USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

ALERT: Machine Learning-Enhanced Risk Estimation for Databases Supporting Encrypted Queries

Longxiang Wang, *City University of Hong Kong;* Lei Xu, *Nanjing University of Science and Technology and City University of Hong Kong;* Yufei Chen, *City University of Hong Kong;* Ying Zou, *Nanjing University of Science and Technology;* Cong Wang, *City University of Hong Kong*

https://www.usenix.org/conference/usenixsecurity25/presentation/wang-longxiang

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

ALERT: Machine Learning-Enhanced Risk Estimation for Databases Supporting Encrypted Queries

Longxiang Wang[†,*], Lei Xu[‡,†,*], Yufei Chen[†], Ying Zou[‡], Cong Wang[†]

[†] Department of Computer Science, City University of Hong Kong
[‡] School of Mathematics and Statistics, Nanjing University of Science and Technology
longxwang4-c@my.cityu.edu.hk, xuleicrypto@gmail.com, yufeichen8@cityu.edu.hk
zouying@njust.edu.cn, congwang@cityu.edu.hk

Abstract

While searchable symmetric encryption (SSE) offers efficient, sublinear search over encrypted data, it remains susceptible to leakage abuse attacks (LAAs), which can exploit access and search patterns to compromise data privacy. Existing methods for quantifying leakage typically require a comprehensive analysis of all queries, making them unsuitable for real-time risk assessment. Since leakages in SSE are revealed incrementally with each query, there is a pressing need for risk assessments to be conducted on the fly, enabling prompt alerts to clients about potential privacy threats. To address this challenge, we propose ALERT, a machine learning-enhanced framework for real-time risk assessment in searchable encryption. ALERT leverages sophisticated learning algorithms to automatically identify keyword features from public auxiliary information, learning them as a classifier. When a query is executed, ALERT efficiently predicts the associated keyword and estimates the likelihood of leakage. Experimental results show that ALERT can deliver predictions within seconds, achieving a substantial speed-up of 31.1× compared to existing state-of-the-art methods.

1 Introduction

Searchable symmetric encryption, known as SSE, enabling sublinear complexity search over encrypted data, has been recognized as the most promising approach to serving encrypted databases. Significant progress has been made in SSE to enhance its expressiveness, efficiency, and security to meet the various demands of real-world applications [14, 15, 21–23, 35, 51, 52, 55–58, 66, 72, 75, 97, 100]. Parallel to these works, industries like MongoDB [6] and AWS [1] are actively working on deploying SSE to leverage its capability in practice.

Despite substantial academic success and wide adoption, concerns about SSE security persist. Several works, known as leakage abuse attacks (LAAs) [37, 43–46, 63–65, 74, 77, 81–83, 105], have shown that structural-equivalent leakages, such as access and search patterns, can be exploited by real-world adversaries to recover queries or reconstruct data. To better understand the impact of leakages in searchable encryption, studies have been initiated to quantify these leakages [13, 54, 62], aiming to provide clear insights into the privacy vulnerabilities of databases using SSE for search services.

However, a significant limitation of existing approaches to quantifying leakage in searchable encryption is their inability to provide timely and effective alerts. Such alerts are essential for notifying users when an adversary might infer sensitive information from their queries, helping them understand the potential risk of exposure. This point is widely recognized in other domains. For instance, in the context of safe browsing, systems monitor and promptly alert users about potentially malicious websites [3, 5], and operating systems warn users of security risks when opening executables from unauthorized developers [4, 7]. Regulatory frameworks like HIPAA [78] and NIST-SP-800-30 [79] further highlight the necessity of proactive notifications to mitigate security risks. Additionally, some works have proposed mitigating leakage by periodically rebuilding the encrypted dataset or index to reset the system's vulnerabilities [32, 40], real-time alerts can offer valuable guidance on when such rebuilding should occur to balance security and system performance.

In light of these observations, this paper aims to develop a real-time risk assessment framework that quantifies leakage in encrypted databases built on SSE. This framework operates on the fly, providing clients with prompt alerts about potential privacy threats as they arise.

1.1 Overview of the Contributions

We conduct a feasibility study proposing ALERT (illustrated in Figure 1) as the first attempt to develop a machine learning-enhanced framework for real-time risk assessment in databases supporting encrypted queries. ALERT automatically learns and memorizes features of plaintext queries and their clear responses as a classifier, enabling efficient monitoring and risk assessment in subsequent operations.

* The first two authors contributed equally to this work.

