Security of Social Information from Query Analysis in DaaS

Junpei Kawamoto
Graduate School of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku
Kyoto 606-8501, Japan
j.kawamoto@db.soc.i.kyoto-u.ac.jp

Masatoshi Yoshikawa
Graduate School of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku
Kyoto 606-8501, Japan
yoshikawa@i.kyoto-u.ac.jp

ABSTRACT

Databases as a Service (DaaS) are recently in the spotlight as an essential component of the cloud computing framework. We can more easily develop applications to facilitate users' collaboration than without DaaS. Unfortunately, DaaS brings a new risk of data compromise because most data are currently stored and managed by the service provider. Securing users' information, the data are generally encrypted at a trusted client. Although encryption ensures that no one can sneak a look at the data, malicious users still have opportunities to take users' relational information. The relational information, which we name social information as opposed to personal information, has not been adequately examined, but it is becoming important because of improvement of social analysis. As described herein, we first introduce an attack model of obtaining social information from analyses of query logs in DaaS; then we provide a solution to the problem. Our method converts some different queries in the same query, so that malicious attackers cannnot analyze the query logs. We also define an overhead cost of the translation and propose a method that can optimize the conversion using extensible hashing.

1. INTRODUCTION

Database as a Service (DaaS) is a component of the cloud computing framework and it is attracting someone's attention. Nowadays major companies launch DaaS and many innovative applications are using it. In the applications, most of data are stored into servers and managed by service providers. Users use them through the web. Therefore if only they have a connection to the web, they can utilize the applications whenever and wherever they are. Additionaly DaaS has another feature: users can share their data and easily collaborate with all over the world. The possession enables to develop many new collaborative applications. However DaaS has a negative property. It brings down a risk of compromise. The risk from DaaS is more crucial than one from traditional database systems because we cannot directly control their data. Actually some cases such as a server is taken over by attacker and an operator of a server opens confidential by mistake are reported. Preventing the compromise risk and providing safe DaaS are essential challenge.

To guarantee the safety, there are three kinds of studies for this purpose; 1) security of data stored in servers; 2) preventing guess of data from query analysis; 3) protecting personal information of users such as interest or favorite from query analysis. In the study of security of data, data are encrypted at trusted clients before sending to the server. Therefore even server cannot know what data are stored but the server also cannot process queries. How to process queries on the encrypted data is a main topic of this research. For solving the problem, current encrypted data have some indices for query processing [5, 4]. Another topic of the study of security of data stored in servers is how to share data stored in encrypted database. This topic is how to share the encryption key between the authorized users. Damiani et al. introduced a method of dynamically deriving the encryption keys and reducing the number of keys each user has to manage [1]. The second group of the studies; preventing guess of data from query analysis is well known for the problem of k-anonimity [8]. There are many kinds of specific problems and solutions for them. There is a demand of requesting data but keeping how data is requested secret for servers. This demand is called as Private Information Retrieval (PIR) [3] and PIR is the third category of the studies.

In this paper, we add a new problem to the studies. It is a problem of protecting social information. Social information, which we think as opposite idea of personal information identifying each person, has not been discussed yet. However, many researches about social network from web services are nowadays studied. Discussing security of social information as privacy issue is therefore becoming significant. We are not identified by only social information but organizational size, how organization work, our role in organizations may be leaked. We would not generally like to take all association public therefore we must have a right of deciding public or secret about social information as same as personal information. Thinking about a scenario of using DaaS, this service is used by not only individual but also enterprise. In such case, organization structure is important confidential.

We firstly introduce an attack model of social information.

For setting the model, we focus attention on query log and presupposed users who send same and discriminative queries have high relativity. Secondly, we discuss a method of query alteration to conceal social information from the attack model. In our proposal approach, some different queries are rewritten to a same query using query transfer tree based on the extendible hashing [2]. Finally we set a cost function and use a property of extendible hashing, that is the distribution of the number of hash value stored in each leaf is flat, to optimize the cost. Our contributions of this paper are summarized as follows: 1) We raise a new security problem about social information; 2) We introduce a concrete algorithm of analyzing social information from query logs in the DaaS scenario; 3) We propose a method of query conversion to protect social information from query analysis.

