Disclosure Detection over Data Streams in Database Publishing

Deming Dou
Université de Lyon, CNRS
INSA-Lyon,LIRIS,UMR5205
F-69621, Villeurbanne, France
deming.dou@liris.cnrs.fr

Advisor: Asst. Prof. Dr. Stéphane Coulondre

ABSTRACT

As huge amount of personal data collected into databases of service providers increase, so does the risk of private information disclosure. Recently, a number of privacy-preserving techniques have been proposed, however, they are either pre-controls or post-controls and limited in protection of privacy without information loss or distortion, quite a few techniques can be seen among literatures for detecting information disclosure in the process of data transmission. In this paper, we focus on disclosure detection related to database publishing, and present a novel approach of detecting privacy leakages over data streams on querying databases by using dynamic pattern matching and data stream processing techniques. Experimental results via Cayuga system verified the feasibility of our proposal.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Security, Integrity, and Protection

General Terms

Security, Algorithms, Verification

Keywords

Privacy preservation, Disclosure detection, Data stream, Continuous query, Pattern matching, Cayuga system

1. INTRODUCTION

In today's world of universal data exchange, as the danger of privacy leakage increases, the data publishers have to face trade-offs between publishing highly valuable data for better service and protecting sensitive information from unsolicited and unsanctioned disclosure. This undertaking is called Privacy-preserving data publishing (PPDP). Solutions to this problem cannot be achieved by any physical protection mechanism, such as access controls, audit trails and firewalls. In the past decade, PPDP has received considerable attention and many effective approaches have been proposed for publishing useful information while limiting privacy breaches.

However, most of the previous work fall into the pre-control category. De-identification based methods [11] which strip away the explicit identifier from data records cannot provide protection against *linking attacks* [12]. Anonymization-based methods which transform a dataset to meet some privacy principle using techniques like generalization or suppression so that linking attacks can be completely eradicated. Perturbation-based methods which perturb individual data values or results of queries by swapping, adding noise or condensation. To some extent these solutions unavoidably introduce information loss or data distortion. We refer readers to the survey [6] for further details. Recently, some researchers extended these models to privacy preservation over data streams [14, 9, 3, 15] and developed some efficient algorithms. But none of them is easily applicable to detect and prevent privacy leakages in the process of data transmission for two main reasons. First, the transmitting data are under the form of streaming data, they are continuous, transient and usually unbounded flows of ordered tuple pairs $\langle tuple, timestamp \rangle$. Second, they do not support time constraints unless we add manually timestamps to data tables or views.

A theoretical study similar to our research is the query-view security problem. In 2004, Miklau et al. [10] proposed the concept of *perfect-security* and presented a information-theoretic stantard for query-view security which can be used to provide precise analyses of information disclosure between relational views and queries, although this definition can capture very subtle disclosure, but sometimes too strict. The issue of relaxing it was later investigated in [4]. Accordingly, Stefan et al. [2] detected privacy violations in sensitive XML databases and proposed an algorithm which identifies a set of suspicious XPath queries with respect to an audit XPath query. Goyal et al. [7] proposed an algorithm for rewriting of queries of a specified malafide intension to detect the privacy disclosure through either explicit accesses or inferencing by functional dependencies. Millist et al. [13] proposed a method of detecting possible privacy violations using disjoint queries, based on determining if a secret query and a published view can return tuples in common.

EDBT/ICDT PhD 2011, 21-MAR-2011, Uppsala, Sweden.

In this paper, we focused on disclosure detection with respect to database publishing and proposed a process detection method over relational data streams using dynamic pattern matching and data stream processing techniques. We treated each privacy leakage as an *event*, and we aimed to detect these events "*on the fly*", this can be seemed as some kind of *"process monitoring"*. To the best of our knowledge, this research direction has never been followed before.

2. CONCEPTS AND NOTATIONS

In contrast to macro-data, many situations call today for the release of entity-data, while this has led to concerns that some personal data may be abused or privacy may be accidentally disclosed. When a data publisher releases some person-specific information or when users submit queries containing sensitive attributes to a database, the best way to prevent privacy from disclosure is detecting all the leaking channels and then blocking them in the process of data transmission, we can regard such a procedure as some kind of process-controls. In this section, we present some correlative background knowledges and notations.

