An Index Structure for Private Data Outsourcing

Aaron Steele and Keith B. Frikken

Department of Computer Science and Software Engineering
Miami University, Oxford, OH 45056
steelea@muohio.edu, frikkekb@muohio.edu

Abstract. Data outsourcing provides companies a cost effective method for their data to be stored, managed, and maintained by a third-party. Data outsourcing offers many economical benefits, but also introduces several privacy concerns. Many solutions have been proposed for maintaining privacy while outsourcing data in the data as plain-text model. We propose a method that can maintain a similar level of privacy while improving upon the query performance of previous solutions. The motivating principle behind our solution is that if the data owner possesses a small amount of secure local storage, it can be used as a pseudo-index table to improve query performance for selection queries involving conjunctions. We offer a heuristic approach for calculating the required storage resources and provide experimental analysis of the scheme.

Keywords: Data Outsourcing, Privacy, Indexing

1 Introduction

For small organizations the burden of maintaining large volumes of information can be overwhelming. In an attempt to lessen the burden, organizations turn to third-parties to maintain the data for them. On one hand the economic savings from outsourcing the data is tremendous, but on the other hand concerns arise about the confidentiality of sensitive information being outsourced.

As an example, a credit card company maintains a large collection of financial information, shopping patterns, and personal information about their customers. To minimize costs, the company wants to use a third-party to host the data. The credit card company then accesses the data for billing and advertising. The drawback with such outsourcing is that the data cannot be revealed to the hosting party, for fear of compromising their customers' privacy. The scheme proposed in this paper attempts to minimize the risks of data outsourcing while improving the efficiency of data access.

Several solutions have been proposed to address this problem (Section 2.1). Several of these solutions allow the data owner to efficiently query the data using equality selections for single attributes. However, most prior solutions do not allow the data owner to efficiently query the data using conjunction. This would be most problematic if the same pair of attributes were queried frequently together in such a selection. In this paper, we introduce an indexing technique that can be applied on top of previous solutions, that facilitates answering such queries efficiently.

The remainder of the paper is structured as follows. First, we describe some recent solutions that have been introduced to solve the data outsourcing problem (Section 2). Next, we introduce our scheme that extends previous solutions by incorporating a small amount of secure local storage and a method for calculating the required amount of local storage (Section 3). We provide experimental results (Section 4) that support the flexibility and robustness of the scheme. Finally, we present some related work (Section 5) and summarize our results (Section 6).

2 Background

The scenario we consider is the same one addressed by [1, 3–6], primarily that the data to be outsourced is represented as a relational schema $R(a_1, a_2, \ldots, a_n)$ where R is a set of tuples over the set of all attributes $\{a_1, a_2, \ldots, a_n\}$ where the domain of attribute a_i is denoted by D_i. Formally, we let A be the set of all attributes and D be the set of all domains in R. We use the privacy model of confidentiality constraints that was originally introduced in [12], and subsequently has been used in [1, 3–7, 11]. Confidentiality constraints are sets of attributes within R that cannot be revealed together. The set of confidentiality constraints are written as $C = \{c_0, c_1, \ldots, c_m\}$ where $c_i \subseteq R$ for $i \in (0, m]$. Confidentiality constraints can be singletons, implying that privacy will be compromised if that specific attribute is visible. In the case of non-singleton constraints privacy is compromised through the association of the attributes, not any single attribute.

A sample relational dataset and corresponding confidentiality constraints are shown in Fig. 1. For instance, c_0, a singleton confidentiality constraint, indicates that to maintain data privacy the **SSN** must be obscured. Similarly, c_3, indicates that privacy will be violated if **Zip, Dob,** and **CardNumber** all appear in plaintext together. To satisfy this constraint, the relationship needs to be obscured, not every attribute in the tuple.