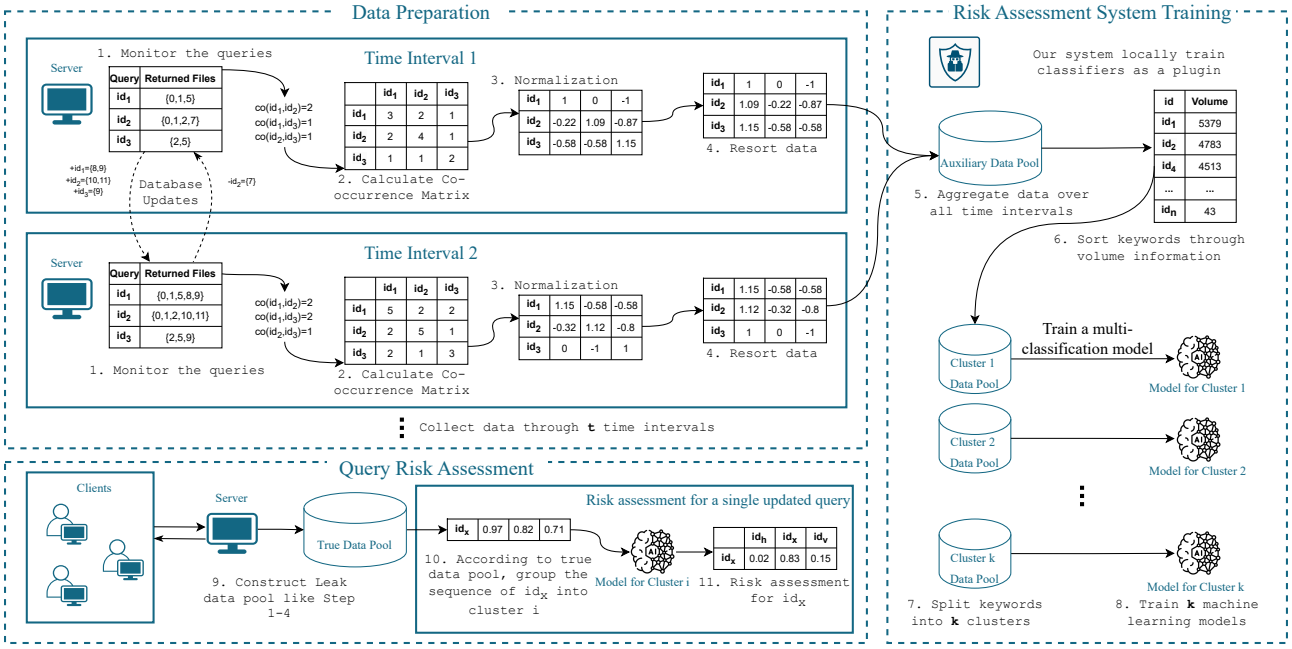


Figure 1: An overview of ALERT operation pipeline.

When an encrypted query is received, the classifier can swiftly extract relevant features, identify the potential corresponding keywords, and estimate their likelihoods. This process provides clients with valuable privacy insight by indicating which queries can be revealed by the newly proposed query if an adversary with background knowledge of the database were to monitor the query transcripts. Our evaluations demonstrate that ALERT can provide estimation results within seconds. Below, we briefly introduce the technique roadmap of developing the classifier and explain how it is employed to perform real-time risk estimation.

Learning-based Privacy Risk Estimator. To achieve the goal outlined above, the first step is to build an estimator for ALERT. However, this is not easy as the responses for the query responses are encrypted which makes it challenging to extract its features for prediction. We observe that a similar issue arises in the domain of encrypted traffic data analysis [8, 10, 18, 28, 29, 38, 49, 70, 73, 89–92, 99, 103]. These works propose identifying encrypted streams using invariant features, like volume, captured and learned by a classifier trained on client request features [87]. Encrypted streams with distinct features can then be directly identified. Inspired above, we also aim to train such a classifier for the SSE scheme by treating keywords as labels, and SSE, fortunately, has similar features termed structurally equivalent leakages. However, this does not mean their solution can be directly applied to SSE, as SSE presents additional challenges. In SSE, most structural leakages are query-dependent, often spanning multiple queries, which complicates the alignment between leaked information and the data used for training. Moreover, volume leakage in SSE is typically subtle and not significantly distinct from each other, as demonstrated by [12, 106], making it difficult to distinguish individual query responses.

To address these gaps, we introduce several data preprocessing measures. First, we apply vector reordering to align the training data for ALERT with the observed leakages, ensuring consistency across training and testing data. Second, we use repeated random sampling (RRS) to mitigate the distribution instability in dynamic databases [106].