2.1 Microdata and Privacy Violation

Typically, a private table T has the most basic form:

$$D(A^{ei}, A^{qi}, A^{s}, A^{ns})$$

where A^{ei} denotes Explicit Identifier which is a set of attributes, such as name and social security number(SSN), containing information that explicitly identifies record owners; A^{qi} denotes Quasi-Identifier which is a minimal set of attributes that can be joined with external datasets to reidentify individual records (with sufficiently high probability); A^{s} denotes Sensitive attributes which consist of sensitive person-specific information such as disease, salary, and disability status; and A^{ns} denotes Non-sensitive attributes which cover all attributes that do not fall into the previous three categories.

Let T_d a release candidate of T produced by de-identification mechanism:

$$T_d = (A_1, A_2, \ldots, A_n) = T - \left\{A^{ei}\right\}$$

It is obvious that T_d itself can guarantee limits on privacy breaches as a result of the suppression of A^{ei}. But, suppose there is another internal or external available table T_e:

$$T_e = (B_1, B_2, \ldots, B_m)$$

if $A^{qi} \in T_d \cap T_e$ and $\left\{A^{ei}\right\} \cap T_e \neq \emptyset$, then an adversary may obtain some sensitive attributes by joining T_d and T_e.

Example 1. Given two tables: table T_d=(zipcode, date of birth, sex, disease) is to be published, and each tuple in T_d is about an individual; table T_e=(name, zipcode, date of birth, sex) are publicly available. If an adversary knows that A^{ei}=(SSN, name) and A^{qi} = (zipcode, date of birth, sex), according to the principles described above, the adversary can easily infer that some individual named *name* has an illness *disease* by linking T_d and T_e together through A^{qi}, here privacy breach occurs even though A^{ei} attributes in T_d are removed before being published.

In real world, the data are usually distributed to different websites and applications, the adversaries can track multiple datasources and mine meaningful, secret or sensitive information. In this paper, we pay only attention to the detection of privacy breaches in the process of data transmission from a single relational database to its applications.

2.2 Relational Data Streams

Instead of conducting the research on various database models, such as relational or XML, we build data streams directly on top of a relational database. Traditionally, in PPDP, most of person-specific data sets are collected and stored in a relational database, at a certain moment, it stores sets of relatively static records, with insertions, updates, and deletions executed less frequently than queries.

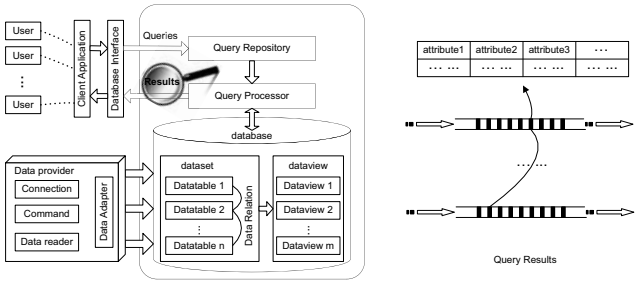

Figure 1: Data flow in a relational database

As shown in Figure 1., when users perform some browsing or searching operations, the client application will fire one-time queries over the database via database interface, over data tables or data views, to be exact, after a series of internal treatment, the records of query results will be transmitted to the client application and displayed to different users. In this process, because of the dynamics of query plans and scheduling policies, it is not a one-to-one correspondence between the order of results and that of queries. Therefore, to analyse the data transmission in terms of query statements is unpractical. Here, we introduce data stream conception.

A data stream is either a sequence of records that log interactions between entities or that monitor evolution of entity states. A typical data stream is made up of two elements: the data itself, which includes a series of data tuples and a timestamp which denotes the logical arrival time of a tuple, not system or wall-clock time. The timestamp is part of the schema of a data stream, and there could be multiple data tuples with the same timestamp.

Definition 1. **(Data stream)** A data stream S is an ordered spatio-temporal sequence of records with the tag of timestamps: $\{(s[0], t0), (s[1], t1), ...\}$, where $s[i]$ can be a z-tuple containing attributes $a[0], ..., a[z-1]$.

