CASTLE: Continuously Anonymizing Data Streams

Jianneng Cao, Barbara Carminati, *Member*, *IEEE*,
Elena Ferrari, *Senior Member*, *IEEE*, and Kian-Lee Tan

Abstract—Most of the 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 incoming data and the corresponding anonymized output. To cope with these requirements, in this paper, we present *Continuously Anonymizing STreaming data via adaptive cLustEring* (*CASTLE*), 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 w.r.t. the quality of the output data.

Index Terms—Data stream, privacy-preserving data mining, anonymity.

1 Introduction

Data streams are common to many application environments, such as, telecommunication, market-basket analysis, network monitoring, and sensor networks. Mining these continuous data streams [12], [13], [14] helps companies (the owner of data streams) to learn the behavior of their customers, thus, bringing unique opportunities. Many companies do not have the in-house expertise of data mining, so it is beneficial to outsource the mining to a professional third party [27]. However, data streams may contain much private information that must be carefully protected. 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 that the sales transaction stream has the schema $S(tid, cid, goods)$, where tid is transaction identifier, cid is customer identifier, and $goods$ are a series of items bought by the corresponding customer. Suppose that a relation C containing the information about Amazon customers is stored on disk, with schema $C(cid, name, sex, age, zipcode, address, telephone)$. Let SC be the stream generated by joining S with C on cid. Suppose moreover that, to analyze customers' buying behavior (e.g., building a decision tree), the mining is on SC,[1] and Amazon.com outsources it to a professional third party. To protect customers' privacy, attributes that explicitly identify customers (such as $name$, $address$, and $telephone$) are projected out of SC. However, the remaining data in SC may still be vulnerable to *linking attacks*: some attributes (e.g., sex, age, and $zipcode$) can be exploited to reidentify individuals by linking or matching them to external public databases (e.g., a voter registration table). Therefore, the streaming transactions in SC need to be carefully anonymized before they are passed to the third party.

A well-known technique to anonymize data is k-anonymity [25]. A data set T satisfies the k-anonymity property with respect to attribute set QI, if each combination of values of QI in T occurs at least k times; here, the attributes in QI are termed as *quasi-identifier* attributes and can be used to link with external information. A well-known technique to achieve k-anonymity, exploited by many of the methods proposed so far (e.g., [8], [18], [21], [22], [28]), is generalization. Generalization implies that a quasi-identifier attribute value is replaced by a less specific but semantically related value. For instance, a value in the Sex domain $\{male, female\}$ can be generalized to $person$.

However, traditional k-anonymity schemes (see Section 6 for a survey) are not suitable for streaming data. The main reason is that these methods cannot be directly applied on streaming data, because they are designed for static data sets. First, these techniques typically assume 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. Second, data streams have a *temporal dimension*, since they 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 application, 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.

1. In real stream systems, typically customer information does not appear in the stream to reduce redundancy. Mining, which needs customer information, requires joining the data stream with local customer databases. In what follows, we consider mining and anonymization on joint streams.

- *J. Cao and K.-L. Tan are with the National University of Singapore, School of Computing, 13 Computing Drive, Singapore 117417, Republic of Singapore. E-mail: {caojianneng, tankl}@comp.nus.edu.sg.*
- *B. Carminati and E. Ferrari are with the University of Insubria, DICOM, via mazzini, 5 22100 Varese, Italy. E-mail: {barbara.carminati, elena.ferrari}@uninsubria.it.*

Manuscript received 5 Dec. 2007; revised 20 Apr. 2009; accepted 7 Oct. 2009; published online 3 Dec. 2009.
Recommended for acceptance by E. Bertino.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2007-12-0184.
Digital Object Identifier no. 10.1109/TDSC.2009.47.

As we will further explain in Section 2 also the naive solution of joining S with a k-anonymized version of the relation C is not adequate, mainly because 1) the dimensions to be anonymized (i.e., quasi-identifier attributes) may come from both C and S; 2) linking attacks are still possible due to the possible inferences that can be performed by inspecting the sequence of anonymized tuples given in output.

Recently, the problem of anonymizing dynamic data sets, i.e., data sets where new tuples are inserted as well existing ones are deleted, has started to be investigated [9], [26], [29], [30]. Anonymizing data streams and anonymizing dynamic data sets have some similarities, in that they both suffer from potential inferences due to dynamic updates. However, as it will be discussed in Section 6, the inferences that may arise when anonymizing dynamic data sets are totally different from those that can take place during anonymization of data streams, that is, they are possible under different assumptions. This makes the methods proposed for secure anonymization of dynamic data sets not suitable for data streams.

To cope with all the above-discussed requirements, in this paper, we present *Continuously Anonymizing STreaming data via adaptive cLustEring* (*CASTLE*), a cluster-based scheme that k-anonymizes streams on-the-fly and, at the same time, ensures the freshness (i.e., the maximum delay between the arrival of a tuple and its release to the third party) of anonymized data by satisfying specified delay constraints. Moreover, we propose an extension of CASTLE to support ℓ-diversity [24] on data streams. To the best of our knowledge, this is the first reported work that considers k-anonymity and ℓ-diversity on data streams. However, note that relevant clustering schemes [4], [17] for data streams have been proposed. They focus on finding φ centers in the streaming data so that the sum of distances from data points to their closest centers is minimized. As a consequence, the principle regulating clustering generation is to minimize the distance while the number of total clusters is at most φ. In contrast, in order to k-anonymize streaming data, the proposed approach has to follow another principle (i.e., the number of tuples in each cluster has to be at least k), which requires to devise new clustering algorithms.

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 into clusters and releases all tuples belonging to the same cluster with the same generalization. CASTLE supports the anonymization of both numerical and categorical attributes, 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 parameter δ. We refer to this constraint as *delay 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 all its tuples have been given in output, 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 that exploit the sequence of anonymized tuples returned as output. In the paper, we present the reuse strategy employed by CASTLE to avoid such privacy breaches.

The remainder of this paper is organized as follows: Section 2 defines k-anonymity and delay constraints for data streams. In Section 3, we introduce the basic ideas underlying CASTLE (refer to our poster paper [11] for the basic ideas). Section 4 presents detailed CASTLE algorithms and formal results about their security. Section 5 reports experimental results on the effectiveness and efficiency of CASTLE. Section 6 reviews related work, whereas Section 7 concludes the paper. Due to lack of space, we move to the supplementary material section proofs of the main formal results presented in the paper, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2009.47, the complexity analysis of CASTLE algorithms, and additional experiments w.r.t. those reported in Section 5.

2 Anonymizing Data Streams

As discussed in Section 1, the traditional k-anonymity definition and the related algorithms are not adequate for data streams. Let us see in more details why. Consider once again the Amazon.com example. A naive solution to achieve stream anonymity is to apply standard k-anonymization techniques to relation C and then joining the anonymized version with stream S. As remarked in Section 1, the first problem with this solution is that the 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.

Further, applying the naive solution to data streams results in what has been called *weak* k*-anonymity* [7]. Weak k-anonymity does not require that each equivalence class[2] has a size of at least k. It only requires that the QI value of each released tuple can be linked to at least k individuals in a public table. It is proven that once k-anonymity is achieved, weak k-anonymity is automatically satisfied. However, the reverse is not true. Therefore, weak k-anonymity is less secure than k-anonymity. Furthermore, weak k-anonymity provides privacy only under the assumption that an adversary does not know a priori whether an individual is in the released data or not, which appears too strong for most of the application environments related to data streams.

As an example, assume Table 1 is a portion of the customer table, in which quasi-identifier attributes are *Sex*, *Zipcode*, and *Age*. Table 2 is a 3-anonymized version of Table 1, where CID and Name are put only for row referencing. Suppose that Bob has bought something from a store and there is an anonymized record in the output stream published by this store, i.e., $\bar{t}(M, [53703\text{-}53708], [26\text{-}31], sex\ video)$. In addition, no other record with the same generalized quasi-identifier value is found. Suppose that Beth is an attacker and that she

2. The set of released tuples which have the same generalized QI value is called equivalence class.

TABLE 1
Customer Table

CID	Name	Sex	Zipcode	Age
01	Mike	M	53708	31
02	Alice	F	53715	21
03	John	M	53703	28
04	Bob	M	53706	26
05	Beth	F	53703	24
06	Carol	F	53706	22

TABLE 2
Three-Anonymized Customer Table

CID	Name	Sex	Zipcode	Age
01	Mike	M	[53703-53708]	[26-31]
03	John	M	[53703-53708]	[26-31]
04	Bob	M	[53703-53708]	[26-31]
02	Alice	F	[53703-53715]	[21-24]
05	Beth	F	[53703-53715]	[21-24]
06	Carol	F	[53703-53715]	[21-24]

knows the detailed information of Bob's quasi identifier. If Beth does not know that Bob appears in the stream (the fact that Bob has made a purchase), by joining $\bar{t}$ with Table 2 she will find that $\bar{t}$ is linkable to three persons: Mike, John, and Bob. This is *weak 3-anonymity*. The identity of the buyer of *sex video* is protected. However, once Beth knows that Bob has made a purchase, she is sure that $\bar{t}$ refers to Bob, not to John or Mike, and knows that Bob bought sex video. Bob's privacy is therefore violated.

In addition, applying the naive solution requires a further anonymization step if we want to support ℓ-diversity [24]. ℓ-Diversity requires that the tuples with the same generalized value should have at least ℓ distinct values for each different sensitive attribute. Since 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.