Dynamic Keyword Clustering. Our initial empirical results demonstrate the effectiveness of machine learning-enhanced risk assessment. However, this approach is suboptimal due to the large number of classes, which resulted in an excessive number of model parameters, leading to increased latency and a lower recovery rate (as shown in Table 1). To avoid the sheer quantity of classes, ALERT employs a top-down hierarchical classification strategy - DCM. Similar strategies have been widely deployed in other scenarios where the label space is very large, such as IoT device classification and gene interaction [41, 80, 85, 101, 107]. DCM utilizes volume information to dynamically cluster keywords into several groups. For each cluster, DCM trains a dedicated model to identify the keywords within that group, thereby reducing model complexity and risk assessment latency.

Efficient Risk Assessment and Results Aggregation. Following the training phase, ALERT leverages the trained model to perform risk assessments for each new proposed query. This phase consists of two steps: processing to extract features from query responses (leakages) and predicting risk with the trained model. To facilitate process performance, ALERT

introduces an update operation that dynamically renews leakage patterns associated with the updated query, eliminating the need to rebuild them from scratch. This significantly accelerates data processing, particularly for databases with large keyword universes. During risk assessment, ALERT aggregates the predicted probabilities of identical queries across different timestamps in the dynamic encrypted database, making risk assessment results more accurate.

To validate the effectiveness of ALERT against the adversary who passively monitors the database query process, we conduct a series of experiments on three commonly-used datasets, Enron [2], NYTimes [96], and Wikipedia [102]. The results demonstrate that ALERT achieves a 31.1× speedup in database leakage assessment compared to existing SOTA leakage analysis methods [31, 76, 82] while maintaining comparable query recovery accuracy. Moreover, under low-latency scenarios, ALERT demonstrates a 24.4% higher median query recovery rate and can continue to perform reliably even under stricter time limits where other approaches struggle. Additionally, ALERT also remains robust across varying datasets and adversarial assumptions (Figure 4), different SSE schemes (Figure 6), and against SSE countermeasures (Figure 10).

In summary, our contributions are the following:

- We propose a scenario that estimates risks caused by potential query leakage in real-time from the user's perspective, identifying new requirements for leakage analysis in the SSE field.
- We develop a new ML-enhanced real-time risk assessment system that incorporates a dynamic keyword clustering mechanism to reduce assessment latency and model complexity, along with an efficient update operation that selectively adjusts leakage patterns to expedite data processing.
- We implement ALERT and conduct a series of experimental evaluations, demonstrating its capability to support robust real-time risk assessment across various data types and adversarial scenarios.

2 Preliminaries

2.1 Notations and Syntax of DSSE

Let $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ denote a set of n keywords arranged in lexicographic order, and let $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$ represent a collection of m files, each containing one or more keywords from $\mathcal{W}$. Each file f_i is associated with a unique identifier id_i. Correspondingly, let $\mathcal{Q} = \{q_1, q_2, \ldots, q_n\}$ represent the encrypted version of the keyword set $\mathcal{W}$. A database $\mathsf{DB} = \{(\mathsf{id}_i, \mathsf{W}_i)\}$ is a collection of identifier/keyword-set pairs, where W_i denotes the set of keywords occurrences in the file f_i. For any given keyword $w_i \in \mathcal{W}$, let $\mathsf{DB}(w_i) = \{\mathsf{id}_j : w_i \in f_j, 1 \le j \le m\}$ be the set of files containing. With those notations, the following gives the syntax of DSSE.

Definition 1. *A dynamic searchable symmetric encryption (DSSE) scheme comprises an algorithm* Setup *and two protocols,* Search *and* Update*, conducted between the client and the server.*

- $(K, \mathsf{EDB}) \leftarrow \mathsf{Setup}(\kappa, \mathsf{DB})$ *takes as input a security parameter* κ *and a plaintext database* DB*. It outputs the client's secret key* K *and an encrypted database* EDB *for the server.*
- $R \leftarrow \mathsf{Search}(K, w, \mathsf{EDB})$ *allows the client, using* K*, to search* EDB*, where* w *refers to the client's query and* R *is the set of files matching the query.*
- $\mathsf{EDB}' \leftarrow \mathsf{Update}(K, in, op, \mathsf{EDB})$ *allows the client using* K *and* $in \in \{(w, \mathsf{id})\}$*, and an update operation* $op \in \{\mathsf{Add}, \mathsf{Del}\}$*. The* Update *protocol adds (or deletes) the document to (or from)* EDB*, resulting in an updated encrypted database* EDB'*.*

We say that a DSSE scheme is correct if the Search protocol returns all matching files and the Update protocol consistently reflects the changes made to the plaintext database in the encrypted database.