For an database application registered by millions of end-users, it's common that a large number of queries are submitted in a relatively short time interval, although a conventional database query executes once and returns a small set of records, the execution of multiple concurrent queries may introduce a large quantity of relational tuples in a time dependent manner which make up the full pieces of information of data transmission (see Figure 1.), and therefore data elements are continuous and unbounded, to some extent, several macroscopic "data streams" are generated.

Definition 2. **(Relational data stream)** Let $\mathrm{V}_1, \mathrm{V}_2, \ldots, \mathrm{V}_\mathrm{m}$ a set of views, respectively with schema $\mathrm{R}_1, \mathrm{R}_2, \ldots, \mathrm{R}_\mathrm{m}$, assume that $\mathrm{A}_1, \mathrm{A}_2, \ldots, \mathrm{A}_\mathrm{n}$ is a set of answer records corresponding to queries $\mathrm{Q}_1, \mathrm{Q}_2, \ldots, \mathrm{Q}_\mathrm{n}$, according to the schema of each A_i, we can dynamically divided them into k ordered groups $\mathrm{S}_1, \mathrm{S}_2, \ldots, \mathrm{S}_\mathrm{k}$, and each group S_i is a relational data stream and has its own schema, each tuple $s[j] \in S_i$ has a timestamp t[j]. This can be seen as an operator which takes a relation as an input and produce a stream as an output.

In this definition, every two relational data streams do not have the same relational schema, but their original data sources may be the same tables or views. With the new submission of queries, new tuples may be added into existing data streams or new data streams may be generated.

3. PROBLEM DEFINITION

In contrast to the problem of finding frequent itemsets in data streams, the privacy breach is relatively a small probability event. In this section, we discuss the problem of detecting information disclosure on the context of data streams.

Definition 3. **(Time interval)** Let $t[i]$ and $t[j]$ denote the timestamps of tuple $s[i]$ and $s[j]$ where $s[i] \in S_a$ and $s[j] \in S_b$ s.t. $a \neq b$, if $t[i] \neq t[j]$, then $T_{int} = |t[j] - t[i]|$ indicates the time interval of two tuples belonging to two different data streams respectively.

if $T_{int} > 0$ then the logical arrival time of $t[j]$ is prior to that of $t[i]$, conversely, $t[i]$ comes logically into data stream before $t[j]$. Then the problem can be specified in Figure 2.

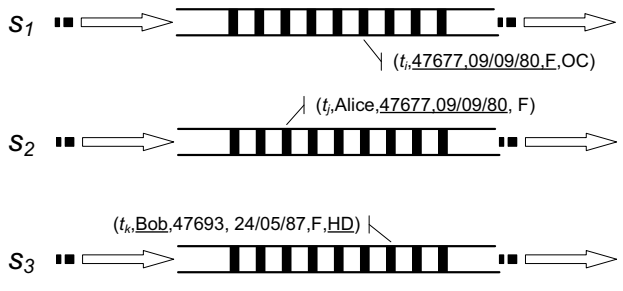

Figure 2: Relational data streams

Let S_1, S_2, S_3 be three data streams in which each tuple comes with a timestamp at a rapid rate. Assume an interval constant T_c gives the range of data for discovering privacy breaches. For linking attack, the two tuples showed in S_1 and S_2 have the same value in three common attributes, if their combination is a quasi-identifier and $|t[j] - t[i]| < T_c$, then information disclosure occurs, we can infer that *Alice* has the disease OC. Another kind of disclosure can be found in S_3, here the tuple itself contains both an A^{ei} attribute *Bob* and an A^s attribute HD, this is a so-called total disclosure.

In order to detect such kind of leaking channels, we should to real-timely monitor masses of continuous data streams, the most effective way is to deploy long running queries which are called continuous queries. Relational data streams feed raw data to them, they filter and output detected results.

Definition 4. **(Continuous query)** A continuous query operates either on relational data streams or on a combination of data streams and relations, and meanwhile produces results continuously, each of its results is also a new data stream. We issue it once and it monitors and processes streaming data efficiently, continuously and real-timely along with the arrival of new tuples.

First, let us consider the simplest scenario, suppose we have a single data stream S and a continuous query Q (can be considered as a "*secret query*" [10]), we are interested in the exact answer to Q (as opposed to an approximation) over S.