Furthermore, k-anonymity requires that in the released data one person's information is indistinguishable to at least $k-1$ other persons' information w.r.t. QI. Traditional k-anonymity schemes simplify the problem by assuming that an individual only has one record for publishing. Under this assumption, once an equivalence class contains k tuples, it will refer to k persons and the information of these k persons will be indistinguishable w.r.t. QI. However, this assumption is not realistic for streaming data and incurs the following attack. Consider that the QI size is big enough and the QI values of two different persons are different with a high probability. Consider again the example of Amazon.com and assume that one customer has purchased $n \geq k$ items. After joining these transactions with customer relation C, n tuples with the same QI value will appear in the resultant stream. By traditional k-anonymization schemes these n tuples can be output immediately without any generalization of their QI values. However, the precise QI value can be linked to a distinct person in the voting list with a high probability. Thus, the customer is reidentified and his/her privacy is violated.

Therefore, we strongly believe that solutions specifically conceived to take into account the characteristics of data streams are needed. In the following, 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. Then, we present the information loss metrics used throughout the paper.

2.1 k-Anonymity of Data Streams

In the following, we model a data stream 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.

Definition 2.1 (k-Anonymity of Data Streams). *Let $S(p, pid, a_1, \ldots, a_j, q_1, \ldots, q_n)$ be a stream, where $\{q_1, \ldots, q_n\}$ are quasi-identifier attributes, pid is the person's identity, p is the 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. We say that S_{out} is* k-anonymized, *if both the following conditions hold:*

- *For each tuple $t \in S$, there exists in S_{out} the corresponding anonymized tuple $\bar{t}$.*
- *Given a tuple $\bar{t} \in S_{out}$, we define qg as the corresponding* QI group, *where $qg = \{\bar{t}' \in S_{out} | \bar{t}'.q_j = \bar{t}.q_j, j \in [1, n]\}$. Given a QI group qg, let $DP(qg)$ be the set of distinct persons which tuples in qg refer to. For each possible distinct $qg \subset S_{out}$, $|DP(qg)| \geq k$.*

In the following, we denote k-anonymity of data streams by k_s-anonymity to distinguish it from k-anonymity for static data sets. 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 2.2 (Delay Constraint). *Let X be a k_s-anonymization scheme that takes as input a data stream S and generates in output a data stream S_{out}, and let δ be a positive integer. We say that X satisfies the* delay constraint *δ if and only if for each new tuple $t \in S$ with position $t.p$, all tuples with position less than $t.p - \delta$ have already been output by X.*

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 needs to be output. Note that the δ parameter can be tuned on the basis of the application domain, the temporal requirements, and the required information quality. Indeed, when δ increases, the maximum delay between the arrival of a tuple and its release to the third party for data mining is increased. However, this allows CASTLE to buffer more tuples, and the defined metric space (each QI attribute is one dimension) becomes denser. So, CASTLE is more likely to group similar tuples together and reduces information loss. Therefore, it is possible to trade off between the allowed delay and the obtained information quality. The experiments in Section 5

verify this relationship between the allowed delay and information loss.

In Section 1, we have discussed the prior knowledge (capability) of an attacker. We formalize it as follows:

Definition 2.3 (Adversary Model). *At any instant $\imath$, an attacker's prior knowledge includes:*

- *The deployed generalization principle (that is, the algorithms according to which tuples are generalized by CASTLE).*
- *The values of all the released tuples in the output stream at instant $\imath$, and those of all the previously released tuples.*
- *The QI value of any victim, and whether a tuple related to the victim appears in the output stream.*

2.2 Information Loss Metrics

As we are dealing with streaming data, we need an information loss metric that can be calculated in an incremental manner. Various information loss metrics have been proposed. The *Discernibility Metric* (DM) [8] measures the size of equivalence classes, but it does not consider the distribution of tuples in the defined metric space (e.g., a small-size equivalence class may have big minimum bounding box). The *Classification Metric* (CM) [19] is a good candidate when the use for anonymized data is to build classification models. But the extension of CM to general purpose applications is not clear. *Generalized Loss Metric* (GLM) [19] is more appropriate, because it considers both the size of equivalence class and the distribution of tuples. It captures the general notion of information loss. Anonymized data by GLM can be disseminated for multiple purposes (see references [19], [32] for a discussion). For those reasons, we adapt GLM to streaming data as follows: Let $\{q_1, \ldots, q_n\}$ be the 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.

In contrast, given a continuous attribute q_i and an interval $I = [l, u]$ from the domain $[L, U]$ of q_i, used to generalize q_i'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 $g = (v_1, \ldots, v_n)$ as follows:

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

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

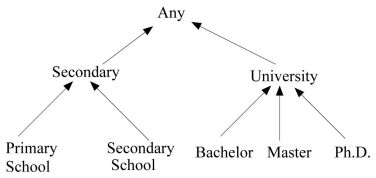

Fig. 1. Domain generalization hierarchy of Edu.

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

3 The CASTLE Framework

In the following section, before presenting and discussing CASTLE in more details, we introduce the notion of k_s-anonymized cluster.

3.1 k_s-Anonymized Clusters

The basic idea of the proposed approach is to exploit quasi-identifier attributes to define a metric space, such that tuples can be considered points in this space. According to this strategy, clusters can be defined as n-dimensional intervals, where n is the number of quasi-identifier attributes. The formal definition of cluster is given below.

Definition 3.1 (Cluster over a Data Stream). *Let $S(p, pid, a_1, \ldots, a_j, q_1, \ldots, q_n)$ be a stream where $\{q_1, \ldots, q_n\}$ are the quasi-identifier attributes. Let $S' \subset S$ be a set of tuples. A cluster C over S' is defined as a set of intervals, called* range intervals*, in the quasi-identifier attribute domains. For each quasi-identifier attribute q_i, 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 q_i's values of tuples in S'.*
- *If q_i is a categorical attribute, let $\text{Leaves}(DGH_i)$ be the set of leaves of DGH_i generated by a leftmost 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 leftmost and the rightmost values of L_{q_i}, respectively.*

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

Example 1. Consider the stream Customer($p, pid, a_1, \ldots, a_j, Age, Edu$), where Age and Edu are the quasi-identifier attributes, and the DGH_{Edu} is presented in Fig. 1. Let us consider the following tuples:[3] (pid_1, 25, Bachelor), (pid_2, 26, Master), and (pid_3, 30, Ph.D.). According to Definition 3.1, 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

3. For simplicity, here and in the following, we only consider the pid and QI attributes. Moreover, where not relevant, pid attribute is omitted.

to three. If we further add the tuple (pid_1, 25, Bachelor) to C, it does not change its range intervals nor its size (which is still equal to three).

Once a cluster C reaches the size of at least k, it implies that there exist at least k distinct individuals which tuples in C refer to. Moreover, since all tuples are contained in the same cluster, each quasi-identifier attribute of all of them is enclosed into the same range interval. Intuitively, if all these tuples are given in output by generalizing the value of each quasi-identifier attribute in the same way, we satisfy properties of Definition 2.1. Thus, according to our approach, the way by which quasi-identifier attributes are given in output is determined on the basis of cluster range intervals and formally defined as follows:

Definition 3.2 (Cluster Generalization). *Let $C(r_1, \ldots, r_n)$ be a cluster. The corresponding cluster generalization (or simply generalization), denoted as $G = (g_1, \ldots, g_n)$, is defined such that, for each $r_i, i \in [1, n]$, g_i is computed 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 w.r.t. 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 g_i in the generalization associated with C.

The information loss of a cluster C, denoted in what follows as *InfoLoss*(C), is the information loss of the corresponding generalization (see Section 2.2). We can now state when a cluster is k_s-anonymized.

Definition 3.3 (k_s-Anonymized Cluster). *Let $C(r_1, \ldots, r_n)$ be a cluster, and $(g_1, \ldots, g_n)$ be the corresponding generalization. If at a given time instant $\imath$, $C.size$ is greater than or equal to k and all tuples in C are output with C's generalization $(g_1, \ldots, g_n)$, we say that, starting from $\imath$, C is a k_s-anonymized cluster.*

3.2 CASTLE

In this section, we present CASTLE. We start by giving a general overview of the underlying approach and by showing how it adapts to the data stream distribution. Detailed algorithms are presented in Section 4.

3.2.1 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, CASTLE selects, among all the existing clusters, the one to which t can be assigned, that is, the one whose range intervals enclose t's attribute values. However, it could be the case that no clusters can contain the new tuple, that is, the values of quasi-identifier attributes of t are not contained into the range intervals of any cluster.

When a new tuple cannot be assigned to any existing cluster, there is the need to enlarge one of them in order to accommodate the new tuple. Cluster enlargement implies the enlargement of its range intervals and, as a consequence, an increase of the information loss. To minimize information loss, when selecting the cluster where a new tuple is pushed, CASTLE chooses the one that requires the smallest enlargement. Cluster enlargement is formally defined as follows:

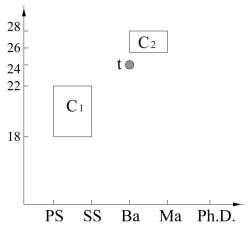

Fig. 2. Cluster selection.

Definition 3.4 (Cluster Enlargement). *Let $S(p, pid, a_1, \ldots, a_j, q_1, \ldots, q_n)$ be a stream, where $\{q_1, \ldots, q_n\}$ are the quasi-identifier attributes. Let $C(r_1, r_2, \ldots, r_n)$ be a cluster defined over tuples in $S' \subset S$, and let $(g_1, g_2, \ldots, g_n)$ be the associated generalization. Let t be a tuple in $S \setminus S'$. The enlargement of C w.r.t. t is defined as follows:*[4]

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

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

Example 2. Consider clusters C_1 and C_2 in Fig. 2. To enclose tuple $t =$ (24, Bachelor) 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). Assume that [18, 120] is the domain of the Age attribute. Thus, the enlargement of C_1 due to absorbing t is

$$Enlargement(C_1, t) = 1/2 * (6/102 + 4/4) - 1/2 * (4/102 + 1/4) = 0.384.$$

In contrast, the range intervals of C_2, enlarged to enclose t, are [24, 28] and [Bachelor, Master], respectively, which correspond to the new generalization associated with C_2: ([24, 28], University). This implies that the enlargement of C_2 due to absorbing t is $Enlargement(C_2, t) = 1/2 * (4/102 + 2/4) - 1/2 * (2/102 + 2/4) = 0.01$. Thus, t is pushed into C_2.