2.2 Leakage Profiles

Based on the previously defined background of DSSE, we next defined the associated leakage profile. The leakage refers to the invariant features in the EDB before and after the SSE linear transformation. Depending on the database update operations and their corresponding timestamps, let EDB_μ denote the dynamic encrypted database at timestamp μ, where $|\mathsf{EDB}_\mu|$ represents the number of encrypted files at this timestamp. Some types of database leakage can be defined as follows:

Result pattern $\mathcal{M} = \{\mathsf{M}_1, \mathsf{M}_2, \cdots, \mathsf{M}_t\}$. M_μ describes the keyword search results for the submitted queries at timestamp μ. Let $\mathsf{M}_\mu[i, j] = 1$ denote that the result set of query q_i contains the encrypted file id_j at timestamp μ. For query set $\mathcal{Q}$, the result pattern at μ is defined as

$$\mathsf{M}_\mu(\mathcal{Q}) \in \{0,1\}^{n \times |\mathsf{EDB}_\mu|}.$$

Volume pattern $\mathcal{V} = \{\mathsf{V}_1, \mathsf{V}_2, \cdots, \mathsf{V}_t\}$. V_μ reports the number of returned files containing the submitted query at timestamp μ. Let $v_i^{(\mu)} = \sum(\mathsf{M}_\mu[i, j] = 1), 1 \le j \le |\mathsf{EDB}_\mu|$ denote the volume pattern of query q_i at timestamp μ. For $\mathcal{Q}$ and timestamp μ, the query volume pattern is defined as:

$$\mathsf{V}_\mu(\mathcal{Q}) = \{v_1^{(\mu)}, v_2^{(\mu)}, ..., v_n^{(\mu)}\} \in \mathbb{R}^n.$$

Co-occurrence pattern $\mathcal{COC} = \{\mathsf{CoC}_1, \mathsf{CoC}_2, \cdots, \mathsf{CoC}_t\}$. CoC_μ reports the number of files returned by two queries at timestamp μ. Let $\mathsf{co}_\mu[i, j] =< \mathsf{M}_\mu[i, :], \mathsf{M}_\mu[j, :] >$ denote the co-occurrence number of encrypted files returned by query q_i and q_j at timestamp μ. For $\mathcal{Q}$ and timestamp μ, the query co-occurrence pattern is defined as:

$$\mathsf{CoC}_\mu(\mathcal{Q}) = \mathsf{M}_\mu \cdot \mathsf{M}_\mu^\top \in \mathbb{Z}^{n \times n}.$$

2.3 Gradient Boosting Decision Trees (GBDT)

Gradient Boosting Decision Trees (GBDT) and its variants [27, 59, 84] are widely used tree-based machine learning algorithms. These models have become a standard approach in various industrial applications, including financial risk management [19, 60, 68] and online advertisement fraud detection [42, 69]. GBDT's popularity is due to its ability to model complex relationships and perform well on small datasets.

Given a training dataset $\mathsf{D} = (\mathbf{X}, \mathbf{y})$, where $\mathbf{X} \in \mathbb{R}^{n \times \varepsilon}$ represents n model inputs, each with ε dimensions, and $\mathbf{y} \in \mathbb{R}^n$ contains the corresponding n labels. GBDT aims to learn a sequence of decision trees $\mathsf{T}_\nu : \mathbb{R}^\varepsilon \mapsto \mathbb{R}$ in an additive manner to minimize a pre-defined loss function $\mathcal{L}$, such as cross-entropy for classification. At each iteration ν, a new tree T_ν is constructed to fit the negative gradients of the loss with respect to the current ensemble predictions, i.e., $\mathsf{T}_\nu(\mathbf{X}) \approx -\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}^{(\nu-1)})}{\partial \hat{\mathbf{y}}^{(\nu-1)}}$, where $\hat{\mathbf{y}}^{(\nu-1)} = \sum_{i=1}^{\nu-1} \mathsf{T}_i(\mathbf{X})$ is the prediction of the ensemble up to the previous iteration. This gradient step allows the new tree to correct the mistakes made by the existing ensemble.

3 Problem Formulation

3.1 Participants and Assumptions

Figure 2 illustrates the system model in the context of query risk assessment. Our system model consists of two parties: a client and a server. The client possesses a privacy-sensitive dataset that must be securely stored on an "honest but curious" server. To safeguard data privacy while retaining search functionality, the client encrypts the dataset using searchable symmetric encryption before outsourcing. When the client wants to retrieve the files containing a specific keyword, they generate a search token with the secret key and the keyword and send it to the server. The server leverages the token to identify the matching files and returns the files or corresponding identifiers to the client.

As mentioned, the server is honest but curious. While the server faithfully follows the protocol for storing and processing encrypted data and queries, it is also interested in learning as much as possible about the data and queries it handles. Additionally, the server has access to auxiliary information that has been well-motivated in many existing works, such as response volume and co-occurrence frequency across keywords, which it can learn from analyzing similar public datasets.