Example 2. Let *(SSN, Name, Zipcode, date of birth, Sex, Disease)* the schema of the data stream S, SSN and $Name$ are *Explicit Identifier* attributes, *Disease* is a *Sensitive* attribute. The objective of the continuous query Q is to find out all the tuples with the concurrence of the value of *Explicit Identifier* and *Sensitive* attributes as new tuples arrive.

However, in real-life world, distributed databases and applications and users make the scenario much more complicated, and in most cases we have to deal with multiple data streams together to discover our interesting information, meanwhile, the number of continuous queries could be larger than 1. As shown in Figure 3., a series of continuous queries operate over a number of incoming data streams $Stream\ 1, Stream\ 2, \ldots Stream\ n$ and output the answer streams $Stream\ 1, Stream\ 2, \ldots Stream\ m$. In order to detect some specific privacy breaches, joining or merging two or more data streams is also needed.

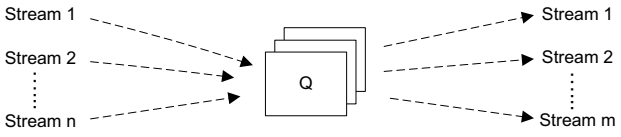

Figure 3: Continuous queries over data streams.

4. PROPOSED STRATEGIES

In our research, we treat each information disclosure as an event, directly or indirectly, totally or partially. In advance, we translate continuous queries to specific event patterns,

and then, we combine event pattern matching techniques with data stream processing techniques to check the presence of the constituents of a specific pattern. Event patterns match whenever an event satisfies the definition of the pre-determined pattern. Sometimes, in pattern matching, we need to join or merge event streams, and to feed pattern match results from one event stream to subsequent statements for further analysis and processing. Moreover, we need to integrate database functionality by integrating streaming operations with a specific query language and register long-running pattern matching queries to derive and aggregate interesting information from one or multiple streams of events by comparison to a standard of some kind.

Definition 5. **(Stream pattern)** In the operational semantics, a stream pattern P is a combination of a set of constraint matching rules: $Rule[0], Rule[1], \ldots$ which must be satisfied by discrete records of relational data streams.

Consider a simple example, assume a stock ticker stream with the schema $Stock(Name, Price, Volume)$, if a pattern is to find a monotonically increasing run of prices for all companies, where the run lasts for at least 10 stock quotes, and the first quote has a volume greater than 10000. The rules of this pattern can be followings:

Rule[0]: *Company s.t. all companies*
Rule[1]: *Price s.t. monotonically increasing*
Rule[2]: *Time interval s.t.* 10 *stock quotes*
Rule[3]: *First quote s.t. greater than* 10000

In these rules, $Company, Price, Time$ *and* $First\ quote$ can be considered as attributes of a data table, each of them has their own attribute domain or constraint conditions. Along with the increase of the number of the rules, the pattern will be more complex, and so will the difficulty of pattern matchings over data streams.

Pattern-matching developed for string matching that deals with equality and inequality between characters can be extended to data streams that require more complex relational comparison among their attributes [8]. Patterns matching for detecting events (*e.g.* privacy breaches) over data streams can involve unbounded looping operations, and pattern changes has been largely human-driven and infrequent.

Definition 6. **(Pattern matching)** Over data streams, pattern matching is an act of checking the satisfaction of rules $Rule[0], Rule[1], \ldots$ which compose a stream pattern P, and the matching results are exact and accurate.

Recall that pattern matching over two views. Assume a relational schema is $Employee(Name, Department, Phone)$, denoted as $Emp(n, d, p)$, let $V_1(n, d) : -Emp(n, d, p)$ and $V_2(d, p) : -Emp(n, d, p)$, if a pattern P is a secret query expressed as $S(n, p) : -Emp(n, d, p)$ which means that *phone* must not be inferred with regard to *name*. For the first step, two views should be joined $V1 \bowtie_\theta V2$, returns the set of all pairs $\langle r1, r2\rangle$, where $r1 \in V1$, $r2 \in V2$, and if join condition $\theta(r1, r2)$ is true, that means $r1.d = r2.d$, then the pattern P can be satisfied. A straightforward extension of pattern matching to streams gives the following semantics: within any time interval $T_{int} > 0$, the set of output tuples generated thus far by the pattern matching over two streams $S1$ and $S2$ should be the same as the result of the relational (non-streaming) pattern matching over the sets of input tuples that have arrived thus far in $S1$ and $S2$. The only difference between them is the time interval T_{int}.