2. ATTACK MODEL

Malicious attackers can obtain social information by analyzing query literals and their frequency. The query literal is a term in the query such as "Nov. 15, 2008" in the following query: $date =$ "Nov. 15, 2008". In this chapter, we model a type of data monitored by attackers and introduce a method of computing users' similarity from the data.

We consider an application using DaaS. Let U be the set of users who send requests to the server, and let L be the set of literals received by the server. An attacker peeping requests can count how many literal $l \in L$ are sent by the user $u \in U$: the attacker has a logging function of

$$\mathrm{Log} : U \times L \longrightarrow Z. \tag{1}$$

We next introduce a method of calculating similarities from the observed data using a query feature vector for a method of extracting social information. Analysis of user similarity using a query feature vector presupposes that users who sent the same request have a relation. For example, users who ask an event at a particular date and place have an interest: they have a relation in this meaning. Analyses of user similarity calculate frequencies for how a user pair sends the same query and extracts a user pair to which they have high similarity. The query feature vector of a user is calculated using the frequency of the literals in requests from that user. The attacker observes how many users u sent a literal l as $\mathrm{Log}(u, l)$ described above. The user frequency of the literal l: uf_l, which is how many users sent the literal, is calculated using the following formulation.

$$uf_l = \sum_{u \in U} \begin{cases} 1; \text{ if } \mathrm{Log}(u, l) \neq 0 \\ 0; \text{ otherwise} \end{cases}$$

For the weight of a user u and a literal l; $w_{u,l}$ is defined as

$$w_{u,l} = \frac{\mathrm{Log}(u, l)}{uf_l}.$$

We defined the query feature vector of user u: QV_u. The feature vector comprises the weights as elements.

In this paper, we define the similarity of user u and v as $\mathrm{sim}(u, v)$, by the cosine degree, as

$$\mathrm{sim}(u, v) = \frac{QV_u \cdot QV_v}{|QV_u||QV_v|}.$$

Attackers set a threshold θ and assess a user pair that has a similarity that is greater than the threshold as related users.

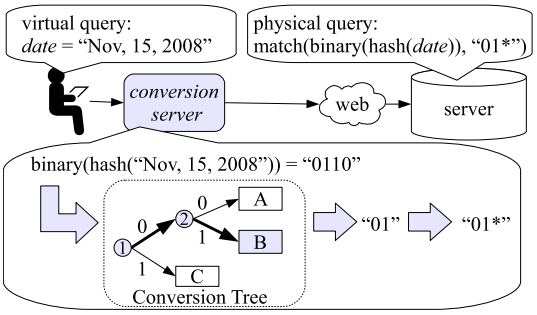


Figure 1: Sketch of query conversion.

3. QUERY CONVERSION

We start by introducing a sketch of our approach before describing intimate algorithms related to query conversion. Figure 1 depicts the basic flow. As an example, we consider a schedule management application. On the application, we attempt to convert the following query: $date =$ "Nov. 15, 2008".

We use a trusted intermediate server for this purpose and it works according to the following process. It first converts the literal "Nov. 15, 2008" into a binary string. Where hash is a hashing function and binary is a function that converts the input string into a binary string, this step is described as