Therefore, a tuple is pushed into the cluster which requires the smallest enlargement to include the tuple. However, 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 3.2.3 for more details).

To satisfy delay constraints, when a new tuple arrives, CASTLE checks whether a tuple in some cluster is going to expire. In such a case, the corresponding tuple must be immediately output. Here, there are two main cases. The first is when cluster C, hosting the expiring tuple, has already a size greater than or equal to k. In this case,

4. In the case C contains only one tuple $\bar{t}$, we say that $Enlargement(C, t)$ returns the distance between $\bar{t}$ and t.

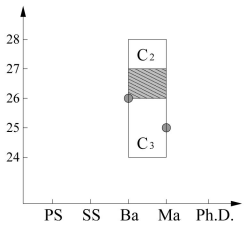

Fig. 3. Overlapping clusters.

CASTLE simply outputs all the tuples in C with its generalization, and, starting from that instant, it considers C as a k_s-anonymized cluster. The second case is when the cluster C hosting the expiring tuple has size less than k. To immediately output the expiring tuple, CASTLE merges C with some of its neighboring clusters such that the size of the resultant cluster is greater than or equal to k. More precisely, CASTLE selects those that result in minimum enlargement to C. Then, all the tuples contained in the new cluster can be output with its generalization (see Section 4 for more details on the merge operation).

In both cases, before outputting the cluster's tuples, CASTLE verifies whether the cluster can be split into smaller subclusters. Indeed, 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 $2k$, before outputting the tuples, CASTLE splits it into two or more subclusters, each with size at least k (see Section 4 for more details on the split operation).

3.2.2 Reuse of k_s-Anonymized Clusters

To increase the information quality of anonymized data, we have enhanced CASTLE with a strategy that enables to *reuse* k_s-anonymized clusters (i.e., their generalizations). According to this approach, after a cluster becomes k_s-anonymized, it is not deleted from memory. Instead, its generalization is kept, and is used later on to output expiring tuples contained in it.

A k_s-anonymized cluster will be no more k_s-anonymized if its range intervals are enlarged due to the insertion of new tuples. Therefore, when a new tuple t arrives, CASTLE selects the cluster to absorb t only among the set of *non*-k_s-anonymized clusters. This not only avoids the enlargement of k_s-anonymized clusters, but also gives to the *non*-k_s-anonymized clusters more possibilities to become k_s-anonymized. It is relevant to note that this strategy leads to overlaps between k_s-anonymized and *non*-k_s-anonymized clusters. The main advantage of overlaps is that the generalizations of k_s-anonymized clusters can be used for anonymizing expiring tuples that have been absorbed by *non*-k_s-anonymized clusters but are also enclosed into a k_s-anonymized cluster. Therefore, every time a tuple t inside a *non*-k_s-anonymized cluster is going to expire, CASTLE verifies whether t also falls in a k_s-anonymized cluster KC. If this happens, t is immediately output with KC's generalization. Thus, the reuse strategy avoids some cluster merges and, therefore, improves information quality.

Example 3. Let us suppose that, at a given instant, cluster C_2([26, 28], [Bachelor, Master]) becomes k_s-anonymized. Moreover, suppose that, after some time, a new cluster C_3([24, 27], [Bachelor, Master]) is generated (see Fig. 3). In case a tuple $t =$ (26, Bachelor) of C_3 is going to expire and C_3 still has size less than k, t can be given in output with C_2's generalization, that is, ([26, 28], University).

When a tuple t is expiring, the best way to preserve information is to select from all the k_s-anonymized clusters the one which contains t and has the minimal information loss. However, this method enables an attacker to infer additional knowledge about a tuple's value or even guess its exact value, as the following example shows:

Example 4. With reference to Fig. 3, suppose that at a given instant $\imath$, both clusters C_2([26, 28], [Bachelor, Master]) and C_3([24, 27], [Bachelor, Master]) are k_s-anonymized. Suppose, moreover, that a tuple $\tilde{t} =$ (25, Master) arrives after instant $\imath$ and that, after some time, it is expiring. 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 $\tilde{t}.Age$ is not [26, 27] (the age overlap between C_2 and C_3), otherwise, t would be in C_2 and would have been output with C_2's generalization, which preserves more information comparing with C_3's generalization. Thus, an attacker can infer that $\tilde{t}.Age$ belongs to $C_3.Age \setminus (C_2.Age \cap C_3.Age) = [24, 25]$.

To overcome this attack, we adopt the following reuse strategy: if an expiring tuple falls into the overlap of two or more k_s-anonymized clusters, CASTLE randomly selects one of them and anonymizes the tuple with its generalization. This avoids the security flaw previously discussed (see Section 4.3 for a formal proof).

Example 5. Consider again Example 4 and assume that the strategy discussed above is adopted. When an attacker sees the generalization of $\tilde{t}$ (the generalization of C_3), s/he knows that $\tilde{t}$ could be in $C_2 \cap C_3$ (the overlap between C_2 and C_3) or $C_3 \setminus (C_2 \cap C_3)$. This inference tells that $\tilde{t}$ could be in any place of $(C_2 \cap C_3) \cup (C_3 \setminus (C_2 \cap C_3))$, which is exactly the generalization of C_3.

3.2.3 Adaptability to Data Stream Distribution

In order to adapt CASTLE to data stream distribution we adopt several strategies. The first is related to the value of τ, that is, the threshold used 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 (see Section 5 for a discussion on how to set μ). 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, which increases the chance of a new tuple to be pushed into an existing cluster, 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 $non\text{-}k_s$-anonymized clusters that can be in memory. This number is limited by a given parameter β. This parameter cooperates with τ to handle the variation of data distribution. More precisely, suppose that, at the beginning, all tuples are well clustered, so τ has a small value. If the newly arriving tuples are sparsely 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. For this reason, we adopt β to prevent the generation of a possibly large number of clusters and thus limit the overhead. β can be set by taking into account the available computational and storage resources. Therefore, a new cluster is generated only if the number of $non\text{-}k_s$-anonymized clusters is fewer than β. Otherwise, a tuple is pushed to the existing cluster, which requires the minimum enlargement to enclose it.

As a further strategy to improve data quality of anonymized tuples, CASTLE manages *outliers*, that is, tuples that are faraway from other tuples. Indeed, the presence of few outlier tuples can drastically increase the information loss of the k_s-anonymized stream. Due to lack of space, the discussion on the management of outliers is provided in the supplementary material section, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2009.47.

4 CASTLE Algorithms and Analysis

In the following, we present the algorithms implementing the techniques illustrated in the previous section. Then, we illustrate their extension to achieve ℓ-diversity. Moreover, we analyze their security.

4.1 Algorithms

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, 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         delay_constraint(t');
```

The algorithm takes as input the stream S to be anonymized, and the parameters k, δ, and β. At the beginning, the set of $non\text{-}k_s$-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 function $best\_selection()$ (step 6), to select from Γ the best cluster into which t is pushed (step 10). If such a cluster does not exist, Algorithm 1 creates a new cluster for t (steps 7 and 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 $delay\_constraint()$ (step 13). In the following, we illustrate the procedures/functions used by Algorithm 1.

Function $best\_selection()$. We recall that according to the reuse strategy, a new tuple is always pushed into a $non\text{-}k_s$-anonymized cluster, by selecting one among those which require the minimum enlargement. Thus, to find out the best $non\text{-}k_s$-anonymized cluster where inserting the new tuple t, $best\_selection()$ 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 requiring the minimum enlargement (steps 5 and 6). According to the adaptability strategy described in Section 3.2.3, $best\_selection()$ selects from the returned clusters only those whose information loss is less than or equal to τ (steps 7-10). Among those, $best\_selection()$ chooses the cluster with the minimum size (step 17). By contrast, if no clusters have information loss less or equal to τ, it implies that a new cluster should be created over t. However, it is necessary to verify whether the constraint on the maximum number of $non\text{-}k_s$-anonymized clusters is satisfied. Thus, there could be two different cases. The first case is when the number of $non\text{-}k_s$-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, i.e., $SetC_{min}$, it is returned the one with the minimum size (step 13). Otherwise, the function returns a NULL value (step 15), which triggers in Algorithm 1 the generation of a new cluster over t.

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 in E;
6  Let SetC_min be the set of clusters C̃ in Γ with Enlargement(C̃, t) = 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;
```

Procedure $delay\_constraint()$. When a tuple t is expiring, Algorithm 1 calls procedure $delay\_constraint()$, whose main goal is to output t. According to the proposed approach, this can be achieved in several ways. First, $delay\_constraint()$ verifies whether the expiring tuple can be output with the generalization of the $non\text{-}k_s$-anonymized hosting cluster (steps 1-3). The procedure first verifies whether it has size greater than or equal to k. In this case, $delay\_constraint()$ calls procedure $output\_cluster()$ (step 3). Otherwise, $delay\_constraint()$ verifies whether it is possible to apply on the expiring tuple the reuse strategy illustrated in Section 3.2.1 (steps 5-9). The reuse strategy can be applied if there exist one or more k_s-anonymized clusters, i.e., KC_{set} is not empty, containing the expiring tuple. If this is the case, to overcome the inference problem described in

Section 3.2.2, the *delay_constraint*() procedure randomly selects a cluster in KC_{set} and gives in output the tuple with its generalization.