CardNumber	Name	DoB	Zip	SSN	Code
1234 5678 9012 3456	J. Johnson	03/01/85	98765	012-34-5678	135
2345 6789 0123 4567	B. Roberts	04/02/86	87654	123-45-6789	246
3456 7890 1234 5678	S. Smith	05/03/87	76543	234-56-7890	357
4567 8901 2345 6789	M. Michaels	06/04/88	65432	345-67-8901	468
5678 9012 3456 7890	A. Alexander	07/05/89	54321	456-78-9012	579

$c_0 = \{\text{SSN}\}$, $c_1 = \{\text{Name, Zip, DoB}\}$, $c_2 = \{\text{Name, CardNumber}\}$
$c_3 = \{\text{Zip, DoB, CardNumber}\}$, $c_4 = \{\text{CardNumber, Code }\}$

Fig. 1. Credit Card Relational Dataset and corresponding Confidentiality Constraints

2.1 Previous Solutions

Several solutions have presented methods to outsource a portion of the database as plaintext, which allows the client to query parts of the database efficiently. The first

method proposed for outsourcing data as plaintext was introduced in [1]. This method divides the outsourced data into two fragments that are to be stored on separate, non-communicating servers. These fragments are constructed in such a way as to satisfy the maximum number of confidentiality constraints. Any remaining unsatisfied confidentiality constraints are then satisfied either through encoding or encrypting specific attributes. The second method proposed for outsourcing data as plaintext was introduced in [4] and refined in [3]. This method is similar to the method in [1] except it doesn't require the strong assumption of non-communicating servers. This method divides the data into fragments to be stored on one or more servers. Encryption is applied at the attribute level to satisfy confidentiality constraints and prevent the linking of data between fragments. Similarly, salt is applied to each encryption to prevent frequency attacks. An example of correct fragmentation using this method can be seen in Fig. 2. The final method proposed for outsourcing data as plaintext was introduced in [6, 5]. This method, similar to the method in [1] divides the data into two fragments. The first fragment, which is maintained by the data owner, contains all sensitive attributes, or attributes in a relationship that could be used as a quasi-identifier. The second fragment contains the remaining attributes and is outsourced. A summary of methods for outsourcing data with plaintext can be found in [7, 11].

salt	enc	Name	Zip	Code
s_1	α	J.Johnson	98765	135
s_2	β	B. Roberts	87654	246
s_3	γ	S. Smith	76543	357
s_4	δ	M. Michaels	65432	468
s_5	ϵ	A. Alexander	54321	579

salt	enc	CardNumber	DoB
s_6	ζ	1234 5678 9012 3456	03/01/85
s_7	η	2345 6789 0123 4567	04/02/86
s_8	θ	3456 7890 1234 5678	05/03/87
s_9	ι	4567 8901 2345 6789	06/04/88
s_{10}	κ	5678 9012 3456 7890	07/05/89

Fig. 2. Correct fragmentation of dataset in Fig 1

2.2 Problem Definition

The solution in [6, 5] describes a scheme that relies on that the data owner to maintain a significant portion of the data. The amount of storage needed to store a single fragment only differs from storing the entire database by a constant factor. Similarly, [1] requires the assumption that the servers hosting the data lack the ability to communicate with one another. This is a strong assumption that is impossible to mandate in a real-world implementation.

A drawback of the solutions described in [3, 4], occurs when there exists a set of attributes where each attribute isn't highly selective by itself, but the conjunction of the attributes is highly selective. More formally, suppose attributes $R.A$ and $R.B$ are in different fragments and many queries of the from $\sigma_{A=p_1 \wedge B=p_2}(R)$ are issued. To answer using the method in [4, 3] one needs to obtain $\sigma_{A=p_1}(R)$ or $\sigma_{B=p_2}(R)$. The main contribution of this paper is an index-strategy that allows the system to answer the query $\sigma_{A=p_1 \wedge B=p_2}(R)$ efficiently. This technique could be used for multiple pairs of

attributes. Having such an index adds a cost to the system, but the benefits and disadvantages can be considered by the data owner, much like a DBA considers the pros and cons of adding an index to a traditional DBMS to improve performance.

3 Proposed Scheme

Our scheme is presented in the context of maintaining privacy in a relational dataset where there are two attributes; scenarios involving more than two attributes are not formally addressed and are left as potential extensions to the scheme. For our scheme we also assume that the data owner has access to a unspecified amount of secure local storage(T). Our scheme is most effective when used in conjunction with the scheme in [4, 3]. That is these schemes will be used to handle most queries, and our index technique will only be used to handle conjunctive selection queries over R. Before describing our scheme we introduce the Attribute Matrix as a method for representing the relationships between two attributes in a dataset.

