that at any instant there are at most δ tuples in memory. Thus, let S_t be the space required to store a tuple, the first component requires $\delta \cdot S_t$ space. For each *non-*k_s-anonymized cluster as well as for each k_s-anonymized cluster, CASTLE stores the corresponding generalization. Let S_g be the space required to store the generalization of a quasi-identifier attribute, this information requires $(D \cdot S_g)$ space. Moreover, according to the strategy adopted by CASTLE, the number of *non-*k_s-anonymized clusters in memory is at maximum β. Whereas, we assume that the number of k_s-anonymized that CASTLE keeps in the memory is equal to N at maximum, where N is a threshold that can determined based on the available memory.[4] Therefore, the total space required by CASTLE is:

$$S_{cost} = \delta \cdot S_t + (\beta + N)(D \cdot S_g)$$

5 Performance evaluation

We have implemented CASTLE and conducted several experiments. We report our results in this section. Our experiments have been designed with two objectives in mind. First we would like to verify that the proposed method is able to continuously anonymize a data stream while keeping the data useful (i.e., with a low information loss). Second, to illustrate the effectiveness of CASTLE, we compare it with the approach presented in [1], which is the one comparable to our approach since it k-anonymizes the data set by a single pass on them. For these experiments, we used both synthetic and real world data. In particular, we have adopted the Adult data set from UC Irvine Machine Learning Repository,[5] which has become a standard for studying k-anonymity. We refer to this as *UCI-Adult*. Moreover, in order to have a better simulation of a data stream, we have also considered the data set used in [7]. We refer to this as *SFU-Adult*. The experiments were conducted on an Intel Pentium IV 2.4GHz with 1 GB RAM.

We configure *UCI-Adult* by removing tuples with missing values. Thus it contains 30,162 tuples. *SFU-Adult* is configured by adding 15,060 extra tuples to *UCI-Adult*, so it has 45,222 tuples. For k-anonymization, our quasi-identifier attributes are selected from the following attributes:*age, final-weight, education-number, capital-gain, capital-loss, hours-per-week, education, martial-status, occupation, nation*. The first six of them are continuous, and the left four are categorical. We adopt from [7] the hierarchies for categorical attributes, and the intervals for continuous attributes. In the following, we evaluate CASTLE using the metric described in Section 4.1.

5.1 Tuning CASTLE

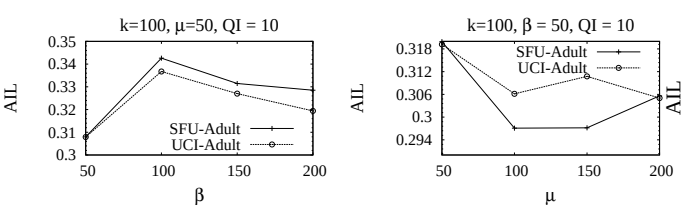
In this section, we evaluate CASTLE with different parameters. The parameters that affect the performance of CASTLE are: δ, *k*, the number of quasi-identifier attributes, μ, β, and the data distribution. In all the figures, AIL represents the Average Information Loss.

Effects of β and μ

CASTLE's adaptability to data distribution is controlled by two parameters: μ, the number of most recent k_s-anonymized clusters on which τ is calculated, and β, the

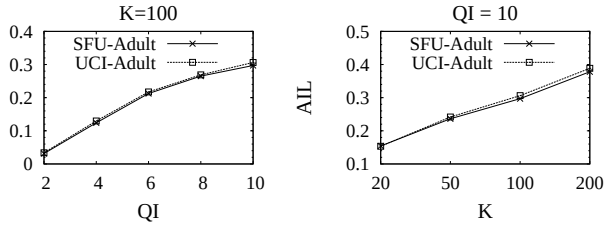
[4] This constraint is satisfied by deleting the k_s-anonymized clusters that are seldom used to generalize a tuple, so that the number of k_s-anonymized clusters is less than N

[5] www.ics.uci.edu/~mlearn/MLSummary.html.