3.2 Formal Problem Description

To address the potential security concerns posed by the server, it is essential to develop a systematic approach that enables the client to identify these risks and assess the vulnerabilities associated with the encrypted database in use. For clarity, we refer to this approach as the "privacy controller". Specifically, the privacy controller operates by analyzing the server's

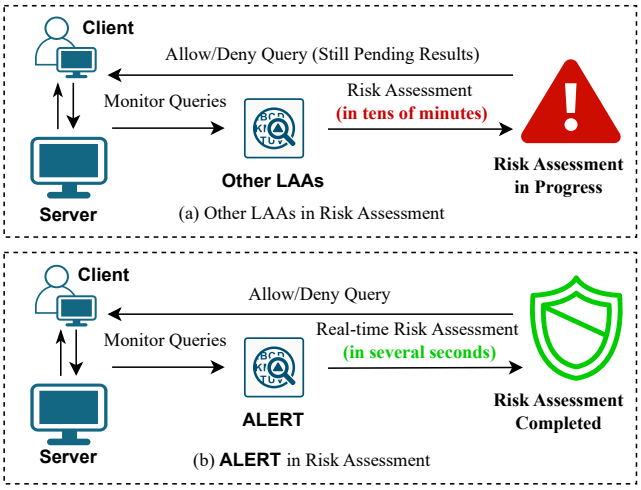


Figure 2: Usability comparisons between other LAAs and our ALERT in query risk assessment.

responses to each search query. By doing so, it can predict potential keywords that may be inferred from the query, along with the associated likelihood of these inferences, from the view of an adversary such as the server. The assessment operations need to be efficient to avoid introducing significant delays to the query process.

In the context of SSE, the potential keywords queried by the client inherently belong to a predefined keyword space. By treating these keywords as labels and considering the leakages extracted from the server (detailed described in Section 2.2) responses as attribute values, the problem can be reframed as a multi-classification task, a well-established problem in machine learning. Specifically, the client leverages a classifier to label each server response based on the observed leakage attributes. However, it is not easy, as the data observed by the adversary are encrypted and cannot be directly used. Regarding this, our objective involves precisely mapping the auxiliary information and observed leakages into a structured format suitable for machine learning, followed by selecting appropriate algorithms and training strategies to optimize the classifier's performance in predicting potential inferences.

Formally, let $\mathcal{W} = \{w\}$ be the label space and Aux represent the auxiliary information available to the adversary. This auxiliary information could be a dataset that shares a similar distribution with the target dataset or statistical information that is publicly accessible. Accordingly, let $\mathsf{Leak} = \{(\tilde{\mathcal{Q}}, \tilde{\mathcal{M}}(\tilde{\mathcal{Q}}), \tilde{\mathcal{V}}(\tilde{\mathcal{Q}}), \ldots)\}$ be leakages to the adversary which can be observed through the query transcripts. Then, the task of designing a privacy controller can be broken down into the following three subtasks:

Data Preparation. We devise the data preparation algorithm, DaTPre, to map Aux and Leak into a structured format suitable for machine learning. The dataset obtained from the former one is namely the training dataset, and the latter one

is the test dataset. For clarity, we denote them as

$$\mathsf{D}_{\mathsf{train}} := \{(\mathbf{x}_i, y_i)\} \leftarrow \mathsf{DaTPre}(\mathsf{Aux})$$

and

$$\mathsf{D}_{\mathsf{test}} := \{\hat{\mathbf{x}}_j\} \leftarrow \mathsf{DaTPre}(\mathsf{Leak}),$$

where $\mathbf{x}_i$ denotes the feature of the i-th single data point with label y_i and $\hat{\mathbf{x}}_j$ is the feature of the j-th data point to be tested.

Classifier Training. We train a classifier with $\mathsf{D}_{\mathsf{train}}$ to learn the complex relationships between structured leakage and its associated keywords, with the objective of minimizing the error between the predicted and actual labels. Mathematically, the desired model can be formalized as

$$\mathcal{T}^* := \mathrm{argmin}_{\mathcal{T}} \mathcal{L}(\mathcal{T}; \mathsf{D}_{\mathsf{train}}).$$

Privacy Assessment. We update the test dataset with the updated query-file set $\mathcal{U}$ and previous database state $\mathcal{S}$. Here, $\mathcal{U}$ represents the result pattern associated with the updated query q extracted from the query-file set $\mathcal{M}$ across t timestamps. $\bar{\mathcal{S}}$ includes the prior query-file set $\bar{\mathcal{M}}$ and the co-occurrence set $\mathcal{C}\bar{\mathcal{O}}\mathcal{C}$ constructed from earlier queries, which contains matrixes across t timestamps. Formally, the updating process can be described as

$$\mathsf{D}_{\mathsf{test}} \leftarrow \mathsf{DaTPre}(\mathsf{Leak}, \bar{\mathcal{S}}, \mathcal{U}).$$