Definition 1. ***(Attribute Matrix)***
An attribute matrix M for two attributes A_1 and A_2, is an mxn matrix, where $m = |D_1|$, $n = |D_2|$ and D_1 is the domain of A_1 and D_2 is the domain of A_2. Each entry in M_{ij} corresponds to the number of records in the data set where $a_1 = v_i$ and $a_2 = v_j$ ($v_i \in D_1$ and $v_j \in D_2$). Every attribute matrix has a set of row and column totals V and W s.t. $V_i = \sum M_{i*}$ and $W_j = \sum M_{*j}$.

Given an attribute matrix M for two attributes a_1 and a_2 that are stored on different fragments. We would like the server to be able to answer queries of the form $\sigma_{a_1=c_1 \wedge a_2=c_2}(R)$ for constants $c_1 \in D_1$ and $c_2 \in D_2$. In this example, suppose that the confidentiality constraints state that information about the relationship between a_1 and a_2 must be hidden. Ideally, a solution would return only $M_{c_1 c_2}$ records, but this may reveal information about the relationship between a_1 and a_2. As an example, suppose that the first attribute has a domain $\{A, B\}$, and the second attribute also has a domain $\{\epsilon, \delta\}$, and that the server knows that there are 9 A's, 6B's, 11 δ's, and 4 ϵ's (which it knows since it has a_1 in plaintext form in one table and a_2 in plaintext from in the other table). Suppose that the adversary discovered that the conjunction of $A\delta = 7$ then the adversary also discovers the remaining cells in the attribute matrix (that is there must be 2 $A\epsilon$, 4 $B\delta$, and 1 $B\epsilon$. Hence, in order for a solution to be secure against such leakage, it must not reveal anything that is not deducible from the V and W values. It is acceptable to reveal such information, because the adversary has the V and W values.

The main idea of our approach is to designate a query response size b, where if $M_{ij} < b$ then additional records will be added to pad to b. Similarly, for every $M_{ij} > b$, then b records will be outsourced and $M_{ij} - b$ records will be stored locally by the data owner. To answer a query $\sigma_{A=p_1 \wedge B=p_2}(R)$, the querier simply obtains the b cells corresponding to this entry and retrieves any values in its local storage. In the situation the data owner can select a larger b to reduce the number of records to be stored locally or select a smaller b and store more records locally. Therefore, our scheme creates an inverse relationship between query performance and required storage, thus making it a potential solution for all datasets regardless of the amount of available local storage.

3.1 Calculating Storage

The amount of local storage (T) required depends on the number of records in the dataset, the attribute distribution within these records, and the query response size (b). We assume that the server hosting the data doesn't know the attribute matrix M, but that it does know the row totals V, the column totals W, the query size requirement b, and the amount of secure local storage being used T. The goal is to determine if it is possible to outsource the data using these parameters. With this capability, the outsourcer could choose either b or T and then calculate the minimum T or b values.

A naive approach is to calculate the required storage directly by using the attribute matrix. For elements in the attribute matrix that exceed the b, store the excess locally. That is, $T = \sum_{i=1}^{m} \sum_{j=1}^{n} max\{M_{ij} - b, 0\}$. If the storage is calculated this way, then the hosting server will be able to use V, W, b, and the amount of local storage, T, to infer a large portion, if not all, of the original attribute matrix. For example, if we return to the example in the previous section and the server sees that $b = 5$ and $T = 2$, then the server is able to infer that the number of values with $A\delta$ is 7 (as this is the only such combination that requires $T = 2$ when $b = 5$.

Therefore, in order to guarantee that additional information isn't leaked to the server, we need to calculate our storage needs based on information already known to the server, primarily V, W, and b. By restricting our technique to information already known to the server we don't reveal any additional information to the server. Thus the goal is to determine if every attribute matrix M that satisfies $V$and W can be stored using T local storage and b query time. The following problem can be used to determine if such a (b, T) pair is sufficient for V and W.

Problem 1. (Optimal Storage) Given an attribute matrix row total V, attribute matrix column total W, the query response size b, and a target amount of local storage T, does there exist an attribute matrix M, such that $\sum_{i=1}^{m} \sum_{j=1}^{n} max\{M_{ij} - b, 0\} \leq T$.

Theorem 1. *The Optimal Storage Problem is NP-Hard*

Proof. (Sketch) The proof is a reduction from Subset Sum. Recall that Subset sum is: Given natural numbers $s_1, \ldots s_n$, and a target number t, is there $S' \subseteq S$ whose sum is precisely t?