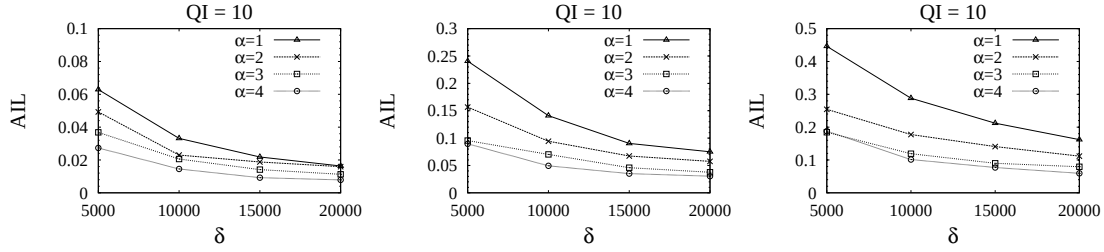
(a) β change (b) μ change

Figure 5: Varying β and μ



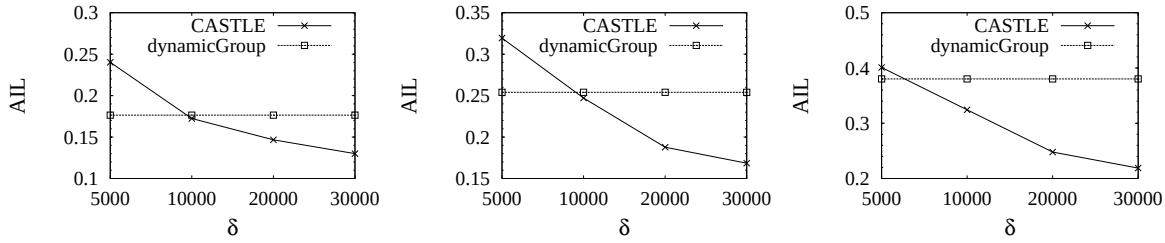
(a) QI change (b) K change

Figure 6: Varying k and quasi-identifier



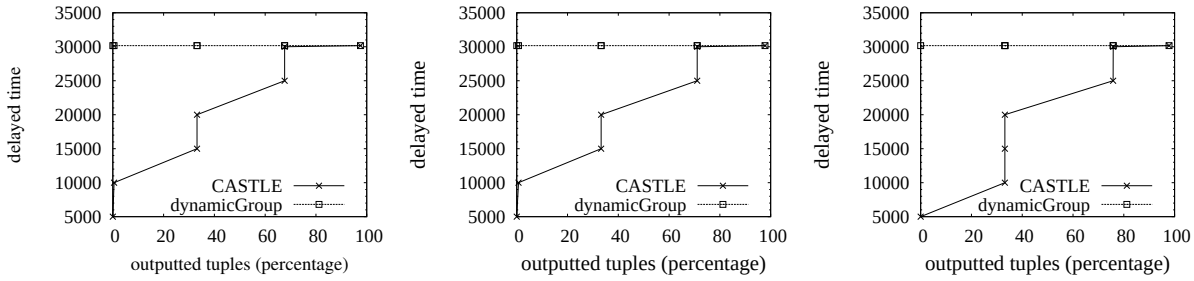
(a) k =100 (b) k = 400 (c) k = 1000

Figure 7: Information loss on power-law synthetic data



(a) k = 100 (b) k = 200 (c) k = 400

Figure 8: Information loss on UCI-Adult



(a) k=100, δ = 10000 (b) k=200, δ = 10000 (c) k=400, δ = 10000

Figure 9: Output percentage on UCI-Adult

threshold for controlling the maximum number of clusters in the memory. Figure 5 presents the average information loss of k_s-anonymized tuples by separately varying β (cfr. Figure 5(a)) and μ (cfr. Figure 5(b)). In both the experiments, we have set $k = 100$, $\delta = 20000$ and have used 10 quasi-identifier attributes. We run the experiments on both the *UCI-Adult* and *SFU-Adult* data sets. In Figure 5(a) we set $\mu = 50$ and increased β. $\beta = 50$ gives the best information quality. So in Figure 5(b) we set $\beta = 50$ and increased μ. When $\mu = 100$, the information loss is minimized. In the following experiments, we shall use $\beta = 50$ and $\mu = 100$ as the default values.

Effects of *quasi-identifier*

A further experiment measures how the number of quasi-identifier attributes affects the average information loss. The experiment has been conducted on the *UCI-Adult* and *SFU-Adult* data sets with $k = 100$, $\beta = 50$, $\mu = 100$, $\delta = 20000$. Figure 6(a) reports how the average information loss varies by increasing the number of quasi-identifier attributes. As expected, the information loss increases when increasing the number of quasi-identifier attributes.

Effects of *k*

Figure 6(b) shows how the average information loss decreases by increasing the value of *k*. In this experiment we have considered 10 quasi-identifier attributes (4 categorical and 6 continuous), $\beta = 50$, $\mu = 100$, $\delta = 20000$, and both *UCI-Adult* and *SFU-Adult* data sets. The results are expected as a larger k implies that we need a larger cluster to anonymize data - this translates to greater loss in information. Moreover, it may require more merging to take place.