Procedure `delay_constraint(t)`

```
1  Let C be the non-k_s-anonymized cluster to which t belongs;
2  if C.size ≥ 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 KC be a cluster randomly selected from KC_set;
8          Output t with the generalization of KC;
9          Return;
10     Let m be an integer set to 0;
11     foreach C_j ∈ Γ do
12         if C.size < C_j.size then
13             m = m + 1;
14     if m > |Γ|/2 then
15         Suppress tuple t;
16         Return;
17     if Σ_{C_i∈Γ} C_i.size < k then
18         Suppress t;
19         Return;
20     MC =merge_clusters(C, Γ \ C);
21     output_cluster(MC);
```

If the reuse strategy cannot be applied, i.e., KC_{set} is empty, the *delay_constraint*() procedure verifies whether t is contained into an outlier cluster, so to suppress it (steps 10-16). As discussed in the supplementary material section, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2009.47, to judge an *expiring cluster* (i.e., a cluster containing expiring tuples) as an outlier, CASTLE verifies whether it is smaller than the majority of the other clusters in size. More precisely, the *delay_constraint*() procedure scans all *non-*k_s-anonymized clusters, by checking their sizes. If the expiring cluster is smaller than $\frac{|\Gamma|}{2}$ existing *non-*k_s-anonymized clusters in size, it is regarded as an outlier. In this case, CASTLE suppresses t, that is, it outputs t with the most generalized QI value.

As the last alternative, procedure *delay_constraint*() verifies whether a merge among C and some of 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 17), a merge operation would not generate a cluster with the size at least k. Therefore, the only way to output the expiring tuple is suppressing it (step 18). Otherwise, the merge can take place (step 20), and after the merge, the merged cluster is output (step 21).

The *merge_clusters*() function receives as input the cluster to be merged, i.e., C, and the set of *non-*k_s-anonymized clusters excluding C itself. The procedure, for every *non-*k_s-anonymized cluster C_i, 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. Then, the resulting cluster is given in output.

Procedure *output_cluster*(). When outputting a cluster C, in order to minimize information loss, procedure *output_cluster*() verifies if C can be split, i.e., whether its size is at least $2k$ (step 1). If this is the case, it calls function *split*() that splits C into subclusters, each with size at least k. Then, all tuples of the newly created clusters (or of C, respectively) are given in output with the generalization of the corresponding cluster (step 6). Moreover, τ is updated to be the average information loss of the most recent k_s-anonymized clusters including the 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 C_i into Ω only if its information loss is less than τ (steps 8-11). In addition, C_i is deleted from Γ (step 12).

Procedure `output_cluster(C)`

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

Function *split*(). The splitting technique exploited in CASTLE is based on the KNN algorithm proposed in [5], which we adapt to the definition of k_s-anonymity. In general, the *split()* function randomly selects a tuple t from a cluster C and creates over it a new cluster C_{new}. Then, it populates the new cluster C_{new} with other tuples of C, obviously t excluded, until C_{new} has size equal to k. This process of creating and populating a subcluster is repeated until C's size is less than k. Then, the remaining tuples of C are accommodated into the new generated subclusters.

Function `split(C)`

```
1  Initialize SC to be empty;
2  Let BS be the set of buckets created by grouping tuples in C by pid attribute;
3  while |BS| ≥ k do
4      Randomly select a bucket B̄ from BS, and pick one of its tuples t̄;
5      Create a new sub-cluster C_new over t̄;
6      if B̄ is empty then
7          delete B̄;
8      Let H_{k-1} be a heap with k − 1 nodes, each with an infinite distance to t̄;
9      foreach bucket in BS \ B̄ do
10         Pick one of its tuples t, and calculate t's distance to t̄;
11         if t is closer to t̄ than the root of H_{k-1} then
12             t replaces the root, and H_{k-1} is adjusted accordingly;
13     foreach node in H_{k-1} do
14         Let t̃ be the tuple in the node;
15         Insert t̃ into C_new;
16         Let B_j be the bucket containing t̃;
17         Delete t̃ from B_j;
18         if B_j is empty then
19             delete B_j;
20     Add C_new to SC;
21 foreach B_i ∈ BS do
22     pick a tuple t_i in B_i;
23     find t_i's nearest cluster in SC, and add all the tuples in B_i to it;
24     delete B_i;
25 return SC;
```

The definition of k_s-anonymity (cf. Definition 2.1) requires that, for each possible QI group, there exist at least k distinct individuals referring to its tuples. To be compliant with this definition, clusters generated by function *split()* must contain tuples referring to at least k distinct individuals. Therefore, in selecting a tuple of C to be pushed into C_{new}, the *split()* function considers only those tuples having *pid* different from *t.pid*. Among these, it selects the ones that incur the minimum enlargement to C_{new}, so that the information loss is kept low.

Let us see in more details how $split()$ works. First, the function groups all tuples in C by their pid values, creating a set of buckets BS (step 2), each one containing only tuples with the same value for attribute pid. Then, it randomly selects a tuple $\bar{t}$, i.e., it randomly selects a bucket $\bar{B}$ and picks one of its tuples (step 4) and creates a new cluster over it (step 5). To choose the $k-1$ tuples nearest to $\bar{t}$ to be pushed into the new generated subcluster, $split()$ sorts the buckets according to their distance to $\bar{t}$. Note that all tuples in a bucket have the same distance to $\bar{t}$, because they all have the same pid and the same quasi-identifier attribute values. To speedup this step, we exploit a heap structure of $k-1$ nodes, where each one contains a tuple belonging to a different bucket in BS. The heap is generated in lines 8-12, by updating its nodes based on their distances to $\bar{t}$ (step 10).

Then, $split()$ creates the new subcluster C_{new} by inserting into it the tuples stored into the heap nodes (steps 13-15). This ensures that the new generated cluster C_{new} contains k tuples of k different buckets, that is, k tuples refer to k distinct individuals, so that the k_s-anonymity definition is satisfied. The above steps are repeated while the number of buckets are greater than or equal to k, which implies that there exist at least k further tuples referring to k distinct individuals; thus, another subcluster can be generated (step 3). Finally, each of the remaining tuples in buckets in BS is pushed into one of the new generated subclusters (steps 21-23). More precisely, a tuple is pushed into the cluster that requires the minimum enlargement to enclose it.

4.2 ℓ-Diversity

Recently, Machanavajjhala et al. [24] proposed the ℓ-diversity principle to avoid possible inference attacks on k-anonymized data. Ensuring ℓ-diversity requires that all tuples with the same generalization, i.e., all tuples belonging to the same QI group, have at least ℓ distinct values for the sensitive attribute. In this section, we illustrate how our scheme for k_s-anonymizing streaming data can be easily extended for k_s-anonymizing and ℓ-diversifying streaming data. In order to ensure that tuples with the same generalization, i.e., tuples belonging to the same cluster, have at least ℓ distinct values for the sensitive attribute, we need to modify the definition of k_s-anonymized cluster, to include the ℓ-diversity principle. In doing that, we consider a single sensitive attribute a_s. Moreover, given a cluster C, we denote with $C.diversity$ the number of distinct values of a_s for tuples in C.

Definition 4.1 (k_s-Anonymized and ℓ-Diversified Cluster). *Let $C(r_1, \ldots, r_n)$ be a cluster, and $(g_1, \ldots, g_n)$ be the corresponding generalization. If, at a given time instant $\imath$: 1) $C.size$ is greater than or equal to k, 2) $C.diversity$ is greater than or equal to ℓ, and 3) all tuples in C are output with C's generalization $(g_1, \ldots, g_n)$, we say that, starting from $\imath$, C is a k_s-anonymized and ℓ-diversified cluster.*

In order to ensure that tuples are given in output only if they belong to a k_s-anonymized and ℓ-diversified cluster, it is necessary to slightly modify CASTLE algorithms. We recall that in Algorithm 1 tuples are given in output only when procedure *delay_constraint()* is called. The procedure verifies whether one of the following cases is satisfied: 1) the expiring tuple belongs to a cluster that is ready to become anonymized; 2) the expiring tuple belongs to an outlier cluster; or 3) if neither of the previous conditions is satisfied, *delay_constraint()* verifies whether a merge is possible, that is, whether merging some of the existing clusters generates a new k_s-anonymized cluster. Thus, the new definition of k_s-anonymized and ℓ-diversified cluster, requires to modify cases 1 and 3. In particular, the latter one simply entails to update the instruction that verifies whether a merge operation is applicable (see line 17 of *delay_constraint()*). Besides the condition that $\sum_{C_i \in \Gamma} C_i.size < k$, it must also be checked that there exist at least ℓ distinct values of a_s among all clusters in Γ.

In contrast, to properly handle case 1, it is necessary to replace the condition according to which a cluster C is judged ready to be anonymized (see line 2 of *delay_constraint()*). Indeed, to take into account also ℓ-diversity, the conditions to be satisfied are now two: $C.size \geq k$ and $C.diversity \geq \ell$. It is relevant to notice that, when a cluster is ready to be anonymized, it is passed to the $output\_cluster()$ procedure that, before giving in output all its tuples, verifies whether a split is possible, that is, whether $C.size \geq 2k$. If this is the case, it calls $split()$, whose aim is to generate k_s-anonymized subclusters. However, due to the new definition of anonymized cluster (cf. Definition 4.1), the *split()* function needs to be redefined. Indeed, the ℓ-diversity principle requires to take into account during subclusters generation also the sensitive attribute a_s, in that subclusters must have size and diversity at least equal to k and ℓ, respectively. To satisfy both these conditions, we have designed a different $split()$ function (see $split^{\ell}()$ below) that, differently from the previous one, should be called only if the cluster C to be split has size and diversity at least equal to 2k and ℓ, respectively (this implies to update line 1 of $output\_cluster()$).