Then, predict the label of each data point $\hat{x}_j \in \mathsf{D}_{\mathsf{test}}$ via the trained model $\mathcal{T}^*$. The predicted result for the query,

$$\{(\hat{\mathbf{x}}_j, \{\hat{y}_k, p_{j,k}\}_{k=1}^n)\} \leftarrow \mathcal{T}^*(\mathsf{D}_{\mathsf{test}})$$

is a set of label-likelihood pairs, where $\hat{y}_k$ is the potential label (i.e., keyword) and $p_{j,k} := \mathsf{P}(\hat{y}_k|\hat{\mathbf{x}}_j)$ is the corresponding likelihood (represented by predicted probabilities).

4 Our ML-enhanced Query Risk Assessment

We now introduce our machine learning-enhanced privacy assessment framework. We begin by outlining the pipeline, which details the process of data preparation, classifier training, and real-time risk prediction for queries. Following this, we present an enhanced version of the pipeline that improves efficiency through the application of top-down principles.

4.1 Pipeline of Risk Assessment

Following the roadmap above, we start with mapping the auxiliary information and observed leakages into a structured format suitable for ALERT. Subsequently, we train a multi-class classifier using the prepared data, which is then employed to estimate potential leakages upon the arrival of a new query. Algorithm 1 details the procedure for data initialization and updating, with an update flag upd to distinguish between these two operations. Algorithm 2 outlines the workflow for classifier training and risk assessment, focusing particularly on the risk assessment process for updated queries.

Algorithm 1: Data Preparation (DaTPre)

Input: Auxiliary(Observed) information Aux(Leak), time intervals t_{ori}, data preparation parameters β, δ, η, update flag upd, total number of keywords n; (if upd $= True$: previous database status $\bar{\mathcal{S}}$, updated query-file set $\mathcal{U}$.)

Output: Prepared dataset D, database status $\mathcal{S}$

initialize query-file set $\mathcal{M}$ with Aux(Leak);
$\mathcal{S} \leftarrow \emptyset$ and add $\mathcal{M}$ into $\mathcal{S}$;
$\mathsf{D} \leftarrow [\,], t \leftarrow \lfloor \beta \times t_{ori} \rfloor, \delta \leftarrow \lceil \delta \times t \rceil, \eta \leftarrow \lceil \eta \times t \rceil$;
for $\mu = 1$ **to** t **do**
  if upd $= True$ **then**
    $\mathsf{CoC}_\mu \leftarrow \begin{bmatrix} \bar{\mathsf{CoC}}_\mu & (\mathsf{U}_\mu \cdot \bar{\mathsf{M}}_\mu^\top)^\top \\ \mathsf{U}_\mu \cdot \bar{\mathsf{M}}_\mu^\top & \mathsf{U}_\mu \cdot \mathsf{U}_\mu^\top \end{bmatrix}$
  else
    $\mathsf{CoC}_\mu \leftarrow \mathsf{M}_\mu \cdot \mathsf{M}_\mu^\top$; // Initialization
  add CoC_μ into $\mathcal{S}$;
  foreach *vector* $\mathsf{co} \in \mathsf{CoC}_\mu$ **do**
    pad co to length n;
    $\bar{\mathsf{co}}, \sigma_{\mathsf{co}} \leftarrow$ calculate mean and std of co;
    $\hat{\mathsf{co}} \leftarrow \left\{ \frac{(\mathsf{co}_j - \bar{\mathsf{co}})}{\sigma_{\mathsf{co}}} \mid \mathsf{co}_j \in \mathsf{co} \right\}$;
    $\mathsf{co}' \leftarrow$ sort $\hat{\mathsf{co}}$ in descending order ;
    add co' into CoC'_μ;
$\mathcal{COC}' \leftarrow \{\mathsf{CoC}'_1, \cdots, \mathsf{CoC}'_\mathsf{t}\}$;
foreach $\mathsf{QM}_k = \left[\mathsf{co}_k'^{(1)}, \ldots, \mathsf{co}_k'^{(t)}\right] \in \mathcal{COC}'$ **do**
  initialize a random binary matrix $\mathsf{A}_k \in \{0,1\}^{t \times \eta}$;
  $\mathsf{D}_k \leftarrow (\mathsf{QM}_k \; \mathsf{A}_k)/\delta$ and concatenate D_k into D;
D $\leftarrow$ PCA(D);
return D, $\mathcal{S}$;

