An SVM-based Query Monitoring Method For Inference Control

Yikan Chen
University of Virginia
yc2r@virginia.edu

David Evans
University of Virginia
evans@virginia.edu

***Abstract*—Many widely-used data sharing applications rely on allowing untrusted clients to query against secret data. Inference control for secure data is a key issue. In these scenarios, the owner of secret data opens the secret data for querying to provide general information while trying to ensure not too much information is leaked from these queries. In this work, we propose a method based on Support Vector Machines (SVM) to quickly determine whether a set of queries is leaking too much information. Through experiments over several sample functions, we demonstrate the performance and flexibility of this method.**

***Index Terms*—Inference Control, Support Vector Machines**

I. Introduction

Access control protocols are commonly used to protect secure data. However, a malicious user may send a series of seemingly innocent queries and infer much information of the secure data from previous replies. The need for this kind of *inference control* is explored in many database-related researches [6], [3], [5]. In this work, we discuss this problem for the case when the secret data is one entry. For example, a pharmacist needs to perform a drug interaction check over a patient. What she needs to know is only certain part of the patient's history such as if the patient is allergic to certain drug combinations. Unlike the patient's doctor, the pharmacist should not know the detail sensitive health information. Similar situation happens to personal recommendation systems such as on-line shopping websites and music players. Privacy is always a concern for these systems and there are also papers [1] about sensitive information leakage by these recommendation systems.

We model the process of gathering necessary personal information for certain purposes as a set of querying processes. The query client can perform the *inference attack* by analyzing a set of query replies. The objective of the data owner is to ensure that not too much information is leaked while still provide necessary information.

Secure computation is considered as a powerful tool to handle applications with privacy concerns. Secure computation guarantees that each party only knows the result and its own inputs after the computation. However, in the previous scenarios, after several queries, the querier can still infer a lot of information even if each query-reply process is done by secure computation. So, a method that can be embedded into a secure computation protocol is needed here if both each party's inputs should be protected over several computations.

II. Approaches and Experiment Results

In our model, the information leakage is defined by the number of remaining possible private values that are consistent with all previous query results. This is a very straightforward measurement: if after applying exclusive rules which investigate query results, there are still many possibilities for the secret, we can say the private data remains mostly undetermined after these queries.

Apparently it is too complicated to compute information leakage by exhaustively testing all remaining possibilities of the secret. There is a previous work [4] trying to simplify this computation by approximation. Although they achieves polynomial complexity for string secrets, still, it suffers for two reasons. First, polynomial complexity is still too complex to be implemented in secure computation protocols. Moreover, the method only works for a certain class of functions. Most approaches in the database area mentioned before are either too complicated for secure computation or too restricted to database applications. Raymond et al. [6] use inference rules to detect information leakage. However, some of their operations require exponential time. Yu et al. [3] use a Bayesian network to evaluate the inference probability in a database. Sastry et al. [5] propose an aggregation graph to represent inference. Both of these two papers are based on exploring the structure of a database and data dependencies.

In this work, we notice that the owner does not care much about how much information is leaked. Generally, the owner has an idea of a self-defined threshold about the upper limit of the information leakage. What we need is just a classifier. There are two classes for incoming queries: "leaking too much information." or "it's okay." The classifier will determine which class the incoming queries belong to. Using a classifier has two advantages. First, the most complicated training process can be separated from the classifying process which is much simpler. Since the classifier can be trained before actual queries come, we can train the classifier extensively with well-prepared samples and these training samples can be pre-computed. The classifying algorithm is normally quite simple. In this work, the SVM classification can be done by a vector multiplication and one comparison. Second, the classifier can be applied to many secret data formats and query functions efficiently. The validity of the classifier lies in that very similar queries over identical secret leaks very similar

amount of information, which is true or almost true for most cases.

We use the Support Vector Machines (SVM), a quite popular supervised machine learning algorithm in this work. We test the performance of SVM-based classification with preliminary experiments when the secret is a string and the query function is Hamming or Levenshtein distances. We use LibSVM [2] as our SVM tool. The performance of the SVM is determined by its kernel function, which maps the original data to a higher dimension. One kernel function we use is the classic RBF kernel, defined as:

$K(\mathbf{x},\mathbf{y}) = Ce^{-(\gamma(||\mathbf{x}-\mathbf{y}||^2))}$

where $\mathbf{x}$,$\mathbf{y}$ are two data points and C, γ are parameters. The key question is how to define the distance between two data points. By customizing the distance function based on actual applications, we can construct an SVM that is best fit to the problem. For example, in our work when the secret is a string and the query function is Hamming distance represented as $H(x,y)$, a data point is a certain set of queries. In the case where the number of queries is 1, we define the distance function between two sets of queries $\mathbf{x}$ and $\mathbf{y}$ as:

$||\mathbf{x}-\mathbf{y}|| = \mathtt{min}(H(\mathbf{x},\mathbf{y}), N - H(\mathbf{x},\mathbf{y}))$

where N is the length of the query sequence. We observe that querying with a certain binary sequence X leaks the same amount of information as the querying with the opposite of X. So, two query sequences are considered similar if they are almost identical or almost opposite in the distance function above. When there are multiple queries, we define the distance function as a match of queries that has the minimal sum.

