Volume 5, Issue 4, April 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Detecting the m-Advesaries in Multi-Cloud Computing Using Bottom-Up Algorithm

[1]T. Jayalakshmi, [2]J. R. Thresphine
[1]PG Student, PRIST University, Puducherry, India
[2]Assistant Professor, Department of Computer Science and Engineering,
PRIST University, Puducherry, India

Abstract: The main problem in shared data is multiple data providers. We consider a new type of attack by data providers who may use their own data records in addition to the external background knowledge to infer the data records contributed by other data providers. In this we addresses this kind of threat and makes several contributions. First, we introduce the notion of m-privacy, which guarantees that the anonymized data satisfies a given privacy constraint against m-colluding data providers. Second, we present the algorithms exploiting the equivalence group monotonicity of privacy constraints and adaptive ordering techniques for efficiently checking m-privacy given a set of records. Finally, a data provider-aware anonymization algorithm with adaptive m-privacy checking strategies to ensure high utility and m-privacy of anonymized data with efficiency.

Keywords: m-privacy, Bottom-up Algorithm

I. INTRODUCTION

There is an increasing need for sharing data that contain personal information from distributed databases. For example, in the healthcare domain, share of information among hospitals and other providers, and support appropriate use of health information beyond direct patient care with privacy protection.

Privacy preserving data analysis and data publishing have received considerable attention in recent years as promising approaches for sharing data while preserving individual privacy. When the data are distributed among multiple data providers or data owners, two main settings are used for anonymization . One approach is for each provider to anonymize the data independently (anonymize-and-aggregate), which results in potential loss of integrated data utility. A more desirable approach is *collaborative data publishing* , which anonymizes data from all providers as if they would come from one source (aggregateand-anonymize), using either a trusted third-party (TTP) or Secure Multi-party Computation (SMC) protocols to do computations .

Existing System

We assume the data providers are semi-honest, commonly used in distributed computation setting. They can attempt to infer additional information about data coming from other providers by analyzing the data received during the anonymization. A data recipient could be an attacker and attempts to infer additional information about the records using the published data and some background knowledge such as publicly available external data.

Proposed System

We consider the collaborative data publishing setting with horizontally partitioned data across multiple data providers, each contributing a subset of records. As a special case, a data provider could be the data owner itself who is contributing its own records. This is a very common scenario in social networking and recommendation systems. Our goal is to publish an anonymized view of the integrated data such that a data recipient including the data providers will not be able to compromise the privacy of the individual records provided by other parties.

II. ARCHITECTURE

2.1 Protocol layer

Protocol layer implements the external interface to SQL Server. All operations that can be invoked on SQL Server are communicated to it via a Microsoft-defined format, called Tabular Data Stream (TDS). TDS is an application layer protocol, used to transfer data between a database server and a client. Initially designed and developed by Sybase Inc. for their **Sybase SQL Server** relational database engine in 1984, and later by Microsoft in Microsoft SQL Server, TDS packets can be encased in other physical transport dependent protocols, including **TCP/IP**, **Named pipes**, and **Shared memory**. Consequently, access to SQL Server is available over these protocols. In addition, the SQL Server API is also exposed over bando **web services**.

2.2 Data Storage

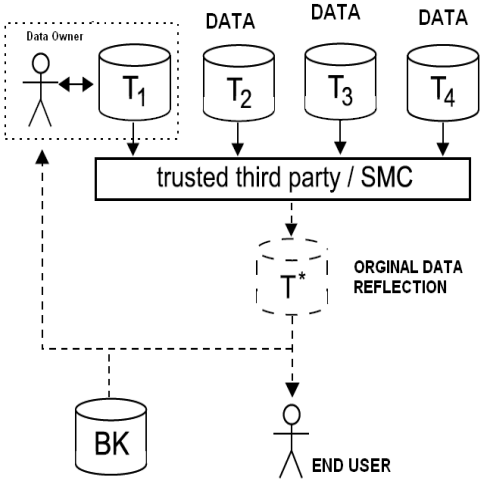
The main unit of **data storage** is a **database**, which is a collection of tables with **typed** columns. SQL Server supports different data types, including **primary types** such as *Integer*, *Float*, *Decimal*, *Char* (**including character strings**), *Varchar* (**variable length character strings**), binary (for unstructured **blobs** of data), *Text* (for textual data) among others. It also allows **user-defined composite types** (UDTs) to be defined and used. SQL Server also makes server statistics available as virtual tables and views (called Dynamic Management Views or DMVs). A database can also contain other objects including **views**, **stored procedures**, **indexes** and **constraints**, in addition to tables, along with a transaction log. A SQL Server database can contain a maximum of 2^{31} objects, and can span multiple OS-level files with a maximum file size of 2^{20} **TB**.[17] The data in the database are stored in primary data files with an extension .mdf. Secondary data files, identified with an .ndf extension, are used to store optional metadata. Log files are identified with the .ldf extension.