Training Data Preparation. Let $\mathsf{Aux} = \{\mathcal{W}, \mathcal{Q}, \mathcal{M}\}$ be auxiliary dataset available to the simulated adversary. We first convert the query result set $\mathcal{M}$ into co-occurrence set $\mathcal{COC}$ as described in Section 2.2. Over the entire time period, the result set contains t_{ori} matrices. We select $t = \lfloor \beta \times t_{ori} \rfloor$ timestamps, where $\beta \in (0,1]$ is a parameter controlling the proportion of data snapshots used in training the classifier. Subsequently, the co-occurrence set consists of t co-occurrence matrices, denoted as $\mathcal{COC} = \{\mathsf{CoC}_1, \mathsf{CoC}_2, \cdots, \mathsf{CoC}_\mathsf{t}\}$. The co-occurrence matrix CoC_μ at μ contains n co-occurrence vectors: $\mathsf{CoC}_\mu = \left[\mathsf{co}_1^{(\mu)}, \mathsf{co}_2^{(\mu)}, \ldots, \mathsf{co}_n^{(\mu)}\right]$, where $\mathsf{co}_k^{(\mu)}$ denotes the co-occurrence vector for the k-th query at μ.

Next, we normalize each co-occurrence vector $\mathsf{co}_i^{(\mu)} \in \mathsf{CoC}_\mu$ with the Z-score technique to eliminate the differences in magnitude across vectors at different timestamps. Specifically, we first calculate the mean $\bar{\mathsf{co}}_i^{(\mu)}$ and standard deviation

Algorithm 2: Classifier Training and Risk Prediction

Input: Auxiliary information Aux, observed information Leak, updated query-file set $\mathbb{U}$, keyword space $\mathcal{W}$, time intervals $t, \tilde{t}$, data preparation parameters $\beta, \tilde{\beta}, \delta, \tilde{\delta}, \eta, \tilde{\eta}$

Output: Recovered queries RQ, risk map RM

$n \leftarrow$ get the total number of keywords in $\mathcal{W}$;
$\mathsf{D}_{\text{train}}, \mathcal{S}_{\text{train}} \leftarrow \mathsf{DaTPre}(\mathsf{Aux}, t, \beta, \delta, \eta, \mathsf{upd}, n)$;
$\mathcal{T}^* \leftarrow \textsc{Training}(\mathsf{D}_{\text{train}})$;
$\mathsf{D}_{\text{test}}, \mathcal{S}_{\text{test}} \leftarrow \mathsf{DaTPre}(\mathsf{Leak}, \tilde{t}, \tilde{\beta}, \tilde{\delta}, \tilde{\eta}, \mathsf{upd}, n)$;
/* Risk assessment phase */
$\mathsf{D}_{\text{test}}, \mathcal{S}_{\text{test}} \leftarrow \mathsf{DaTPre}(\mathsf{Leak}, \tilde{t}, \tilde{\beta}, \tilde{\delta}, \tilde{\eta}, \mathsf{upd}, n, \mathcal{S}_{\text{test}}, \mathcal{U})$;
foreach *sample* $\hat{\mathbf{x}}_i \in \mathsf{D}_{test}$ **do**
  $p_i \leftarrow \textsc{Prediction}(\mathcal{T}^*, \hat{\mathbf{x}}_i)$;
  $\mathsf{P}_\mathsf{j} \leftarrow \frac{1}{\eta}\sum_{k=1}^{\eta} p_k$, $\hat{y}_j \leftarrow \text{argmax}(\mathsf{P}_\mathsf{j})$;
return query risk map $\mathsf{RM} \leftarrow \{\mathsf{P}_1, \ldots, \mathsf{P}_\mathsf{n}\}$ and recovered queries $\mathsf{RQ} \leftarrow \{\hat{y}_1, \ldots, \hat{y}_n\}$;

$\sigma_i^{(\mu)}$ of $\mathsf{co}_i^{(\mu)}$. The normalization is then performed as follows:

$$\hat{\mathsf{co}}_i^{(\mu)} = \left(\mathsf{co}_{i,j}^{(\mu)} - \bar{\mathsf{co}}_i^{(\mu)}\right) / \sigma_i^{(\mu)}, \quad \mathsf{co}_{i,j}^{(\mu)} \in \mathsf{co}_i^{(\mu)}.$$

The sorted co-occurrence matrix CoC'_μ is constructed by resorting each vector $\hat{\mathsf{co}}_i^{(\mu)}$, it can be expressed as $\mathsf{CoC}'_\mu = \left[\mathsf{co}_1'^{(\mu)}, \mathsf{co}_2'^{(\mu)}, \cdots, \mathsf{co}_n'^{(\mu)}\right]$, where $\mathsf{co}_i'^{(\mu)} = \text{sort}(\hat{\mathsf{co}}_i^{(\mu)})$. The "sort" function rearranges the elements within the vector in descending order. This step is designed to ignore the co-occurrence between specific queries during the training process, only focusing on the co-occurrence for each query facing the whole data. Correspondingly, we obtain the sorted co-occurrence set $\mathcal{COC}' = \{\mathsf{CoC}'_1, \mathsf{CoC}'_2, \cdots, \mathsf{CoC}'_\mathsf{t}\}$.