Effects of data distribution

We have conducted several experiments to evaluate how CASTLE scales with different data distributions. We have evaluated CASTLE on synthetic data sets following power law distribution generated by means of genzipf.[6] More precisely, power law distribution is generated using the following two formulas: $p(i) = \frac{c}{i^\alpha}, i = \{1, \ldots, N\}$ and $\sum_{i=1}^{N} p(i) = 1$. Figure 7 reports the average information loss with different α values. The experiments have been conducted with β, μ and *k* fixed and by varying δ. More precisely, we have evaluated the behavior of CASTLE wrt data distribution with different *k* values (cfr. Figures 7(a),(b),(c)). From the result, it is clear that CASTLE is very effective for clustered data. This is promising as real data are typically clustered.

5.2 Comparative study

In this section we compare the performance of CASTLE wrt the approach presented in [1], hereafter called *dynamicGroup*. We performed the comparison wrt the average information loss and the number of k-anonymized tuples.

[6] www.csee.usf.edu/~christen.

dynamicGroup can only process continuous attributes. Thus, the experiment has been performed with the 6 continuous quasi-identifier attributes of the *UCI-Adult* data set. Moreover, *dynamicGroup* only outputs the anonymized data when the process is complete (i.e., after scanning the entire data set). For a fair comparison, CASTLE anonymizes the stream data up to its end as follows: after the last tuple from the stream is pushed to a cluster, CASTLE outputs all the clusters whose size is not less than k. Then CASTLE generalizes all the tuples which fall in k_s-anonymized clusters in Ω. Finally, CASTLE merges all the remaining non-anonymized tuples to form a cluster and outputs it. *dynamicGroup* uses historical data to build some clusters. We take the first n tuples in UCI-Adult as the historical data, and all the remaining tuples as the streaming data. We vary n from 2000 to 8000, and select the best one for *dynamicGroup*. The best n is 8000. Figure 8 reports how the average information loss of CASTLE and *dynamicGroup* vary with different δ values. We have considered several values of *k* (see Figures 8(a),(b), and (c)). It is important to note that *dynamicGroup* does not consider the $delta$ constraint. Thus, it can retain tuples till the end of the process, which obviously influences the information loss. For a fair comparison between CASTLE and *dynamicGroup* we must consider only the average information loss of CASTLE with a δ set to *infinity*. As shown in Figure 8, as δ increases, the information loss of CASTLE decreases. Moreover, when δ is 10000, CASTLE outperforms *dynamicGroup*. Figure 9 shows an additional advantage of CASTLE over *dynamicGroup*. We present the number of output tuples that are returned progressively over time. Here, delayed time represents the order of the most recently arrived tuple. CASTLE is able to output anonymized data progressively. In fact, when the last tuple (30,162) arrives, over 97% of all the tuples have already been output. However, in *dynamicGroup*, anonymized data are only output at the end. This is not acceptable in a data stream context.

6 Conclusions

In this paper we have presented CASTLE a cluster-based framework to k-anonymize data streams. Relevant features of CASTLE are the enforcement of δ constraints, its adaptability to data distributions and the use of a cluster reuse strategy that improves the performance without compromising security. Performance evaluation reported in this paper have shown that CASTLE is efficient and effective. We plan to extend this work along several directions. A first direction is related to the *t*-closeness principle [6]. Ensuring this principle for data streams arises challenging issues, mainly due to the fact that the data distribution of a stream is unknown a priori. Additionally, in the current paper we have investigated inference problems that could arise by tracing

a k-anonymized data stream. As an extension, we would like to investigate how the proposed solution should be extended to consider other forms of background knowledge. Finally, we will study optimization techniques to improve the efficiency of our system.

References

[1] C. C. Aggarwal and P. S. Yu. A condensation approach to privacy preserving data mining. In Proc. *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 Proc. *PODS*, pages 153–162, 2006.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD*, pages 439–450, 2000.

[4] R. J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, pages 217–228, 2005.

[5] J.W. Byun, Y. Sohn, E. Bertino, and N. Li. Secure Anonymization for Incremental Datasets. In Proc. *VLDB Workshop on Secure Data Management*, pages 48–63, 2006.

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

[7] B. C. M. Fung, K. Wang, and P. S. Yu. Top-down specialization for information and privacy preservation. In Proc. *ICDE*, pages 205–216, 2005.

[8] L. Golab and M. Ozsu. Issues in data stream management. In *SIGMOD Record*, 32(2):5C14, 2003.

[9] V. S. Iyengar. Transforming data to satisfy privacy constraints. In Proc. *KDD*, pages 279–288, 2002.

[10] S. Kim and W. Winkler. Masking microdata files. In Proc. of the *Section on Survey Research Methods*, pages 114–119, 1995.