Definition 7. **(Complex event processing)** Complex Event Processing, or CEP, is a technology to process events and detect both opportunities and threats, to discover complex patterns among multiple streams of event data.

Complex event processings enable organizations to real-timely monitor, analyze high-speed and continuous information, and enable data managers to make real-time calculations of and take action on disclosures. Amongst some types of CEP, detection-oriented CEP and operations-oriented CEP can be integrated within the research of privacy preservation over data streams. Detection-oriented CEP is used for data analysis and is driven by a need to detect faster instead of more accurate the possible risks. Operation-oriented CEP is driven by a need to make reaction faster instead of better to the risks. Of course, these two CEPs are closely and inseparably related. For example, if our interest is: "Notify me when there is a direct information disclosure or some linking attack". We should firstly detect if the disclosure exist and then send a notification to database administrators.

Algorithm 1: Disclosure Detection Over Data Streams

Input: two input data streams: S_1 *and* S_2.
Output: output data streams T

set quasi-identifier A^{qi} and time interval T_c
for $\forall a \in S1.A^{ei}$,$\forall b \in S1.A^s$ **do**
  if $a \neq NULL \wedge b \neq NULL$ **then**
    output (a, b) *as* T_1
  else
    for $\forall c \in S2.A^{ei}$,$\forall d \in S2.A^s$ **do**
      if $c \neq NULL \wedge d \neq NULL$ **then**
        output (c, d) *as* T_1
      else
        if $|t[a] - t[b]| < T_c$ **then**
          if $(a \neq NULL \wedge d \neq NULL) \cup (A^{qi} \in (S_1 \cap S_2))$ **then**
            output (a, d) *as* T_2
          else
            if $(b \neq NULL \wedge c \neq NULL) \cup (A^{qi} \in (S_1 \cap S_2))$ **then**
              output (c, b) *as* T_2
            end if
          end if
        end if
      end for
  end if
end for

Here, we adopt the disclosure classification in [10], total, partial and minute disclosure. Assume the same relational schema $Employee(Name, Department, Phone)$, for an example of total disclosure, given a view-query pair $V_1(n,d) : -Emp(n,d,p)$ and $S_1(d) : -Emp(n,d,p)$, obviously S_1 is answerable using V_1; for partial disclosure, if we connect two views $V_2(n,d) : -Emp(n,d,p)$ and $V_2^{'}(d,p) : -Emp(n,d,p)$ together, then the query $S_2(n,p) : -Emp(n,d,p)$ can be contained rewritten using V_2 and $V_2^{'}$, this can be also considered as some kind of linking disclosure; for minute disclosure, assume we publish a view $V_3(n) : -Emp(n,d,p)$, and submit a query $S_3(p) : -Emp(n,d,p)$ over V_3, seemingly there is no disclosure, but the view does can reveal something like the size of the Employee relation and contains some small amount of information about the omitted column.

In this paper, we aim only to detect the total and partial information disclosure over data streams through pattern matching techniques, among continuously arriving data streams, we match them by pairs in random to check the existence of these two disclosures, the detection algorithm is shown in Algorithm 1. In this algorithm, we take only two input data streams S_1 and S_2 into account, for the rest situations of multiple data streams, this algorithm is also applicable. We assume that we know the quasi-identifier A^{qi} beforehand, and then we set the time interval constant T_c by requirements. For each explicit identifier attribute a in S_1 and c in S_2, as well as each sensitive attribute b in S_1 and d in S_2, after we carry out the dynamic pattern matching of these two streams, we can get the final output data stream T containing total or partial disclosed information.

5. FEASIBILITY VERIFICATION

To verify our proposals, we conducted experiments on relational data streams with sensitive attributes. We want to examine whether our proposals can deal with the detection of privacy disclosure with high data arrival rates. The experiments are preceded by the following two steps. First, we chose and configured Cayuga System which is designed to monitor streams of events. After that, we specified the data source of input data streams. Finally, we defined our continuous queries to state which situations should be detected.