The reduction can be constructed in the following way. First, we assume the existence of a black box function $H(\cdot, \cdot, \cdot, \cdot)$ that can solve the Optimal Storage problem in a polynomial amount of time. If we construct the input as follows, $V = S$, set $|W| = 2$ where $w_1 = t$ and $w_2 = \sum_{s \in S}(s - t)$, let $b = 1$ and finally set $T = \sum_{s \in S}(s) - (b \cdot m)$. If H returns 1, then this implies there is a solution to subset sum, otherwise there is not. Therefore, Optimal Storage must be NP-HARD. □

3.2 Approximating Storage

Since determining the necessary storage, based upon V and W is NP-Hard, we propose an approximation($APRX$) that determines the storage requirement for a dataset. The approximation is based upon a few basic observations. The first observations is, $\forall_i$

where $V_i \leq b$ no local storage will be required for this row of the attribute matrix. The rationale behind this is that if a single row total is less than or equal to b, then it is impossible for any entry in the row to be greater than b. Therefore, when approximating the storage, we discard all rows whose total is not greater than b. Thus, without loss of generality, we assume that each V_i and W_j are larger than b.

The second observation is when $V_i > b$, then the most that an individual row can contribute to the local storage is $V_i - b$, because the worst case would be that all values are placed in a single cell. It is straightforward to see this observations holds true when considering columns as well as rows. Using these two observations we then formulate our approximation algorithm as follows: $APRX(V, W, b) = (\sum_{i=0}^{m} V_i) - max\{m, n\}b$.

We offer an additional method of approximating the required local storage. This method is similar to the first, but requires that we reveal a value u to the local server, with the claim that there does not exist a value in the attribute matrix that exceeds u. This reveals some additional information to the adversary, but this leakage may be acceptable in some circumstances. Since the adversary knows that there does not exist an entry in the matrix that exceeds u, he can discover a minimum of how many non-zero entries exist in a particular row or column of the attribute matrix. This information is the crux of our new approximation algorithm. $APRX'(V, W, b, u) = \sum_{i=0}^{m} V_i - max\{1, \lfloor V_i/u \rfloor\}b$

By divulging more information to the adversary we can help minimize the amount of wasted storage space. It is apparent that $u \in [max\{A_{ij}\}, max\{V \cup W\}]$. If we let $u < max\{A_{ij}\}$ then the amount of storage will be insufficient because there exists an entry in the attribute matrix that exceeds u. Thus, by using $APRX'$ the data owner can achieve an adequate balance between privacy and required storage.

Theorem 2. *The storage required for $APRX \geq OPT$, where OPT is the maximum storage required for any attribute matrix that can produce the row and column totals V, W.*

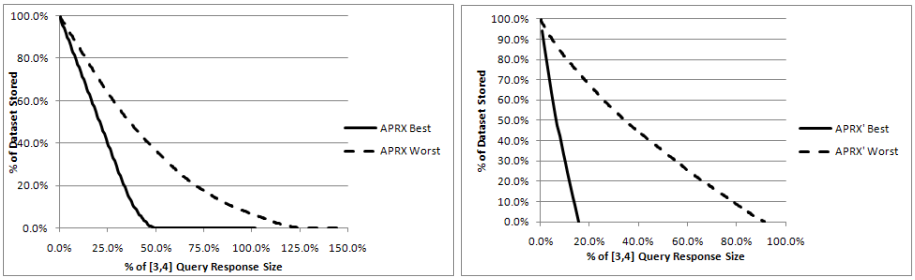
Proof. (Sketch)

First, we are given the row and column totals V, W for an attribute matrix and a query response size b. It is known that all row and column totals must be greater than b otherwise that row or column will not contribute to the required storage. It is also obvious that each row and column must have at least one non-zero entry in it. Therefore, it can be inferred that at least b elements from each row or column can be outsourced, thus leaving the remaining data to be stored locally. Since this requirement holds for both rows and columns, we can determine the minimum amount to be outsourced based upon whether there are more rows or columns. Thus, the data to be stored must be less than the the entire dataset minus b times the maximum of m and n. The proof that $APRX' \geq OPT$ follows a similar construction. □

4 Experiments & Results

The approximation algorithms presented in Section 3.2 have been implemented to obtain experimental data to assess the quality of the algorithms in terms of the accuracy of the calculated storage and query response size. For both $APRX$ and $APRX'$ we

generated 100,000 random attribute matrices. For every possible b value we calculated the percentage of the dataset that would have to be stored locally based on our approximation as well as the average query response size as a percentage of the average query response size for the same matrix using the methods in [4, 3]. Figure 4 shows the matrix that produced the best and worst results for each approximation.