$$\mathrm{binary}(\mathrm{hash}(\text{"Nov. 15, 2008"})) = \text{"0110"}.$$

Then, the conversion server translates the binary string using conversion tree. That is a binary tree; it maps each binary string to the leaf. The leaf is a binary string that is mapped is decided from comparing the string to the label that each edge has. For example, we consider the conversion tree shown in Fig. 1 and convert binary string "0110". We start from the root node (number 1). The first character of the target string is "0". Therefore, we choose the upper edge labeled "0" and go to node number 2. Because the second character is "1", we choose the lower edge and go to the leaf B. Therefore, we decide the leaf mapped by each binary string. After deciding, we concatenate labels of edges from the root to the selected leaf. In this example, the connected label is "01". Next, we check the length of the connected label. The wild-card character "*" is appended to the connected label if the length is shorter than the original binary string. Consequently, the index literal "Nov. 15, 2008" is converted to "01 $*$ ". This literal "01 $*$ " means to require items started with "01", so that we can bind up many queries such as those starting with "01". Therefore, attackers cannot distinguish what the users really want. The query sent to the server, therefore, is

$$\mathrm{match}(\mathrm{binary}(\mathrm{hash}(date)), \text{"01} * \text{"}),$$

where match is the function comparing a binary string and converted strings. The server can use these functions.

Query conversion converts different queries into the same new query. For that reason, the result of converted queries might have many irrelevant items. Therefore, the intermediate server execute an original query to results so that unnecessary items are removed, and users and applications can obtain items as desired.

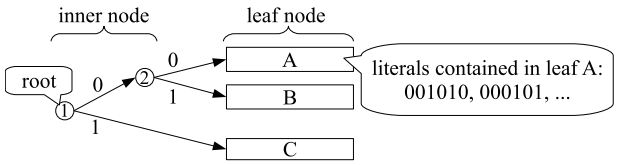


Figure 2: Query conversion tree.

Algorithm 1 convert($Tree, user, literal$)

```
path ⇐ nil, node ⇐ root of Tree
increment Count(user, literal) of Tree
update(Tree)
while node is not leaf node do
  for each child of node do
    e ⇐ label of node to child
    if literal[depth of child] = e then
      path ⇐ path + e, node ⇐ child
      break for-loop
    end if
  end for
end while
if length(literal) ≠ length(path) then
  path ⇐ path + " * "
end if
return path
```

In the rest of this chapter, we firstly define the conversion tree and then we explain our algorithm of the conversion.

3.1 Structure of Conversion Tree

Our conversion tree T is based on extensible hash and Fig. 2 presents a sample of it. We formally define it as

$$T(root, V, L, \text{Counter}).$$

Therein, V is a set of nodes and $root \in V$ is the root node of tree T. Our tree has nodes of two kinds: inner nodes and leaf nodes. The inner node (shown as a circle in Fig. 2) has a set of child nodes. The leaf node (shown as a box in Fig. 2) has a set of literals mapped to the leaf node. Edges connecting a node to another node have a label, where L is a set of literals converted this tree and Counter is a counter that counts the requests of every user. We use numbers recorded by the counter as a function. Therefore, let $\text{Count}(u, l)$ represent the number of users $u \in U$ who request literal $l \in L$.

3.2 Query Conversion Algorithm

When the conversion server receives a query: $attr = l$, it translates the literal l in the query into the binary string b: $b = \text{binary}(\text{hash}(l))$, where hash is a hashing function and binary is a function converting an input to the binary string. The conversion server processes the binary string by steps shown in Algorithm 1. First, the counter of a requesting user u and the literal l ($Counter(u, l)$) is incremented by one, and the conversion tree is updated. This updating process is explained in the next chapter. Then, a leaf node mapped by the binary string b is computed by pursuing the conversion tree. After deciding the mapped leaf, labels set in the edges from the root node to the leaf node are connected. A wild-card character "*" is appended to the connected labels if the length of the connected labels does not equal the length of the binary string. Finally, we obtain the connected labels as a converted string b'. The request sent to the server uses functions binary, hash, and match; it is written as follows

$$\text{match}(\text{binary}(\text{hash}(attr)), b').$$

If a user's request has more than two literals, for example requesting from a date and a place, we use as many conversion trees as the literals contained in the request.