Function $split^{\ell}()$. Let us first introduce the basic idea of the new split function, by assuming that each tuple in the data stream refers to a distinct individual. We illustrate then how we can relax this assumption. Given a cluster C, where $C.size \geq 2k$ and $C.diversity \geq \ell$, the first step to generate a k_s-anonymized and ℓ-diversified subcluster is to group C's tuples into distinct buckets BS on the basis of the values of the sensitive attribute a_s. Then, a new subcluster C_{sub} is generated by randomly selecting from each bucket a tuple to be inserted into it. This is repeated until C_{sub} has absorbed k tuples. These k tuples refer to k distinct persons and guarantee that C_{sub} satisfies the constraint of k_s-anonymity. Note that since k is greater than or equal to ℓ, each bucket in BS participates with at least a tuple in the new subcluster. This ensures that the new created subcluster contains ℓ distinct values of the sensitive attribute, and thus it also satisfies the ℓ-diversity principle. However, relaxing the condition that each tuple in the data set is associated with a different person requires to slightly modify the basic strategy. Indeed, in this case, it may happen that two or more tuples selected from two distinct buckets have the same value for attribute pid. Thus, even if C_{sub} absorbs k tuples from different buckets in BS, the number of distinct pid values in it could still be less than k, that is to say, C_{sub} is still not k_s-anonymized. To overcome this, we consider only buckets without overlaps w.r.t. pid. This is done by selecting only one tuple from C for each distinct pid value, and grouping only these selected tuples into buckets according to their values of the sensitive attribute. Avoiding overlaps

ensures that the k tuples selected from buckets in BS have different pids; thus, the size of the new generated subcluster C_{sub} is equal to k. It is important to note that selecting only a subset of tuples of C does not mean that the other tuples are not considered. By contrast, when all possible subclusters are generated, each tuple t previously not selected from C is inserted into one of the new subclusters. In particular, a tuple t is inserted into the unique subcluster that contains a tuple with the same pid of t.

Function $split^{\ell}(C, a_s)$

1 Let BS be the set of disjoint buckets generated by $generate\_buckets(C, a_s)$;
2 **if** $|BS| < \ell$ **then**
3 return $\{C\}$;
4 Let SC be a set of sub-clusters, initialized empty;
5 **while** $|BS| \geq \ell$ and $sum = \sum_{B_i \in BS} B_i.size \geq k$ **do**
6 Randomly select a $\bar{B}$ from BS;
7 Randomly select a tuple $\bar{t}$ from $\bar{B}$;
8 Generate a sub-cluster C_{sub} over $\bar{t}$;
9 Delete $\bar{t}$ from $\bar{B}$;
10 **foreach** $B_j \in BS$ **do**
11 **foreach** *tuple* $t_i \in B_j$ **do**
12 Let e_i be $Enlargement(C_{sub}, t_i)$;
13 Sort tuples of B_j by ascending order of their enlargement e_i;
14 Let T_j be the set of the first $k \times \frac{B_j.size}{sum}$ tuples in B_j;
15 Insert T_j into C_{sub};
16 Delete T_j from B_j;
17 **if** $B_j.size = 0$ **then**
18 Delete B_j from BS;
19 Add C_{sub} to SC;
20 **foreach** $B \in BS$ **do**
21 **foreach** *tuple* $t_i \in B$ **do**
22 Let C_{near} be t_i's nearest sub-cluster in SC;
23 Insert t_i into C_{near};
24 Delete B;
25 **foreach** *sub-cluster* $SC_i \in SC$ **do**
26 **foreach** *tuple* $\bar{t} \in SC_i$ **do**
27 Let $G_{\bar{t}}$ be the set of tuples in C, such that $G_{\bar{t}} = \{t \in C | t.pid = \bar{t}.pid\}$;
28 Insert $G_{\bar{t}}$ into SC_i;
29 Delete $G_{\bar{t}}$ from C;
30 Return SC;

Let us see now in more details how $split^{\ell}()$ works. As a first step, it generates the buckets BS without overlaps. This is done by calling function *generate_buckets()*. Then, if the number of disjoint buckets is less than ℓ, this means that the split is not possible. Thus, the input cluster C is returned (line 3). Otherwise, once the number of buckets is greater than ℓ and the number of tuples in BS is greater than k, a k_s-anonymized and ℓ-diversified subcluster C_{sub} is generated (lines 5-19). In this case, the function selects, from each bucket $B_j \in BS$ a subset of tuples T_j proportional to the bucket size (lines 10-15). More precisely, to reduce the information loss, the function selects the T_j tuples in B_j which require the minimum enlargement to C_{sub}. Such tuples are then inserted into C_{sub}. Finally, when one or both the conditions of the while loop are no more satisfied, remaining tuples are accommodated into the new created subclusters (lines 20-30).

4.3 Formal Results

In this section, we analyze the security of CASTLE. Complexity analysis, as well as the proofs of the theorems presented in this section, are reported in the supplementary material section, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TDSC.2009.47.

We start by proving that CASTLE generates k_s-anonymized data streams.

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

Proving that CASTLE generates k_s-anonymized data streams is not enough. As shown by Example 4, reuse of k_s-anonymized clusters should be carefully managed; otherwise, a potential attacker may infer additional knowledge about a tuple's value, or even guess its exact value. We have, therefore, to prove that the reuse strategy adopted by CASTLE is secure.

We first introduce some notions. Given a cluster generalization $G = (g_1, \ldots, g_n)$, the generalization g_j of the quasi-identifier attribute q_j gives information about the range in which the real values of q_j for tuples generalized by G fall. For instance, given the domain generalization hierarchy presented in Fig. 1, if the value given in output by CASTLE for the quasi-identifier attribute *Edu* is University, we can infer that the corresponding nonanonymized tuple can have as value for this attribute one of the elements in the set {Bachelor, Master, Ph.D.}. Similarly, if the value given in output by CASTLE for the quasi-identifier attribute *Age* is [25-30], then the *Age* attribute of the corresponding nonanonymized tuple has a value in the interval [25, 30]. In what follows, given a cluster generalization $G = (g_1, \ldots, g_n)$, we denote with $values(G.g_i)$ the set of values implied by the generalization g_i of attribute q_i. More precisely, if q_i is a continuous attribute, $values(G.g_i)$ contains all the values in the interval corresponding to $G.g_i$'s value. In contrast, if q_i is a categorical attribute, $values(G.g_i)$ contains all the leaves of the subtree in DGH_i rooted at $G.g_i$'s value, where DGH_i denotes the domain hierarchy for attribute q_i.

The k_s-anonymity property ensures that, even if an attacker knows $values(G.g_i)$, $\forall i \in [1, n]$, s/he is not able to link the anonymized tuple to an individual with a probability greater than $\frac{1}{k}$. However, as reported in Example 4, if an attacker is able to shrink the possible real values associated with $G.g_i$ to a subset of $values(G.g_i)$, by tracing the generalizations output so far, then s/he could infer that the anonymized tuple refers to an individual with a probability greater than $\frac{1}{k}$, as the following example better clarifies:

Example 6. Let us consider again Example 4. Suppose that $k = 4$ and QI is {*Age*, *Edu*}. Suppose that there are all together four persons P_1, P_2, P_3, and P_4, whose QI values fall in cluster C_3. Moreover, suppose that these four persons' QI values are: $P_1.QI = (24, \text{Bachelor})$, $P_2.QI = (25, \text{Master})$, $P_3.QI = (26, \text{Bachelor})$, and $P_4.QI = (27, \text{Master})$. At instant $\imath$, each of the four persons has at least one tuple in C_3, and C_3 becomes k_s-anonymized after outputting all its tuples by its generalization. After some time, $\tilde{t} = (25, \text{Master})$ is expiring, and it is output by C_3's generalization ([24, 27], University). However, as shown in Example 4, if the reuse of k_s-anonymized clusters always tries to minimize information loss, then an attacker can shrink the possible real values associated with $C_3.Age$ from [24, 27] to [24, 25]. By linking ([24, 25], University) to the QI values of the four persons, we know that only P_1 or P_2 can be the owner of $\tilde{t}$. Obviously, the probability of linking the generalized tuple of $\tilde{t}$ to P_1 or P_2 is increased from $\frac{1}{k} = \frac{1}{4}$ to $\frac{1}{2}$.

To avoid this possible inference attack, CASTLE adopts the cluster reuse strategy introduced in Section 3.2.2. As the following theorem states, this strategy ensures that an attacker cannot infer any additional knowledge on a tuple t

other than the set of real values associated with the generalization according to which it has been given in output.

Theorem 2. *Let $S(p, pid, a_1, \ldots, a_j, q_1, \ldots, q_n)$ be a stream where $\{q_1, \ldots, q_n\}$ are the quasi-identifier attributes. Let S_{out} be the stream generated by CASTLE, and $TG = \{G_1, \ldots, G_m\}$ be the generalizations corresponding to the tuples in S_{out}. Let t be a tuple and let G_r, $1 \leq r \leq m$, be its generalization performed by CASTLE. For each q_i, an attacker is not able to infer that the real value of $t.q_i$ belongs to a subset of* $\text{values}(G_r.q_i)$.

It is important to note that we are assuming a setting (Definition 2.3) where the adversary does not have any temporal background knowledge (e.g., s/he may know that Bob buys an item but not that Bob buys an item at time 10). Considering this further kind of background knowledge is a challenging issue, that we plan to investigate in the future, and that may require the definition of alternative techniques to achieve k_s-anonymity.

Since CASTLE may output some tuples before their expiring time, the ordering of the output stream may not be the same as that of the input stream. Let S be the input stream, and S_{out} be the corresponding output stream generated by CASTLE. We say that S and S_{out} have the same ordering, if the following condition holds: Given any two tuples $t_1, t_2 \in S$, let $\bar{t}_1, \bar{t}_2 \in S_{out}$ be their corresponding anonymized tuples, respectively. If $t_1.p < t_2.p$, then $\bar{t}_1$ is output earlier than $\bar{t}_2$. Some mining applications may be sensitive to the tuple ordering. If this is the case, we use the following method to reorder output streams w.r.t. input streams, so that their orderings are the same. When an original tuple t is anonymized to $\bar{t}$, CASTLE buffers it. $\bar{t}$ is released only after all the original tuples with position less than $t.p$ have been anonymized and given in output. The buffer incurs extra time. Now the total buffer time of a tuple t before its output is composed of two parts: 1) the time interval between its arrival and its anonymization, and 2) the buffer time for reordering. The following theorem proves that with the reordering, the total buffer time is upper bounded by δ, and the *delay constraint* (Definition 2.2) is still guaranteed.