$(APRX)$ $(APRX')$

Fig. 3. Experimental Storage and Query Response Size

5 Related Work

The data outsourcing paradigm has been the focus of a significant amount of research. The work in [2, 8, 9, 13] propose encrypting the data entirely and propose novel methods for efficient query evaluation. The encrypted data usually is accompanied by an index of the data to aid in query evaluation. Similar to all these methods is the solution in [14], where the authors propose a method for allowing people to query statistical information from the data, while protecting the data itself. Work done by the authors in [10] claim that it is infeasible to traditionally query encrypted data efficiently while maintaining security. The authors propose a system that meets a revised definition of security and an efficient query processing method.

6 Conclusions

This paper presented an approach to improve query performance when outsourcing data in the data as plaintext model. Specifically we were interested in improving the performance of queries where the attributes in the query are not highly selective independently, but the conjunction of the attributes is highly selective. The solution presented achieves the desired query performance by leveraging the data owner's secure local storage. The amount of storage required is independent of the dataset and is determined by the data owner based upon desired query performance and availability of storage. In this paper we offer two approximation algorithms for calculating the required storage for a given dataset. We also provide experimental results showing that the approximations we offer are a relatively good estimate for the actual amount of storage.

Acknowledgments

The authors would like to thank the anonymous reviewers for their comments and useful suggestions. Portions of this work were supported by Grant CNS-0915843 from the National Science Foundation.

References

1. G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *In Proc. CIDR*, 2005.
2. A. Cesell, E. Damiani, S. De Capitani Di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM Trans. Inf. Syst. Secur.*, 8:119–152, February 2005.
3. V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation design for efficient query execution over sensitive distributed databases. In *Distributed Computing Systems, 2009. ICDCS '09. 29th IEEE International Conference on*, pages 32 –39, 22-26 2009.
4. V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Combining fragmentation and encryption to protect privacy in data storage. In *ACM TISSEC*, 2010.
5. V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Keep a few: Outsourcing data while maintaining confidentiality. In Michael Backes and Peng Ning, editors, *Computer Security ESORICS 2009*, volume 5789 of *Lecture Notes in Computer Science*, pages 440–455. Springer Berlin / Heidelberg, 2010.
6. V. Ciriani, S. Capitani Di Vimercati, S. Foresti, S. Jajodial, S. Paraboschi, and P. Samarati. Enforcing confidentiality constraints on sensitive databases with lightweight trusted clients. In *Proceedings of the 23rd Annual IFIP WG 11.3 Working Conference on Data and Applications Security XXIII*, pages 225–239, Berlin, Heidelberg, 2009. Springer-Verlag.
7. S. De Capitani di Vimercati and S. Foresti. Privacy of outsourced data. In *Privacy and Identity Management for Life*, volume 320 of *IFIP Advances in Information and Communication Technology*, pages 174–187. Springer Boston, 2010.
8. B. Iyer H. Hacigümüş and S. Mehrotra. Providing database as a service. In *In Proc. of ICDE*, 2002.
9. H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 216–227, New York, NY, USA, 2002. ACM.
10. M. Kantarcioglu and C. Clifton. Security issues in querying encrypted data. Technical Report TR-04-013, Purdue University, 2004.
11. P. Samarati and S. De Capitani di Vimercati. Data protection in outsourcing scenarios: Issues and directions. In *ASIACCS '10: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 1–14, New York, NY, USA, 2010. ACM.
12. R. Agrawaland R. Srikant. Privacy-preserving data mining. *SIGMOD Rec.*, 29:439–450, May 2000.
13. H. Wang and L. Lakshmanan. Efficient secure query evaluation over encrypted xml databases. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 127–138. VLDB Endowment, 2006.
14. L. Xiong, S Chitti, and L. Liu. Preserving data privacy in outsourcing data aggregation services. *ACM Trans. Internet Technol.*, 7(3):17, 2007.