[11] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In Proc. of *SIGMOD International Conference on Management of Data*, pages 49–60, 2005.

[12] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian Multidimensional K-Anonymity. In *ICDE*, page 25, 2006.

[13] A.Machanavajjhala, J. Gehrke, D. Kifer, M. Venkitasubramaniam. l-Diversity: Privacy Beyond k-Anonymity. In Proc. *ICDE*, page 24, 2006.

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

[15] Xiaokui Xiao, Yufei Tao. Anatomy: Simple and Effective Privacy Preservation. In Proc. *VLDB*, pages 139–150, 2006.

[16] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Transaction Knowledge Data Engineering (TKDE)*, 13(6):1010–1027, 2001.

[17] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. In *International Journal on Uncertainty, Fuzziness, and Knowledge-Based Systems*, pages 571–588, 2002.

[18] X. Xiao, Y. Tao. m-Invariance: Towards Privacy Preserving Re-publication of Dynamic Datasets. In Proc. *SIGMOD*, 2007.

A Proofs

Proof. of Theorem 1

In order to prove that for any possible $s \in \Pi_Q(S_{out})$, $|SP(s)| \geq k$, we need to consider when a new tuple is inserted into S_{out}, that is, when a new $s \in \Pi_Q(S_{out})$. CASTLE outputs a tuple in two possible ways: (1) the tuple is output together with all the other tuples in the cluster C to which it belongs to (line 6, $output\_cluster()$ procedure). In this case s is the generalization of C, and (2) a tuple is individually output with the generalization of a k_s-anonymized cluster KC according to the *reuse* strategy (line 8, $delta\_constraint()$ procedure). In this case, s is the generalization of KC. In case (1), CASTLE outputs all tuples of a cluster C with the same generalization, which implies that all tuples in C have the same values for Q, that is, s. Moreover, CASTLE outputs all tuples of a cluster C only if the cluster size is greater than or equal to *k*. This condition ensures that there are at least *k* distinct individuals to which tuples in C refer to. Thus, $|SP(s)| \geq k$. In case (2), CASTLE outputs a tuple t with the generalization associated with a k_s-anonymized cluster KC. Since KC is k_s-anonymized, this implies that at a given instant KC's size has been greater than or equal to k and all its tuples have been generalized with s. This condition ensures that there are at least *k* distinct individuals to which tuples in C refer to. Thus, $|SP(s)| \geq k$. □

Proof. of Theorem 2

For simplicity, let us consider a data stream S with a single quasi-identifier attribute q. Let us consider a tuple t k_s-anonymized with generalization of cluster C. In this case, an attacker could infer a set of possible value $\chi \subset values(t.q)$ for attribute q, only if there exists a set of tuples $T \subseteq S_{out} \setminus \{t\}$ k_s-anonymized by a cluster C that *overlaps* C'. We say that two clusters C and C' overlap if there exists at least a quasi identifier attribute q_j in Q such that the generalization of q_j in C overlaps the generalization of q_j in C', that is: if q_j is a continuous attribute the intersection of generalization of q_j in C and the generalization of q_j in C' is not empty; if q_j is a categorical attribute and DGH_j is the corresponding domain generalization hierarchy, the subtree in DGH_j rooted at the generalization of q_j in C contains the subtree in DGH_j rooted at the generalization of q_j in C' or vice versa. Let us assume that an attacker infers a set of possible value $\chi \subset values(t.q)$ for attribute q, and prove that a contradiction arises. Let us consider a tuple $\tilde{t}$ in $S_{out} \setminus \{t\}$ whose q's generalization overlaps $t.q$'s generalization. Let us first consider the case that q is a continuous attribute. χ can be inferred by analyzing the overlap between the generalization of $\tilde{t}$ and that of t, and can be formalized as follows: $\chi = values(t.q) \setminus values(\tilde{t}.q)$. Thus, χ consists of the values in $values(t.q)$ which are

not in $values(\tilde{t}.q)$. However, if χ is not empty, it means that there exists an intersection between $values(t.q)$ and $values(\tilde{t}.q)$. This implies that CASTLE anonymizes the real value of $t.q_i$ by randomly selecting between $\tilde{t}.q$ and $t.q$ generalization (see line 7, $delta\_constraint()$ procedure). Thus, the possible real values of $t.q$ are, in addition to those in χ also those in $values(\tilde{t}.q)$. Thus, χ= ($values(t.q) \setminus values(\tilde{t}.q)) \cup (values(t.q) \cap values(\tilde{t}.q)$), which implies that χ = $values(t.q)$, thus a contradiction arises. We omit the proof when q is a categorical attribute, since it is very similar to the proof given for continuous attributes. □