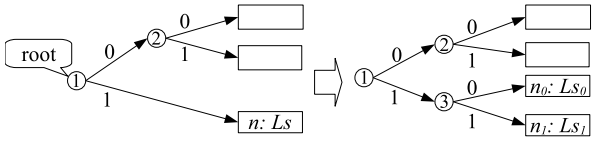


Figure 3: Restricting query conversion tree.

3.3 Update of Conversion Tree

Our proposed approach binds some queries and converts another query. Consequently, we prevent attackers from inferring the social information from observed query logs. However, binding some queries causes users to obtain items they did not ask for, which means that the cost increases. We restrict the conversion tree dynamically to keep the cost under a given maximum cost.

The conversion tree is updated when a user converts a query. We imagine that a user sends a query and that the query is mapped to a leaf node n; and we let the depth of node n be d. In this condition, the literals Ls in the leaf node n are first divided into two subsets Ls_0 and Ls_1, literals for which the d-th character is 0 and 1 respectively belong to Ls_0 and Ls_1. Then, Ls_0 and Ls_1 are examined as to whether they should be divided or not. The tree removes the leaf nodes n and adds two leaf nodes n_0 and n_1, which respectively have the set of literals Ls_0 and Ls_1 if the subsets are divided by the judgment. Figure 3 portrays this update process. The old leaf node n containing the set of literals Ls is changed to an inner node to connect new leaf nodes; also, new leaf nodes n_0 and n_1 are added and connected to the inner node. The new edges from the inner node to n_0 and n_1 respectively bear labels 0 and 1.

Determination of the divide is done according to the following steps considering costs. The costs $cost_0$ and $cost_1$ are first calculated, respectively, for the subsets Ls_0 and Ls_1:

$$cost_i = \sum_{u \in U} \frac{\sum_{l \in Ls_i} \text{Count}(u, l) \times N_l}{total_u} \ (i = 0, 1).$$

The leaf node n are divided into the two leaf nodes n_0 and n_1 if at least one of the costs $cost_0$ and $cost_1$ are bigger than a given max allowable cost c. The following conditions are computed for each subset if the costs are less than the c.

$$\sum_{l \in Ls_i} \left| \frac{\text{Count}(u, l)}{total_u} - \frac{\text{Count}(v, l)}{total_v} \right| < k \ (\forall u, v \in U, i = 0, 1)$$

When the condition presented above is true for L_0 and L_1, then the leaf nodes n are divided into the two leaf nodes n_0 and n_1. The condition means a summation for every literal of a difference of rates for how many users u and v sent a literal l. That it is so small is an expression of a

difference of rates; the number of users u and v who sent literals mapped in the nodes n_0 and n_1 is small, which is a different tendency from that by which requests observed by attackers are small. Consequently, nodes are divided if the intractability of analyzing similarity can remain high, even if the cost is not great.

For this determination, a conversion server must record the number of items N_l matching a request l. The server therefore must examine insert and delete events for the databases.

4. EVALUATION EXPERIMENT

4.1 Dataset used for Experiments

We have an experiment to evaluate our proposal. For this experiment, we need a dataset that has users belonging to some groups and queries by the users to DaaS. Unfortunately, such public datasets are apparently unavailable. Therefore we choose an open dataset which is used by Alexander et al. for evaluating query processing on a P2P network [6]. This dataset is based on the OpenDirectory [7]. OpenDirectory is a project for classifying web sites; it has hierarchical categories. Each category has more segmentalized categories and web sites which are categorized in the category. Each web site might belong to more than two categories. Meaning that the editor of a web site belongs to some category group. In our experiment, we use web sites in OpenDirectory as users and categories as groups as same as the experiment by Alexander et al. That is the dataset is reflect a real community: people might belong to some interest group.

In this experiment, we use categories for which the depth is less than five and use web sites belonging to the categories. Then we create 100,000 queries on the users and groups. Our configuration is that each user must request web sites belonging to a group similarly to a user who would use the web site. The number of queries depends on the Zipf distribution, which is often used in studies of social networks [9]. We calculate the similarity over the converted queries for the dataset; the dataset has 133,602 users and 6,280 groups.

