A Machine-Learning-Based Approach to Suppress Volume Leakage
Project Midterm Milestone for Class CS 507

Jiaming Yuan

Friday 16th February, 2024

Abstract

Searchable Encryption (SE) enables secure keyword queries on cloud-stored documents while preserving user privacy. However, adversaries exploiting leakages and auxiliary knowledge pose a threat known as Leakage Abuse Attack (LAA). A prevalent type of leakage, known as volume leakage, encompasses both query volume (indicating the number of documents related to a keyword) and document volume (indicating the number of keywords related to a document).

Under the consideration of the complexity of effectively suppressing these two forms of volume leakage, this project aims to develop a machine-learning-based approach. This innovative strategy seeks to mitigate volume leakage in searchable encryption, offering an advanced and efficient solution with k-indistinguishability.

Notably, this project is an individual undertaking for the CS 507 course, providing a focused exploration into the realm of privacy preservation in cloud-based searches.

1 Background

Searchable Encryption (SE) has been extensively investigated for enabling users to query keywords on a set of documents stored in a cloud platform while preserving privacy. This is achieved by outsourcing encrypted documents and associated keywords to the cloud, allowing users to search for keywords by sending tokens. However, adversaries, although unable to directly access information from encrypted documents, encrypted keywords, and tokens, can infer sensitive data through various leakages and auxiliary knowledge. This form of attack is known as Leakage Abuse Attack (LAA). To thwart such attacks, it is crucial to eliminate utilized leakages from a searchable encryption scheme, a process known as leakage suppression.

The primary challenge of leakage suppression is to mitigate leakages with limited storage overhead. For instance, padding dummy messages and documents to create only one leakage pattern for each type of leakage can result in significant storage overhead. This presents a trade-off between storage overhead and the number of leakage patterns. In our prior research, we introduced k-indistinguishability to address this, ensuring there are at least k records for each pattern of each type of leakage. For instance, if $k = 10$ for document volume leakage, there are at least 10 documents with a volume of 6.

Another challenge involves suppressing multiple types of leakages simultaneously. For instance, to hide volume leakage, both document volume and query volume leakages must be concealed simultaneously. Additionally, LAAs may leverage multiple types of leakage, including volume leakage and co-occurrence leakage.

To tackle these challenges, we proposed a machine-learning-based approach to hide leakages with k-indistinguishability. In this project, our focus is on hiding volume leakage, as it is the most frequently targeted by LAAs.

2 Problem Statement

We represent the relationships between a set of documents D_n and a set of associated keywords K_m through a matrix $M_{m\times n}$. In this matrix, rows correspond to keywords, columns correspond to documents, and $M_{i,j}$ indicates whether the keyword kw_i is associated with the document d_j. For a given keyword kw_i, its response within the matrix M is calculated as $R(kw_i, M) = \{d_j | M_{i,j} = 1, \forall j \in [1, n]\}$.

Volume Leakage. The query volume leakages are represented by all sums of rows of M, which is calculated as $L_{qv}(M) = \{\sum_{j=1}^{n} M_{i,j}\}_{i\in[1,m]}$. Similarly, the document volume leakages are represented by all sums of columns and calculated as $L_{dv} = \{\sum_{i=1}^{m} M_{i,j}\}_{j\in[1,n]}$.

Obfuscated Matrix. Given k, it produces an obfuscated matrix ${M'}_{m'\times n'}$ with a set of obfuscated documents $D'_{n'}$ and a set of obfuscated keywords $K'_{m'}$, where $D'_{n'} \supseteq D_n$ and $K'_{m'} \supseteq K_m$. For any $i \in [1, m]$ and $i' \in [1, m']$, if $kw_i = kw_{i'}$, then $R(kw_{i'}, M') \supseteq R(kw_i, M)$. This ensures that the response set for any keyword $kw_{i'}$ in the obfuscated matrix M' is a super-set of the response set for the corresponding keyword kw_i in the original matrix M. In addition, for any value $v \in L_{qv}(M')$, it occurs at least k times in $L_{qv}(M')$. The same condition holds for $L_{dv}(M')$.