Theorem 3. *Let S be an input stream, and S_{out} be its output stream generated and reordered by CASTLE. Let δ be an integer representing the maximum allowed delay between a tuple's input and its output. The delay constraint is guaranteed by the fact that the total buffer time of any tuple $t \in S$ before its output is upper bounded by δ.*

Note that, if an attacker has the knowledge of the ordering of the input stream, the reordering will enable him/her to find the released tuple of a victim. For the simplicity of discussion, we assume that tuples' positions in the input stream start from 1 and are consecutive. Suppose that Bob buys an item. Let t be the record for Bob's purchase. If an attacker knows the ordering of input tuples, that is to say, s/he knows that Bob buys the item at time $t.p$, s/he can infer that $t.p$-th tuple in the output stream belongs to Bob. Bob's privacy is violated. Therefore, if an attacker has the ordering knowledge, to protect the privacy, we need to add some randomness to the ordering of output streams.

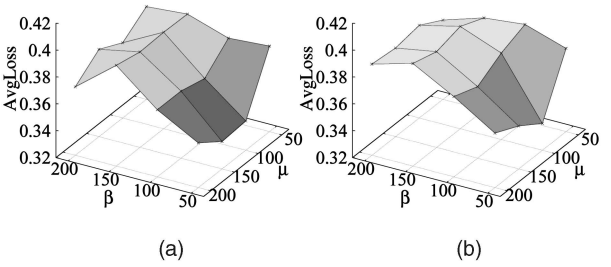

Fig. 4. Varying β and μ. (a) SFU-Adult. (b) UCI-Adult.

5 Performance Evaluation

We have implemented CASTLE and have conducted several experiments. 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. Second, to illustrate the effectiveness of CASTLE, we compare it with the approach presented in [5], 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] *UCI-Adult*, which has become a standard for studying k-anonymity. Moreover, in order to have a better simulation of a data stream, we have also considered the data set used in [18]. We refer to this as *SFU-Adult*.[6] 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*, *marital-status*, *occupation*, and *nation*. The first six of them are continuous, and the left four are categorical. We adopt from [18] the hierarchies for categorical attributes, and the intervals for continuous attributes. The experiments were conducted on an Intel Pentium IV 2.4 GHz with 1 GB RAM. In the following, we evaluate CASTLE using the metric described in Section 2.2.

5.1 Tuning CASTLE

The parameters that affect the performance of CASTLE are: δ, k, the number of QI attributes, μ, β, and the data distribution.

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 threshold for controlling the maximum number of non-k_s-anonymized clusters in memory. Fig. 4 presents the average information loss of k_s-anonymized tuples by simultaneously varying β and μ. We have set $k = 100$, $\delta = 10{,}000$ and have used 10 quasi-identifier attributes. We run the experiments on both the *SFU-Adult* (Fig. 4a) and *UCI-Adult* (Fig. 4b) data sets. In both data sets, $\beta = 50$ minimizes the information loss. In the experiment of *SFU-Adult*, $\mu = 100$ and $\mu = 150$ yield the best information quality; in the experiment of *UCI-Adult*, $\mu = 100$ outperforms $\mu = 150$. In the following experiments, we shall use $\beta = 50$ and $\mu = 100$ as the default values.

5. http://www.ics.uci.edu/~mlearn/MLSummary.html.
6. http://ddm.cs.sfu.ca/.

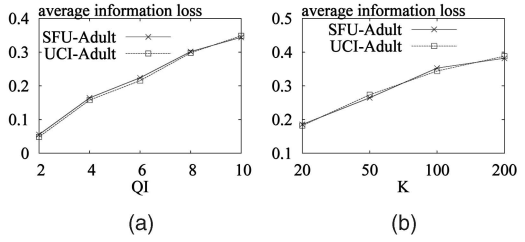

Fig. 5. Varying quasi identifiers and k. (a) Varying QI. (b) Varying k.

Effects of *quasi identifiers*. A further experiment measures how the size of QI affects the average information loss. The experiment has been conducted on the *UCI-Adult* and *SFU-Adult* data sets with $k = 100$ and $\delta = 10{,}000$. Fig. 5a reports how the average information loss varies by increasing the size of QI. The exploited metric space is defined based on QI attributes (e.g., each QI attribute is one dimension). When the size of QI increases, the data become more sparse in the defined space, and clusters are more likely to have "big" minimum bounding boxes. This is related to *the curse of dimensionality* [3]. Therefore, as expected, the information loss increases when increasing the number of QI attributes.

Effects of k. Fig. 5b shows how the average information loss increases by increasing the value of k. In this experiment, we have considered 10 quasi-identifier attributes, $\delta = 10{,}000$, 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. In order to do that, we have evaluated CASTLE on synthetic data sets following power-law distribution generated by means of genzipf.[7] More precisely, a value v generated by genzipf has the following probability property: $p(v) = \frac{c}{v^{\alpha}}, v \in \{1, \ldots, N\}$ and $\sum_{v=1}^{N} p(v) = 1$, where c is the normalization constant automatically initialized. Fig. 6 reports the average information loss with different α values. The experiments have been conducted with fixed k and by varying δ. We have evaluated the behavior of CASTLE w.r.t. data distribution with different k values (cf. Figs. 6a, 6b, 6c, and 6d). From the results, it is clear that CASTLE is very effective for clustered data. This is promising as real data are typically clustered.

Effect of multiple tuples referring to the same person. In the previous experiments, we have considered the UCI-Adult and SFU-Adult data sets, by assuming that each tuple refers to a distinct person, that is, each tuple has a different pid. However, since in a data stream multiple tuples may refer to the same person, we have conducted further experiments to evaluate how this duplication of pids may affect the information loss. In particular, to generate a data set with duplications, we exploit the IBM Quest Synthetic Data Generation Code [1]. By means of this code, we have generated a set of tuples T_{tran} with schema $(pid, tid, list\ of\ items)$, where pid is a customer's id, tid represents transaction id, and *list of items* are the items a customer has bought. Then, the considered data set is obtained by joining T_{tran} with UCI-Adult on their pid. Since T_{tran} is bigger than UCI-Adult, which contains 30,162 tuples, we join tuples of T_{tran} with $pid > 30{,}162$ with randomly selected tuples in UCI-Adult. As a result, tuples in the data set have the schema $(pid, tid, QI, list\ of\ items)$. Moreover, to simulate the presence of more transactions referring to the same person, when a tuple t of the data set is pushed into a cluster, CASTLE splits it into a set of tuples with schema $(pid, tid, QI, item)$, one for each distinct item contained in t. According to this approach, we have generated three different synthetic data sets, by varying the number of items in $list\ of\ items$. The experiments are conducted by setting δ equal to 10,000, and varying k and the size of QI. In particular, in Fig. 7a, we set $k = 100$ and vary the size of QI, where $\mathrm{dup} = 5$, $\mathrm{dup} = 10$, and $\mathrm{dup} = 15$ indicate the synthetic data set with 5, 10, and 15 items in the $list\ of\ items$, respectively. In general, when the size of QI increases, the information quality degrades. In Fig. 7b, we set the size of QI to be four, and vary k. We can observe that when k increases, the information loss is uniformly increased.

5.2 Utility

In this section, we examine the effectiveness of CASTLE by considering its accuracy in answering aggregate queries. In order to do this, we evaluate the same aggregate queries over S_{out} and S and compare the results. To evaluate the utility of CASTLE, we exploit the metric introduced in [30], according to which the evaluated queries are of the form:

$$\text{SELECT COUNT}(^*)\ \text{FROM}\ S_{out}$$
$$\text{WHERE pred}(q_1)\ \text{AND}\ \ldots\ \text{AND pred}(q_n)\ \text{AND pred}(A_s);$$

where:

- $q_1, \ldots, q_n$ are the QI attributes, whereas A_s is the sensitive attribute.
- Let $A \in \{q_1, \ldots, q_n, A_s\}$ be one attribute. $pred(A)$ is in the form of $A \in R$. If A is continuous, R is a randomly generated interval in the domain of A with length $|A| \cdot \theta^{\frac{1}{n+1}}$, where $|A|$ is the domain size of A and θ is the *expected selectivity*. If A is categorical, R is any node in the domain hierarchy of A, such that $\frac{|Leaves(R)|}{|Leaves(A)|} \approx \theta^{\frac{1}{n+1}}$, where $Leaves(R)$ is the set of leaves of the subtree rooted at R, and $Leaves(A)$ is the set of leaves in the domain hierarchy of A.

These queries have been evaluated both over the original stream S and over the generalized stream S_{out}. Given a query Q, let act be the query result on S, and est be the result on S_{out}. est is computed as the sum of the probability that generalized tuple $t \in S_{out}$ satisfies the WHERE clause in Q. Let $SetA = \{q_1, \ldots, q_n, A_s\}$. Thus, est is computed by the following formula:

$$est = \sum_{t \in S_{out}} P(t), \quad \text{where} \quad P(t) = \prod_{A \in SetA} P(t.A \in R).$$

In particular, if A is continuous, $P(t.A \in R) = \frac{|t.A \cap R|}{|t.A|}$. If A is categorical, $P(t.A \in R) = \frac{|leaves(t.A) \cap leaves(R)|}{|leaves(t.A)|}$.

7. http://www.csee.usf.edu/~christen.