4.2 Result

Next, we explain the results. We use the precision and recall for measurement of success of attacks. The lower precision and recall therefore indicate higher safety. The x-axes of cost graphs show cost; the y-axes show the numbers of users. They therefore express that 100 users have 5 cost points if a point stands for $(5, 100)$. The cost value of user u is computed using the following formula;

$$cost_u = \frac{\sum_{l \in L} \mathrm{Log}(u, l) \times N_l}{toral_u},$$

where N_l be the number of items received from the server as a result of a request literal l. That is the formulation expresses an average of the number of items user u receive from the server.

Fig. 4 portrays the capability of our attack model, which is the precision and recall for unconverted queries. Attackers decide that users belong to the same group if the similarity is greater than threshold θ. Consequently, our attack method indicates high precision for any threshold. On the other hand, recall is about 0.5, indicating the need for some improvement.

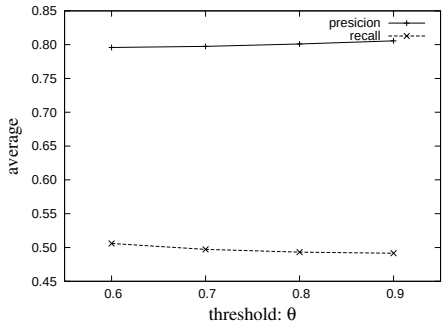
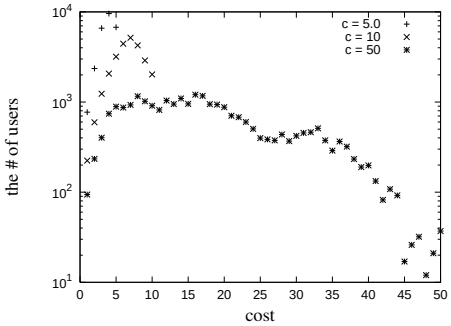
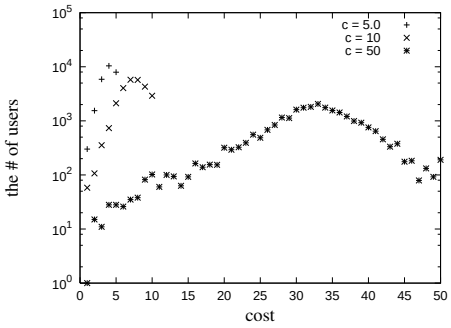


Figure 4: Result of similarity from the feature vectors.



(a) $k = 10$



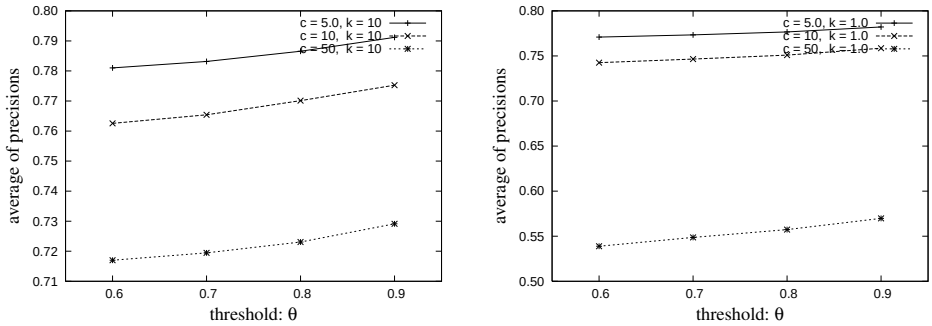
(b) $k = 1.0$

Figure 5: Reception cost of data for each users.