5.1 Cayuga System

Cayuga [5] is a scalable event processing system with a query language based on Cayuga Algebra for naturally expressing complex event patterns.

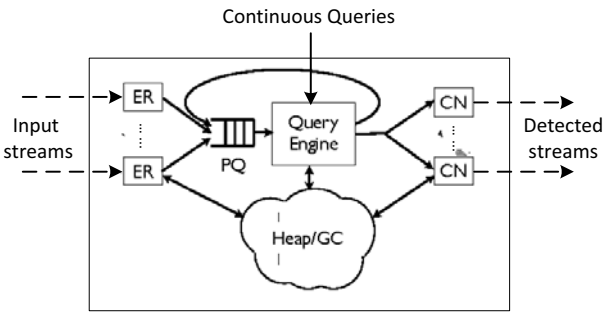

Figure 4: Cayuga system

As shown in Figure 4., input data streams are received by Cayuga Event receivers (ERs), and then passed to other components, output event streams will not be generated until Cayuga query engine executes continuous queries expressed by Cayuga event language (CEL).

5.2 Input Data Streams

The input data streams are generated from the answer of user queries, consider a relational data table with the schema:

Patient(*SSN, Name, Zipcode, DOB, Sex, Disease*)

we have two published views representing two subsets of a data table under this schema are as follows:

(a) View 1

SSN	Zipcode	DOB	Sex	Disease
387-399	47677	09/09/80	F	OC
387-200	47602	24/05/87	F	OC
387-486	47678	08/11/82	M	PC
387-756	47905	27/08/66	M	Flu
387-665	47909	04/10/57	F	HD
387-588	47906	10/10/62	M	HD
⋮	⋮	⋮	⋮	⋮

(b) View 2

Name	Zipcode	DOB	Sex
Alice	47677	09/09/80	F
Bob	47983	24/05/87	M
Carol	47677	08/11/82	F
Dan	47532	27/08/66	M
Ellen	47789	04/10/57	F
Jack	47487	08/05/84	M
⋮	⋮	⋮	⋮

Table 1: Two relational views

If users submit two queries to view 1 and view 2 seperately, and each of them does not breach the privacy violation:

$Q(Zipcode, DOB, Sex, Disease) : -View\ 1$
$Q(Name, Zipcode, DOB, Sex) : -View\ 2$

when the query answers are transmitted to them, two relational data streams could be generated, however, because of the special requirements of Cayuga System in stream schema, here we assumed that all the data streams consist of a special single relation - universal relational assumption [1], see Table 2.

SSN	Name	Zipcode	DOB	Sex	Disease	SNo	TStamp
		47677	09/09/80	F	OC	S1	(1,1)
		47602	24/05/87	F	OC	S1	(2,2)
		47678	08/11/82	M	PC	S1	(3,3)
		47905	27/08/66	M	Flu	S1	(4,4)
		47909	04/10/57	F	HD	S1	(5,5)
		47906	10/10/62	M	HD	S1	(6,6)
		⋮	⋮	⋮	⋮	⋮	⋮
	Alice	47677	09/09/80	F		S2	(4,4)
	Bob	47983	24/05/87	M		S2	(5,5)
	Carol	47677	08/11/82	F		S2	(6,6)
	Dan	47532	27/08/66	M		S2	(7,7)
	Ellen	47789	04/10/57	F		S2	(8,8)
	⋮	⋮	⋮	⋮		⋮	⋮

Table 2: Input data streams

5.3 Continuous Queries

In our example, SSN and Name are two explicit identifier attributes, {Zipcode, DOB, Sex} is a quasi-identifier. As stated before, we want to detect total and partial disclosure,

if we set the time interval constant T_c as 5 time units, we can give out continuous queries expressed in CEL easily. However, using query language, in order to detect a very simple disclosure, we must write lots of sub-queries, and the more the data streams, the explicit identifier attributes and the sensitive attributes are, the more the query number is. For two data streams here, the continuous queries and the processing flow is shown in Figure 5.