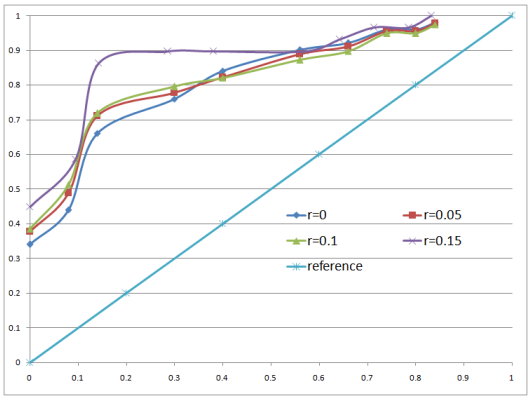

Fig. 1. ROC curve for 24 bit binary secret sequence/2 queries over Hamming distance

A few results of our experiments are shown in Figure 1 and Figure 2. The classification performance is depicted with ROC curves. In an ROC curve, the x-axis and y-axis represent the False Positive Rate (FPR) and True Positive Rate (TPR), respectively. In Figure 1, we use 100 positive and 100 negative samples from 1000 random query sets as training samples. In Figure 2, we use 100 positive and 100 negative samples from 500 random query sets as training samples. The test sample size is 100.

Additionally, we introduce a new parameter r. In our current experiments, the SVM is trained by extreme samples, i.e.

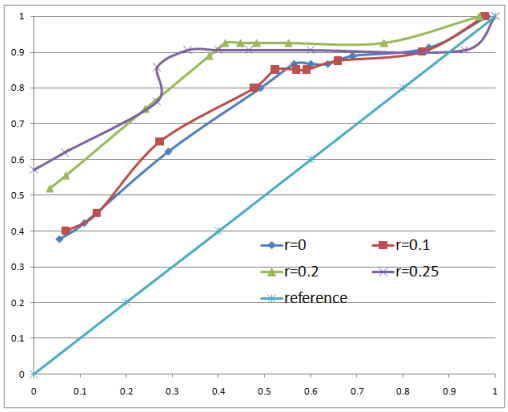

Fig. 2. ROC curve for 10 bit genome sequence/2 queries over Levenshtein distance

queries that will leak almost all or no information. So what we want to confirm is that it is able to detect very dangerous queries and very safe queries. For queries leaks a moderate amount of information, the classification performance is not as good (shown with ROC curves when $r = 0$). The parameter r represents the threshold to determine whether the incoming query set is too close to the average information leakage. If the information leakage for an incoming query set locates within the interval defined by the percentage r around the median, the decision made by the SVM will not be considered.

From these images, we can observe that generally the SVM with 200-400 samples can detect more than 90% dangerous queries with less than 30% false positive rate ($<$20 % in most cases). Note that performances still have huge improving space since the secret owner can easily prepare more samples to ensure their data safety in a practical application.

III. CONCLUSION

Inference control is necessary for multiple query process over single secret data. We introduce an SVM-based learning and classifying approach that can quickly determine whether the query is about to leak too much information. Experiments show that it can achieve high classification performance over different query functions.

REFERENCES

[1] J.A. Calandrino, A. Kilzer, A. Narayanan, E.W. Felten, and V. Shmatikov. You might also like: Privacy risks of collaborative filtering. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 231–246. IEEE, 2011.

[2] C.C. Chang and C.J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[3] Y. Chen and W. Chu. Database security protection via inference detection. *Intelligence and Security Informatics*, pages 452–458, 2006.

[4] Y. Chen and D. Evans. Auditing information leakage for distance metrics. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust (PASSAT)*, pages 1131–1140. IEEE, 2011.

[5] S. Konduri, B. Panda, and W.N. Li. Monitoring information leakage during query aggregation. In *The 4th International Conference on Distributed computing and Internet Technology*, pages 89–96. Springer-Verlag, 2007.

[6] R.W. Yip and EN Levitt. Data level inference detection in database systems. In *11th IEEE Computer Security Foundations Workshop*, pages 179–189. IEEE, 1998.