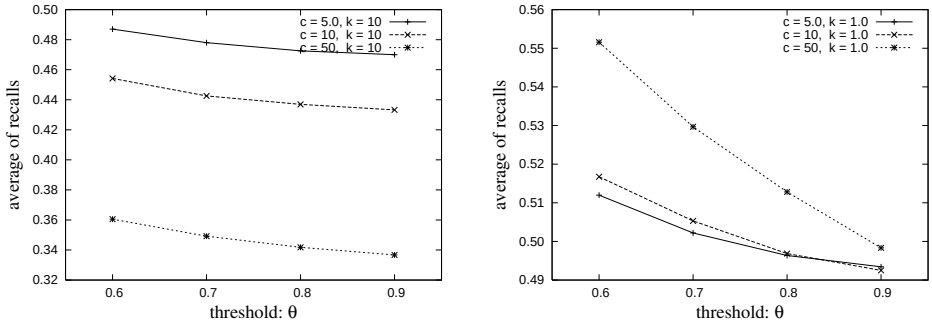
Fig. 5 and 6 depicts the result about our approach. Fig. 5 shows the distribution of the costs; each user can execute queries under the maximum admissible cost for all cases. The value c in the graphs of Fig. 6 is the maximum admissible cost described in section 3.3. Speaking from the graphs, the condition $c = 50, k = 1.0$ conducts the safest result. Each user in this case downloads, at most, 50 items a query; the average of the precisions is less than 0.6. The average of the recalls in this case is about 0.5.

5. CONCLUSION AND FUTURE WORK

As described herein, we introduce a new privacy problem— security of social information— to produce safer DaaS than that which services already provide. Social information is information about relation between users: it differs from



(a) Precision ($k = 10$) (b) Precision ($k = 1.0$)



(c) Recall ($k = 10$) (d) Recall ($k = 1.0$)

Figure 6: Result of similarity using query conversion.

personal information because personal information is information identifying individual users. Therefore, it has been treated as unimportant. Nevertheless, many methods of analyzing social networks are studied so that the relation information can present risk. We believe that discussing social information is important to retain privacy.

Based on these situations, we introduce an attack method of exposing social information from analyzing queries. We propose a method to defend the attack by rewriting the queries. Our proposal has two parameters related to safety and cost; users can choose them according to their demand. Moreover, we evaluate our proposal using an experiment based on real information: an open directory. Based on the result, a specific pair of parameters yields high security and low overhead. However, our method must incur large costs depending on circumstances: more improvement is necessary.

We envision our future work as improvement of the attack method and query conversion to protect privacy from stronger attacks.

6. ACKNOWLEDGMENTS

This research was supported in part by Kyoto University Global COE Program "Informatics Education and Research Center for Knowledge Circulating Society", and in part by "Data Integration and Analysis System" funded by the National Key Technology, Ministry of Education, Culture, Sports, Science and Technology, Japan.

7. REFERENCES

[1] E. Damiani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Selective data encryption in outsourced dynamic environments. Electronic Notes in Theoretical Computer Science, 168:127–142, 2007.

[2] R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong. Extendible hashing—a fast access method for dynamic files. ACM Transactions on Database Systems, 4(3):315–344, 1979.

[3] W. Gasarch. A survey on private information retrieval. Bulletin of the EATCS, 82:72–107, 2004.

[4] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In Proceedings of the Thirtieth international conference on Very large data bases, pages 720–730, 2004.

[5] B. Iyer, S. Mehrotra, E. Mykletun, G. Tsudik, and Y. Wu. A framework for efficient storage security in rdbms. In LNCS 2992, pages 147–164. Springer-Verlag Berlin Heidelberg, 2004.

[6] A. Löser, S. Staab, and C. Tempich. Semantic methods for p2p query routing. In Multiagent System Technologies, volume 3550 of Lecture Notes in Computer Science, pages 15–26. Springer Berlin, 2005.

[7] OpenDirectory. http://www.dmoz.org/.

[8] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, In Proceedings of the IEEE Symposium on Research in Security and Privacy, 1998.

[9] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In Multimedia Computing and Networking.