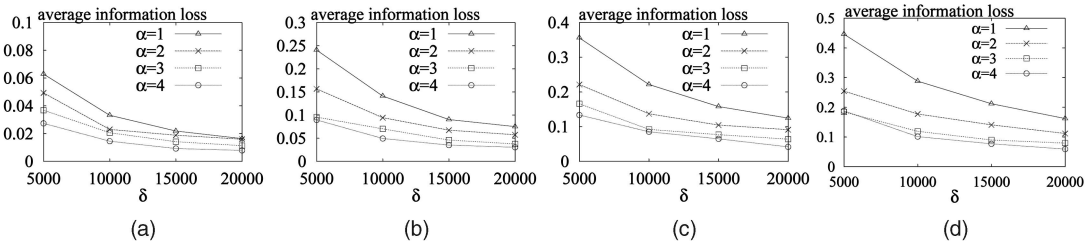

Fig. 6. Information loss on power-law synthetic data. (a) $k = 100$; (b) $k = 400$; (c) $k = 700$; (d) $k = 1{,}000$.

Thus, similar to [30], we measure $|act - est|/|act|$ as the *relative error* of Q. Given a window W in a data stream, we run 5,000 queries on W, and take the median relative error of these 5,000 queries as the *window error*. As W advances, a sequence of windows are generated. We calculate the window errors of all the generated windows, and consider their average as the *workload error*. We use *SFU-Adult*, and *UCI-Adult* data sets to simulate two data streams. Moreover, since *est* is calculated with the assumption that values of each attribute A are uniformly distributed in R, we do not consider those attributes that are nonuniformly distributed. Attribute *capital-gain* has the range [0, 99, 999] in *UCI-Adult* (30,162 tuples), but its value in 27,624 tuples is 0; attribute *capital-loss* has the range [0, 43, 56] in *UCI-Adult*, but its value in 28,735 tuples is 0. Therefore, neither *capital-gain* nor *capital-loss* is considered in the evaluation.

Fig. 8 reports the experiments, in which the window size $|W|$ is as large as δ. In particular, in Fig. 8a, we have considered four QI attributes, set $k = 100$ and selectivity $\theta = 0.1$, and vary δ. As expected, when δ increases the utility is better. In Fig. 8b, we consider four QI attributes, set $\delta = 10{,}000$ and $\theta = 0.1$, and vary k. When k increases, utility degrades since a larger k requires a cluster containing more tuples, which implies that its generalizations are enlarged. Fig. 8c presents the result of varying QI when $k = 100$, $\delta = 10{,}000$, and $\theta = 0.1$; the workload error does not change uniformly as the size of QI increases. However, in general, utility is reduced when the size of QI increases. Fig. 8d is the result of varying selectivity θ with four QI attributes, $k = 100$, and $\delta = 10{,}000$. Bigger θ yields a better accuracy since a bigger θ indicates bigger R and allows a larger generalization of attribute A. In all these experiments, the average median relative error of CASTLE is less than 13 percent, indicating high utility of anonymized streams.

5.3 Comparative Study

Finally, we compare the performance of CASTLE w.r.t. the approach presented in [5], hereafter called *dynamicGroup*.

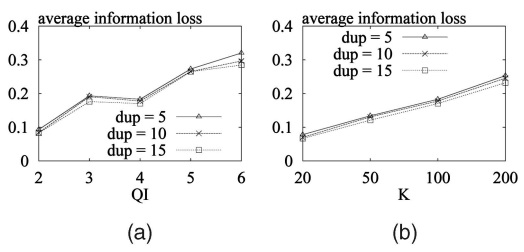

Fig. 7. Information loss on transaction stream. (a) Varying QI. (b) Varying K.

We performed the comparison w.r.t. the average information loss and the number of k-anonymized tuples. Since *dynamicGroup* can only process continuous attributes, the experiment has been performed with the six continuous QI 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 nonanonymized 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 2,000 to 8,000, and select the best one for *dynamicGroup*. The best n is 8,000. Fig. 9 reports how the average information loss of CASTLE and *dynamicGroup* vary with different δ values. We have considered several values of k (see Figs. 9a, 9b, 9c, and 9d). It is important to note that *dynamicGroup* does not consider the delay 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 δ set to *infinity*. As shown in Fig. 9, as δ increases, the information loss of CASTLE decreases. Moreover, when δ is 10,000, CASTLE outperforms *dynamicGroup*. We also compare CASTLE with *dynamicGroup* w.r.t. their effectiveness on utility. In Fig. 10, we consider four continuous quasi-identifier attributes (without *capital-gain* and *capital-loss*), and set $\delta = 20{,}000$. In Fig. 10a, we vary k, by setting $\theta = 0.1$, whereas in Fig. 10b we vary the selectivity θ, by setting $k = 200$. In both experiments, CASTLE outperforms dynamicGroup.

6 Related Work

Several techniques have been proposed and investigated to protect the privacy of data to be mined. Among these approaches, we can find methods based on perturbation [2], [5], [20] and k-anonymity [8], [15], [18], [21], [22], [28]. According to perturbation-based techniques, data privacy is ensured by releasing perturbed individual data or perturbed query results, where perturbation is performed, for example, by swapping [20], condensation [5], or adding noise [2]. However, perturbation-based methods have the weakness of tampering the integrity of released data. In contrast, the records within a k-anonymized data set remain true. In the recent years, several k-anonymity methods have

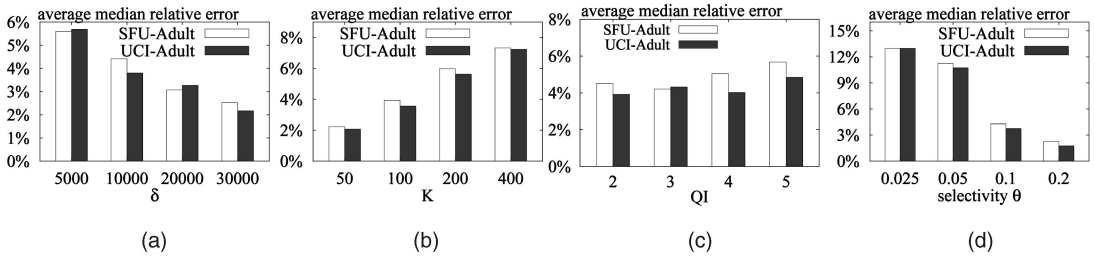

Fig. 8. Data utility. (a) Varying δ. (b) Varying k. (c) Varying QI. (d) Varying selectivity.

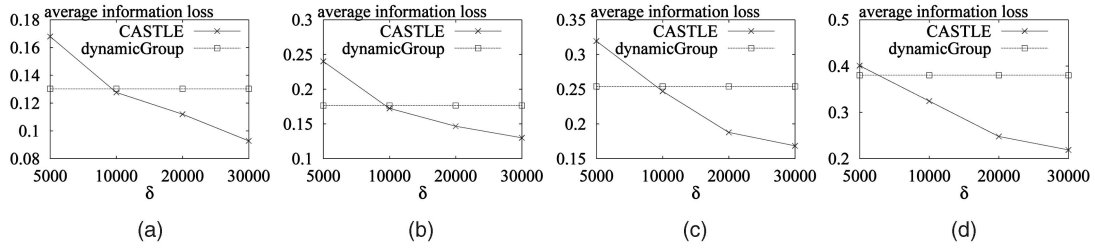

Fig. 9. Information loss on UCI-Adult. (a) $\mathrm{k} = 50$; (b) $\mathrm{k} = 100$; (c) $\mathrm{k} = 200$; (d) $\mathrm{k} = 400$.

been proposed [8], [15], [18], [21], [22], [28]. Recently, other anonymization techniques have been proposed to avoid possible inference attacks on k-anonymized data. For instance, Machanavajjhala et al. [24] proposed the ℓ-diversity principle. Along the same line is t-closeness [23], 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).

However, as pointed out in Section 2, all these methods are not adequate for data streams. In addition to the motivations presented in Section 2, we have to take also into account that all these methods require to scan the data to be anonymized multiple times, and output the anonymous data set in the final step. This assumption is unsuitable in the data stream scenario, in that, the algorithm for streaming data can only scan the data in one pass and executes in a pipeline manner.

Among all the k-anonymity methods proposed so far, CASTLE has some similarities with those exploiting cluster-based techniques. In particular, cluster-based algorithms for k-anonymizing static data sets have been proposed in [5], [6], [10], [16], [31]. References [16], [31] adapt micro-aggregation to achieve k-anonymity by requiring that the size of each cluster (group) after the partition is at least k. They have provable approximation bounds to optimal solution. However, they assume that all the data are available for anonymization, so how to extend them to the context of streaming data is not obvious. Aggarwal et al. [6] present a theoretical framework for grouping tuples into 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. In contrast, the approach presented in [10] adopts a distance metric similar to the one we adopt, but, like the approach in [6], it only considers static data sets and no delay constraint is enforced. An approach similar to [10] has been also proposed for the k-anonymization of incremental data sets [29]. However, this approach does not consider the possible inferences that an attacker can do by analyzing multiple releases of the same table, which makes the approach potentially insecure and thus inapplicable.

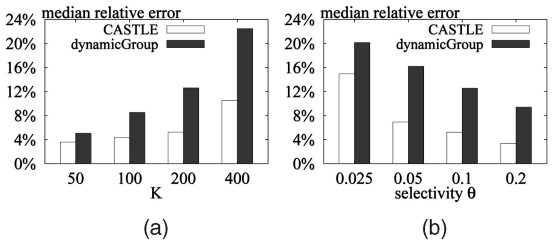

Fig. 10. Utility on UCI-Adult. (a) Varying k. (b) Varying selectivity.