Storage space allocated to a database is divided into sequentially numbered *pages*, each 8 KB in size. A *page* is the basic unit of I/O for SQL Server operations. A page is marked with a 96-byte header which stores metadata about the page including the page number, page type, free space on the page and the ID of the object that owns it. Page type defines the data contained in the page - data stored in the database, index, allocation map which holds information about how pages are allocated to tables and indexes, change map which holds information about the changes made to other pages since last backup or logging, or contain large data types such as image or text. While page is the basic unit of an I/O operation, space is actually managed in terms of an *extent* which consists of 8 pages. A database object can either span all 8 pages in an extent ("uniform extent") or share an extent with up to 7 more objects ("mixed extent"). A row in a database table cannot span more than one page, so is limited to 8 KB in size. However, if the data exceeds 8 KB and the row contains *Varchar* or *Varbinary* data, the data in those columns are moved to a new page (or possibly a sequence of pages, called an *Allocation unit*) and replaced with a pointer to the data.[For physical storage of a table, its rows are divided into a series of partitions (numbered 1 to n).

The partition size is user defined; by default all rows are in a single partition. A table is split into multiple partitions in order to spread a database over a **cluster**. Rows in each partition are stored in either **B-tree** or **heap** structure. If the table has an associated **index** to allow fast retrieval of rows, the rows are stored in-order according to their index values, with a B-tree providing the index. The data is in the leaf node of the leaves, and other nodes storing the index values for the leaf data reachable from the respective nodes. If the index is non-clustered, the rows are not sorted according to the index keys. An indexed **view** has the same storage structure as an indexed table. A table without an index is stored in an unordered heap structure. Both heaps and B-trees can span multiple allocation units.

2.3 SYSTEM ARCHITECTURE

A system architecture is primarily concerned with the internal **i**nterfaces among the system's components **or** subsystems, and the interface between the system and its external environment, especially the user. A system architecture can be contrasted with system architecture engineering, which is the method and discipline for effectively implementing the architecture of a system.It is a *method* because a sequence of steps is prescribed to produce or change the architecture of a system within a set **of** constraints**.** It is a *discipline* because a body **of** knowledge is used to inform practitioners as to the most effective way to architect the system within a set of constraints.



III. MODULES

- ✓ Patient Registration
- ✓ Attacks by External Recipient Using Anonymized Data
- ✓ Attacks by Data Providers Using Anonymized Data and Their Own Data
- ✓ Doctor Login
- ✓ Admin Login
- ✓ Cloud Server Monitoring

IV. MODULE DESCRIPTION

4.1 Patient Registration

In this module if patients have to take treatment, He/she should register their details like Name, Age, Disease, Email etc. These details are maintained in a Database by the Hospital management. Only Doctors can see all their details. Patient can only see his own record. When the data are distributed among multiple data providers or data owners, two main settings are used for anonymization. One approach is for each provider to anonymize the data independently, which results in potential loss of integrated data utility. A more desirable approach is collaborative data publishing which anonymizes data from all providers as if they would come from one source, using either a trusted third-party or Secure Multi-party Computation protocols to do computations .

4.2 Attacks By External Data Recipient Using Anonymized Data

A data recipient could be an attacker and attempts to infer additional information about the records using the published data and some background knowledge such as publicly available external data.