Problem. Given that the number of documents constitutes the majority of the storage overhead, the project problem is how to identify an obfuscated matrix ${M'}_{m'\times n'}$ with minimized n'.

2.1 Project Objectives.

The objectives of this project encompass designing a machine-learning-based approach to conceal volume leakage while relatively optimizing the number of obfuscated documents. Additionally, the project aims to conduct experiments using this approach on a sample set of documents and keywords, comparing the results with our prior approaches in terms of storage overhead and computation overhead.

Outcomes. This project offers practical exposure to the implementation, training, and fitting of machine learning models

3 Methodology

Building on our previous research, the strategy to conceal leakages involves the use of dummy documents, resulting in $K'_{m'} = K_m$. Additionally, we employ the modulo-based partition method to mask document volume and the keyword clustering method to conceal query volume—both techniques introduced in our prior research work.

The focus of the machine-learning-based approach is to seamlessly integrate these two methods while minimizing storage overhead overlap.

4 Challenges

The first challenge is to find a machine learning model with flexible sizes of input and output. Meanwhile, the machine learning model must be unsupervised since there are no labels or tags to identify samples into correct or not. Instead, there is supposed to be a measurement to access the goodness of the result, i.e., how optimized is the number of obfuscated documents.

4.1 Reinforcement Learning

To address the first challenge, we choose the reinforcement learning model as our fundamental machine learning model.

Reinforcement learning is a type of machine learning paradigm where an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or penalties based on its actions, and the goal is to learn a policy or strategy that maximizes the cumulative reward over time. Specifically, the agent observes the current state of the environment, selects an action based on its current policy, represented by a Q table for all state-action pairs, and then receives feedback from the environment in the form of a reward signal. The

agent then updates its policy based on this feedback, with the aim of improving its future decision-making.

However, in our project case, given a matrix $M_{m\times n}$, there would be $\mathcal{O}(m)$ partition actions, then $\mathcal{O}(mn)$ element allocation actions and $\mathcal{O}(m^2 + (m - k)^2 \log(m))$ dummy element allocation actions, which are introduced by our previous research work for document partition and keyword clustering methods. These actions lead to a huge number of states of the environment. In a nutshell, it requires a huge number of training episodes to train the Q table of a reinforcement learning model.

To address this problem, our project utilizes the approximate Q-learning. In approximate Q-learning, instead of using a table to store Q-values for each state-action pair, a function approximator such as a neural network is used to estimate the Q-value. This allows the algorithm to handle environments with large state spaces, as it is not feasible to store Q-values for every possible state-action pair.

The algorithm works by repeatedly taking actions in the environment and updating the Q-values using the Bellman equation, which relates the Q-value of a state-action pair to the expected reward of that action and the Q-values of the resulting state-action pairs. The weights of the function approximator are updated using stochastic gradient descent to minimize the error between the estimated Q-value and the actual Q-value.

Approximate Q-learning has been applied successfully in a wide range of domains, including robotics, game playing, and natural language processing. However, it requires careful tuning of hyper-parameters and can be sensitive to the choice of function approximator and the representation of the state space.

4.2 Measurement

This project uses $\frac{n'-n}{n}$ to access how optimized is the obfuscated matrix. The extremely optimal value is supposed to be 0, which indicates that the original matrix is already suppressing volumes.

5 Reinforcement Learning Approach to Suppress Volume Leakage

Reinforcement Learning approach to Suppress Volume Leakage trains an approximate Q-learning model that takes a matrix $M_{m\times n}$ and a k value, and generates an obfuscated matrix $M'_{m'\times n'}$. The output obfuscated matrix is supposed to be near-optimal.

The approach involves specific rules, reward functions, and feature extraction functions.

The environment maintains an original matrix $M^R_{r\times c}$, an obfuscated matrix M^O, and a column index mapping function Ind. The agent maintains the Q weights corresponding to features.

5.1 Rule

The rule defines the actions and success of the game based on the given environment.