It is interesting to note that most of the k-anonymity schemes proposed so far have been devised under the assumption that the data set to be anonymized is static, that is, the data set once has been anonymized does not change. Recently, the problem of anonymizing dynamic data sets, which evolve with insertions and deletions, has started to be investigated. Obviously, tuple insertions and deletions require to release a new k-anonymization of the updated data set. As shown in [9], [26], [30], multiple releases of k-anonymized data sets might be subject to potential inferences that may tamper the k-anonymity of the data. In this sense, anonymizing data streams and anonymizing dynamic data sets have some similarities, in that they both suffer from potential inferences due to dynamic updates. However, it is important to note, that the inferences that may arise when anonymizing dynamic data sets and those that might happen during anonymization of data streams are different, as the following discussion clarifies. Anonymizing a dynamic data set requires to publish multiple k-anonymized releases of a table. Here, the inference over the real values of a tuple t is possible since t could be present in more than one of the k-anonymized tables. Therefore, an attacker,

by analyzing the generalization of t in the different releases, may be able to narrow down its generalization or even to guess the real values of the quasi-identifier attributes of t, and thus to link t with a victim, as shown in [9], [26], [30]. When anonymizing data stream this inference does not take place, in that a tuple in the stream is anonymized only once. In contrast, when anonymizing data streams the possible inferences are due to the fact that the attacker is able to inspect the sequence of anonymized tuples given in output (as discussed in Section 4.3). Therefore, anonymization methods devised for dynamic data sets cannot be directly applied to anonymize data streams.

7 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 delay constraints, its adaptability to data distributions, and its cluster reuse strategy that improves the performance without compromising security. Performance evaluation reported in this paper has 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 [23]. 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, such as temporal background knowledge. Finally, we will study optimization techniques to improve the efficiency of our system.

Acknowledgments

Jianneng Cao and Kian-Lee Tan are partially supported by a research grant R-252-000-328-112 funded by the National University of Singapore. Barbara Carminati and Elena Ferrari are partially supported by the research project ANONIMO (PRIN-2007F9437X) funded by the Italian MIUR.

References

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," *Proc. Int'l Conf. Very Large Databases (VLDB),* pp. 478-499, 1994.

[2] R. Agrawal and R. Srikant, "Privacy-Preserving Data Mining," *Proc. SIGMOD,* pp. 439-450, 2000.

[3] C.C. Aggarwal, "On k-Anonymity and the Curse of Dimensionality," *Proc. Int'l Conf. Very Large Databases (VLDB),* pp. 901-909, 2005.

[4] C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu, "A Framework for Clustering Evolving Data Streams," *Proc. Int'l Conf. Very Large Databases (VLDB),* pp. 81-92, 2003.

[5] C.C. Aggarwal and P.S. Yu, "A Condensation Approach to Privacy Preserving Data Mining," *Proc. Int'l Conf. Extending Database Technology (EDBT),* pp. 183-199, 2004.

[6] G. Aggarwal, T. Feder, K. Kenthapadi, S. Khuller, R. Panigrahy, D. Thomas, and A. Zhu, "Achieving Anonymity via Clustering," *Proc. Symp. Principles of Database Systems (PODS),* pp. 153-162, 2006.

[7] M. Atzori, "Weak k-Anonymity: A Low-Distortion Model for Protecting Privacy," *Proc. Int'l Security Conf.,* pp. 60-71, 2006.

[8] R.J. Bayardo and R. Agrawal, "Data Privacy through Optimal k-Anonymization," *Proc. Int'l Conf. Data Eng. (ICDE),* pp. 217-228, 2005.

[9] J.W. Byun, Y. Sohn, E. Bertino, and N. Li, "Secure Anonymization for Incremental Data Sets," *Proc. Very Large Databases (VLDB) Workshop Secure Data Management,* pp. 48-63, 2006.

[10] J.W. Byun, A. Kamra, E. Bertino, and N. Li, "Efficient k-Anonymization Using Clustering Techniques," *Proc. Database Systems for Advanced Applications (DASFAA),* pp. 188-200, 2007.

[11] J. Cao, B. Carminati, E. Ferrari, and K.L. Tan, "CASTLE: A Delay-Constrained Scheme for k_s-Anonymizing Data Streams," *Proc. Int'l Conf. Data Eng. (ICDE),* Poster Paper, pp. 1376-1378, 2008.

[12] P. Domingos and G. Hulten, "Mining High-Speed Data Streams," *Proc. Int'l Conf. Knowledge Discovery and Data Mining (KDD),* pp. 71-80, 2000.

[13] P. Zhang, X. Zhu, and Y. Shi, "Categorizing and Mining Concept Drifting Data Streams," *Proc. Int'l Conf. Knowledge Discovery and Data Mining (KDD),* pp. 812-820, 2008.

[14] C. Luo, H. Thakkar, H. Wang, and C. Zaniolo, "A Native Extension of SQL for Mining Data Streams," *Proc. SIGMOD,* pp. 873-875, 2005.

[15] J. Domingo-Ferrer and V. Torra, "Ordinal, Continuous and Heterogeneous k-Anonymity through Microaggregation," *Data Mining and Knowledge Discovery,* vol. 11, no. 2, pp. 195-212, 2005.

[16] J. Domingo-Ferrer, F. Sebe, and A. Solanas, "A Polynomial-Time Approximation to Optimal Multivariate Microaggregation," *Computers and Math. with Applications,* vol. 55, no. 4, pp. 714-732, 2008.

[17] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering Data Streams," *Proc. IEEE Symp. Foundations of Computer Science (FOCS),* pp. 359-366, 2000.

[18] B.C.M. Fung, K. Wang, and P.S. Yu, "Top-Down Specialization for Information and Privacy Preservation," *Proc. Int'l Conf. Data Eng. (ICDE),* pp. 205-216, 2005.

[19] V.S. Iyengar, "Transforming Data to Satisfy Privacy Constraints," *Proc. Int'l Conf. Knowledge Discovery and Data Mining (KDD),* pp. 279-288, 2002.

[20] S. Kim and W. Winkler, "Masking Microdata Files," *Proc. Section on Survey Research Methods,* pp. 114-119, 1995.

[21] K. LeFevre, D.J. DeWitt, and R. Ramakrishnan, "Incognito: Efficient Full-Domain k-Anonymity," *Proc. SIGMOD,* pp. 49-60, 2005.

[22] K. LeFevre, D.J. DeWitt, and R. Ramakrishnan, "Mondrian Multidimensional K-Anonymity," *Proc. Int'l Conf. Data Eng. (ICDE),* p. 25, 2006.

[23] N. Li and T. Li, "t-Closeness: Privacy beyond k-Anonymity and ℓ-Diversity," *Proc. Int'l Conf. Data Eng. (ICDE),* pp. 106-115, 2007.

[24] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, "L-Diversity: Privacy beyond k-Anonymity," *Proc. Int'l Conf. Data Eng. (ICDE),* p. 24, 2006.

[25] P. Samarati and L. Sweeney, "Generalizing Data to Provide Anonymity When Disclosing Information," *Proc. Symp. Principles of Database Systems (PODS),* p. 188, 1998.

[26] J. Pei, J. Xu, Z. Wang, W. Wang, and K. Wang, "Maintaining K-Anonymity against Incremental Updates," *Proc. Int'l Conf. Scientific and Statistical Database Management (SSDBM),* p. 5, 2007.

[27] L. Qiu, Y. Li, and X. Wu, "Protecting Business Intelligence and Customer Privacy While Outsourcing Data Mining Tasks," *Knowledge and Information Systems,* vol. 17, no. 1, pp. 99-120, 2008.

[28] L. Sweeney, "Achieving k-Anonymity Privacy Protection Using Generalization and Suppression," *Int'l J. Uncertainty, Fuzziness, and Knowledge-Based Systems,* vol. 10, pp. 571-588, 2002.

[29] T.M. Truta and A. Campan, "k-Anonymization Incremental Maintenance and Optimization Techniques," *Proc. ACM Symp. Applied Computing (SAC),* pp. 380-387, 2007.

[30] X. Xiao and Y. Tao, "M-Invariance: Towards Privacy Preserving Re-Publication of Dynamic Data Sets," *Proc. SIGMOD,* pp. 689-700, 2007.

[31] M. Laszlo and S. Mukherjee, "Approximation Bounds for Minimum Information Loss Microaggregation," *IEEE Trans. Knowledge and Data Eng.,* vol. 21, no. 11, pp. 1643-1647, Nov. 2009.

[32] G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis, "Fast Data Anonymization with Low Information Loss," *Proc. Int'l Conf. Very Large Databases (VLDB),* pp. 758-769, 2007.

Jianneng Cao received the bachelor's and master's degrees in computer science from South China University of Technology in July 2002 and June 2005, respectively. He is currently a PhD candidate at the National University of Singapore. His research interests include data privacy, access control, data stream, applied cryptography, and data mining.

Barbara Carminati received the PhD degree in computer science in 2004 from the University of Milano, Italy. She is an assistant professor of computer science at the University of Insubria, Italy. Her main research interests are related to security and privacy for innovative applications, like XML data sources, semantic web, data outsourcing, web service, data streams, and social network. She has published numerous papers in international journals and conference proceedings. She is a member of the ACM and the IEEE.

Elena Ferrari received the PhD degree in computer science from the University of Milano, Italy, in 1998. She is a professor of computer science at the University of Insubria, Italy, where she heads the Database and Web Security Group. Her research activities are related to various aspects of data management systems, including web security, access control and privacy, and multimedia databases. On these topics, she has published more than 130 scientific publications in international journals and conference proceedings. She received the IEEE Computer Society's 2009 Technical Achievement Award for her outstanding and innovative contributions to secure data management. She is a member of the ACM and a senior member of the IEEE.

Kian-Lee Tan received the PhD degree in computer science in 1994 from the National University of Singapore (NUS). He is a professor of computer science at NUS. His current research interests include multimedia information retrieval, query processing and optimization in multiprocessor and distributed systems, database performance, and database security. He has published numerous papers in conferences such as the SIGMOD, the VLDB, the ICDE, and the EDBT, and journals such as the *TODS*, the *TKDE*, and the *VLDBJ*. He is a member of the ACM.