4.3 Attacks By Data Providers Using Anonymized Data And Their Own Data

Each data provider can also use anonymized data and his own data to infer additional information about other records. Compared to the attack by the external recipient in the first attack scenario, each provider has additional data knowledge of their own records, which can help with the attack. This issue can be further worsened when multiple data providers collude with each other.

4.4 Doctor Login

In this module Doctor can see all the patients details and will get the background knowledge(BK),by the chance he will see horizontally partitioned data of distributed data base of the group of hospitals and can see how many patients are affected without knowing of individual records of the patients and sensitive information about the individuals.

4.5 Admin Login

In this module Admin acts as Trusted Third Party(TTP).He can see all individual records and their sensitive information among the overall hospital distributed data base.Anonymation can be done by this people.He/She collected informations from various hospitals and grouped into each other and make them as an anonymised data.

4.6 Cloud Server Monitoring

In this module the cloud server will be monitering all end user details. If server found the attacker will update the detail of attacker's details with attacker id, data modified etc. Then the admin(TPA) of cloud server will indimate immediately to the server

V. CONCLUSION

We considered a new type of potential attackers in collaborative data publishing – a coalition of data providers, called *m*-adversary. To prevent privacy disclosure by any *m*-adversary we showed that guaranteeing *m*-privacy is enough. We presented heuristic algorithms exploiting equivalence group monotonicity of privacy constraints and adaptive ordering techniques for efficiently checking *m*-privacy. We introduced also a *provider-aware* anonymization algorithm with adaptive *m*-privacy checking strategies to ensure high utility and *m*-privacy of anonymized data. Our experiments confirmed that our approach achieves better or comparable utility than existing algorithms while ensuring *m*-privacy efficiently. There are many remaining research questions. Defining a proper privacy fitness score for different privacy constraints is one of them. It also remains a question to address and model the data knowledge of data providers when data are distributed in a vertical or ad-hoc fashion. It would be also interesting to verify if our methods can be adapted to other kinds of data such as set-valued data.

REFERENCES

[1] N. Li and T. Li, "t-closeness: Privacy beyond k-anonymity and diversity," in *In Proc. of IEEE 23rd Intl. Conf. on Data Engineering (ICDE)*, 2007.

[2] R. Burke, B. Mobasher, R. Zabicki, and R. Bhaumik, "Identifying attack models for secure recommendation," in *In Beyond Personalization: A Workshop on the Next Generation of Recommender Systems*, 2005.

[3] D. Kifer, "Attacks on privacy and definetti's theorem," in *Proc. of the 35th SIGMOD Intl. Conf. on Management of Data*, 2009, pp. 127–138.

[4] D. Kifer and A. Machanavajjhala, "No free lunch in data privacy," in *Proc. of the 2011 Intl. Conf. on Management of Data*, 2011, pp. 193–204.

[5] K. Lefevre, D. J. Dewitt, and R. Ramakrishnan, "Mondrian multidimensional k-anonymity," in *ICDE*, 2006.

[6] S. Goryczka, L. Xiong, and B. C. M. Fung, "m-privacy for collaborative data publishing," Emory University, Tech. Rep., 2011.

[7] X. Xiao and Y. Tao, "Anatomy: simple and effective privacy preservation," in *Proc. of the 32nd Intl. Conf. on Very Large Data Bases*, 2006,pp. 139–150.

[8] G. Cormode, D. Srivastava, N. Li, and T. Li, "Minimizing minimality and maximizing utility: analyzing method-based attacks on anonymized data," *Proc. VLDB Endow.*, vol. 3, Sept. 2010.

[9] Y. Tao, X. Xiao, J. Li, and D. Zhang, "On anti-corruption privacy preserving publication," in *Proc. of the 2008 IEEE 24th Intl. Conf. on Data Engineering*, 2008, pp. 725–734.

[10] L. Sweeney, "Datafly: A system for providing anonymity in medical data," in *Proc. of the IFIP TC11 WG11.3 Eleventh Intl. Conf. on Database Security XI: Status and Prospects*, 1998, pp. 356–381.