Given an environment, there are three types of actions, including choosing the column partition modulo m, adding an empty column with a capacity to the obfuscated matrix, and assigning 1 to an element in the obfuscated matrix.

5.1.1 Choosing the column partition modulo m

This type of action is only available when the environment doesn't have a value for m. It allows the agent to choose a column partition modulo m from 2 to the maximum column volume value of the original matrix. For example, if the maximum column value of the original matrix is 5, there would be four possible actions for the agent: *set* $m = 2$, *set* $m = 3$, *set* $m = 4$, and *set* $m = 5$.

After receiving m, the environment calculates a mapping C from columns in the original matrix to columns in the obfuscated matrix as follows:

1. Firstly, it initializes $C \leftarrow \{\}$.
2. Then, for each column j where j ranges from 1 to n, it computes $ct \leftarrow \lfloor \frac{L_{dv}[j]}{m} \rfloor$.

3. If $L_{dv}[j] \bmod m > 0$, then $ct \leftarrow ct + 1$.

4. After that, ct new obfuscated column indexes which are mapped to the column j are computed as $|C|+1, |C|+2, ..., |C|+ct$.

5. It adds mappings between the new obfuscated columns and the column j, which is computed as $C \leftarrow C \cup \{(|C|+1, j), (|C|+2, j), ..., (|C|+ct, j)\}$.

6. It initializes three indicators d_c, d_r, d_p to 0, where d_c implies the previously added column index in the obfuscated matrix. d_r implies the row index of the previously assigned element. d_p implies the remaining capacity of the current mapped column in the original matrix.

7. It initializes an empty obfuscated matrix $M'_{m\times 0}$.

There are two functions, Ind_c and Ind_{cs}, associated with C. Ind_c takes as inputs a column index in the obfuscated matrix and outputs the mapped column index in the original matrix, while Ind_{cs} takes as inputs a column index in the original matrix and a set of column indexes in the obfuscated matrix.

5.1.2 Adding an empty column with a capacity

This type of action is only available when there are no columns in the obfuscated matrix with a volume smaller than their capacity. It allows the agent to add an empty column to the obfuscated matrix with a specified capacity. The upper and lower bounds of the capacity are calculated as follows.

1. It computes the new obfuscated column index as $j' \leftarrow d_c + 1$.

2. If j' is greater than $|C|$, which means it is a dummy column, then, the algorithm outputs the upper bound of capacity as m and the lower bound of capacity as 1; otherwise, it goes to the next step.

3. Then, the algorithm obtains the mapped column index $j \leftarrow Ind_c(j')$.

4. If d_p equals to zero, it computes $d_p \leftarrow L_{dv}[j]$.

5. Next, it computes $b_{upper} \leftarrow \min(m, d_p)$ and $b_{lower} \leftarrow d_p \bmod m$. If b_{lower} equals to zero, it lets b_{lower} be m.

6. It outputs the upper bound of capacity as b_{upper} and the lower bound of capacity as b_{lower}.

After receiving the new obfuscated column capacity cap, the environment adds such a new obfuscated column with the capacity cap to the obfuscated matrix as follows.

1. The environment adds a zero column as $M'_{m\times j'} \leftarrow \{M'_{m\times d_c}[i, \cdot] \cup \{0\}\}_{i\in[1,m]}$ where $j' \leftarrow d_c + 1$.

2. It sets $cap_{j'} \leftarrow cap$.

3. If j' is no greater than $|C|$, it updates the capacity indicator $d_p \leftarrow d_p - cap_{j'}$.

4. It updates the column indicator as $d_c \leftarrow j'$.

5. Next, the algorithm resets the row indicator as $d_r \leftarrow 0$.

5.1.3 Assigning 1 to an element

This type of action is only available when there exists a column in the obfuscated matrix whose capacity is less than its volume. This action allows the agent to assign an element with a value of 1 in the last column of the matrix. The possible row index of the element can be computed as follows.

1. If d_p is greater than $|C|$, then the algorithm returns the possible row index values as $1, 2, ..., r$, where r is the total number of rows in the original matrix. Otherwise, it goes to the next step.