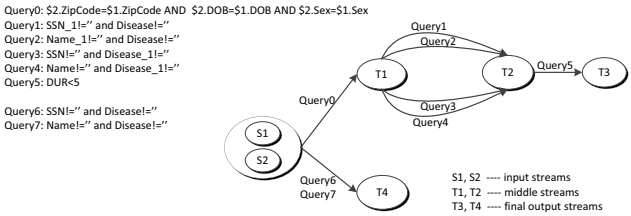

Figure 5: Continuous queries and processing flow

through the process of continuous queries in Cayuga query engine, the number of output results is bigger than 0, see Table 3, that means that some disclosures occur, as an example, we can inference that Alice suffers from the OC disease:

SSN	Name	Zipcode	DOB	Sex	Disease
	Alice	47677	09/09/80	F	OC
	⋮	⋮	⋮	⋮	⋮

Table 3: Output results

6. CONCLUSION AND FUTURE WORK

In this paper, we tackled the problem of detecting total and partial information disclosure over relational data streams. We adopted pattern matching and event processing techniques to detect these leaking channels, and then verified the feasibility of our proposals via an open source CEP software Cayuga system. To the best of our knowledge, our research has led to a new interesting point of view of disclosure detection with respect to query-view security.

However, to recognize automatically the schema of data streams, to develop efficient algorithms for finding quasi-identifiers, to find a "sweet spot" between expressiveness and performances in stream processing, and to define a prohibited correlation expression model for relational data expressing what data must not be inferred are still great challenges.

7. REFERENCES

[1] C. Beeri, P. A. Bernstein, and N. Goodman. A sophisticate's introduction to database normalization theory. In *Proceedings of the fourth international conference on Very Large Data Bases - Volume 4*, VLDB'1978, pages 113–124. VLDB Endowment, 1978.

[2] S. Bttcher and R. Steinmetz. Detecting privacy violations in sensitive xml databases. In W. Jonker and M. Petkovic, editors, *Secure Data Management-SDM 2005, 2nd VLDB Workshop on Secure Data Management*, volume 3674 of *Lecture Notes in Computer Science*, pages 143–154. Springer Berlin / Heidelberg, 2005.

[3] J. Cao, B. Carminati, E. Ferrari, and K.-L. Tan. Castle: A delay-constrained scheme for ks-anonymizing data streams. In *ICDE*, pages 1376–1378, 2008.

[4] N. Dalvi, G. Miklau, and D. Suciu. Asymptotic conditional probabilities for conjunctive queries. In *ICDT*, 2005.

[5] A. Demers, J. Gehrke, and B. P. Cayuga: A general purpose event monitoring system. In *In CIDR*, pages 412–422, 2007.

[6] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.*, 42(4):1–53, 2010.

[7] V. Goyal, S. K. Gupta, and S. Saxena. Query rewriting for detection of privacy violation through inferencing. In *PST '06: Proceedings of the 2006 International Conference on Privacy, Security and Trust*, pages 1–11, New York, NY, USA, 2006. ACM.

[8] L. Harada. Pattern matching over multi-attribute data streams. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval*, SPIRE 2002, pages 187–193, London, UK, 2002. Springer-Verlag.

[9] L. Jianzhong, O. B. Chin, and W. Weiping. Anonymizing streaming data for privacy protection. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 1367–1369, Washington, DC, USA, 2008. IEEE Computer Society.

[10] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 575–586, New York, NY, USA, 2004. ACM.

[11] A. M. Snyder and A. C. Weaver. Health insurance portability and accountability act in 1996.

[12] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):571–588, 2002.

[13] M. W. Vincent, M. Mohania, and M. Iwaihara. Detecting privacy violations in database publishing using disjoint queries. In *EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology*, pages 252–262, New York, NY, USA, 2009. ACM.

[14] W. Wang, J. Li, C. Ai, and Y. Li. Privacy protection on sliding window of data streams. In *COLCOM '07: Proceedings of the 2007 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 213–221, Washington, DC, USA, 2007. IEEE Computer Society.

[15] B. Zhou, Y. Han, J. Pei, B. Jiang, Y. Tao, and Y. Jia. Continuous privacy preserving publishing of data streams. In *EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology*, pages 648–659, New York, NY, USA, 2009. ACM.