2. It obtains the mapped column index $j \leftarrow Ind_c(d_c)$.

3. Next, it obtains the column vector $colV = M_{m\times n}[\cdot, j]$.

4. Starting from the row index d_r, the algorithm finds the nearest subsequent row index i in $colV$ such that $colV[i] = 1$.

5. The algorithm outputs i as the possible row index.

After receiving the row index i of the element to be assigned, the environment sets the corresponding element to 1 in the obfuscated matrix as below.

1. The algorithm sets the corresponding element in the obfuscated matrix to 1 by computing $M'_{m\times d_c}[i, d_c] \leftarrow 1$.
2. After that, it updates d_r as $d_r \leftarrow i$.

5.1.4 Success

Once the environment fulfills certain requirements, the task of generating an obfuscated matrix is considered successful or completed. Recall the definitions of k-ind query volume LR secure and k-ind document volume LR secure introduced by our previous research work, there are two requirements that must be satisfied to complete the task. The first is that all column volume values occur at least k times in the obfuscated matrix. The second is that all row volume values occur at least k times in the obfuscated matrix.

The success of the task is verified before each step, and once it is successful/completed, the model is terminated, and the obfuscated matrix $M_{m\times d_c}$ is outputted.

5.2 Reward

We use the number of columns as the rewards. When the agent takes the action of adding an empty column, if the index of the added column is greater than $|C|$, then the agent obtains a reward -1.

5.3 Feature Extraction

The feature extraction functions are designed to capture the corresponding leakages. To capture how far the current obfuscated matrix is from being k-ind query volume LR secure, the feature function calculates the count of each query volume and computes the sum of the residuals to k, which is formulated as $\sum max(qv_i - k, 0)$, where qv_i is the count of i in the query volumes. Then, it computes $e^{-\sum max(qv_i-k,0)}$ as the feature value. When the value is one, it means that there are no counts less than k.

Similarly, the feature value is computed to identify how far the current obfuscated matrix is from being k-ind document volume LR secure. It is generated as $e^{-\sum max(dv_i-k,0)}$, where dv_i represents the count of i in the document volumes.

5.3.1 Vector deviation

It takes an vector V_n as input and output the feature value.

1. It initializes $CTs \leftarrow \{\}$.
2. For each element j in V_n, it checks whether its value $V_n[j]$ exists in CTs.
3. If $V_n[j]$ exists in CTs, then, it updates $CTs[V_n[j]] \leftarrow CTs[V_n[j]] + 1$; Otherwise, it updates $CTs[V_n[j]] \leftarrow 1$.
4. Next, it computes $dist \leftarrow \sum_{v\in CTs} \max(v-k, 0)$.
5. The feature value is computed as e^{-dist}

5.3.2 k document volume deviation

It computes a column volume vector from the obfuscated matrix firstly and then performs the vector deviation algorithm to produce the feature value of k document volume deviation.

5.3.3 k query volume deviation

Similarly, it computes a row volume vector firstly from the obfuscated matrix and performs the vector deviation algorithm to produce the feature value of k query volume deviation.

6 Deliverables

1. A GitHub repository to maintain the code `https://github.com/yjm9110/ml_vlr/tree/main`.

2. A report on the detailed approach and experimental results.

3. Slides for the project presentation.

7 Timeline

Table 1: Project Timeline

Weeks	**Status**	**Task**
Week 1	●[1]	Explore potential machine learning models for leakage suppression.
Week 2	●	Identify and finalize the machine learning model for the project.
Week 3	●	Dedicate time to thoroughly understand the chosen machine learning model.
Week 4	●	Design the machine-learning-based method for volume leakage suppression.
Week 5	●	Refine the proposed method based on initial design considerations.
Week 6	●	Document the detailed construction of the machine-learning-based approach.
Week 7	○[2]	Implement the designed method.
Week 8	○	Conduct experiments to evaluate the storage and computation overheads of the approach.
Week 9	○	Compile the final project report.
Week 10	○	Prepare presentation slides.

[1] "●" indicates completed task.
[2] "○" represents pending task.