We now have the data in an available format for training. However, as mentioned by Xu et al. [106], the data distribution in a short time interval is often unstable. To address this issue, we enhance model robustness by augmenting the training data with repeated random sampling (RRS). We augment data for each query-related co-occurrence matrix. For query q_k, we extract its related vectors from the sorted co-occurrence set $\mathcal{COC}'$ and organize them into a matrix $\mathsf{QM}_k = \left[\mathsf{co}_k'^{(1)}, \mathsf{co}_k'^{(2)}, \cdots, \mathsf{co}_k'^{(t)}\right] \in \mathbb{R}^{n \times t}$. For data augmentation, we define random sampling parameters δ and η, where δ determines the proportion of data selected per sample, leading to the selection of $\lceil \delta \times t \rceil$ vectors per sample. The augmentation factor η determines the number of augmented samples generated per class. For each query q_k, we generate $\lceil \eta \times t \rceil$ augmented query vectors. Next, we perform augmentation on QM_k. We construct a binary matrix $\mathsf{A}_k \in \{0,1\}^{t \times \lceil \eta \times t \rceil}$, where for each column, we randomly set $\lceil \delta \times t \rceil$ entries to one and the remaining entries to zero, serving as the sampling matrix. The augmentation procedure can be formulated as follows:

$$\mathsf{D}_k = \frac{\mathsf{QM}_k \ \mathsf{A}_k}{\lceil \delta \times t \rceil}, \tag{1}$$

where RRS is applied to all n queries. D_k aggregates co-occurrence vectors across timestamps, capturing cumulative co-occurrence information while preserving temporal context. Finally, the resulting matrices $\{\mathsf{D}_k\}$ are concatenated to construct the overall augmented data matrix $\mathsf{D} \in \mathbb{R}^{n \times (n \times \lceil \eta \times t \rceil)}$.

After constructing D, we apply principal component analysis (PCA) to further process the data $\mathsf{D}_{\text{train}} = \text{PCA}(\mathsf{D})$. PCA is employed to reduce the dimensionality of the generated data matrix D while retaining the most significant features that capture the maximum variance, which helps reduce model complexity and prevents overfitting.

Classifier Training. With the preprocessed training data $\mathsf{D}_{\text{train}}$, we proceed to train our multi-classification model to serve as the risk estimator. To effectively train the model, we incorporate the co-occurrence information between query pairs using one of the most advanced GBDT models, Catboost [84]. The model is optimized by minimizing the categorical cross-entropy loss function.

Real-time Risk Estimation. After obtaining the well-trained model, ALERT utilizes it to predict the risk associated with each new query proposed by the client in real-time. The risk estimation process begins by preparing the test dataset, D_{test}, with the observed leakage Leak. This preparation follows the same principles as in the training data preparation. However, unlike during training, the system updates the co-occurrence set, $\mathcal{C}\tilde{\mathcal{O}}\mathcal{C}$, more efficiently when a new query is proposed by the client. Specifically, before the updated query q_i and its corresponding query-response set $\mathcal{U} = \{\mathsf{U}_1, \mathsf{U}_2, \ldots, \mathsf{U}_t\}$ arrive, the system maintains a reserved database state, denoted as $\bar{\mathcal{S}}$. At time μ, the query-file vector is denoted as U_μ. To update the co-occurrence matrix, ALERT calculates the volume of the updated query (represented by $\mathsf{U}_\mu \cdot \mathsf{U}_\mu^\top$) and the co-occurrence vector between U_μ and other queries. These updates are then integrated into the new co-occurrence matrix. Formally, $\tilde{\mathsf{CoC}}_\mu$ is expressed as:

$$\tilde{\mathsf{CoC}}_\mu = \begin{bmatrix} \bar{\mathsf{CoC}}_\mu & (\mathsf{U}_\mu \cdot \bar{\mathsf{M}}_\mu^\top)^\top \\ \mathsf{U}_\mu \cdot \bar{\mathsf{M}}_\mu^\top & \mathsf{U}_\mu \cdot \mathsf{U}_\mu^\top \end{bmatrix}. \tag{2}$$

This operation reduces the computational load in testing data preparation. Following the update, we apply normalization and augmentation techniques and then use the PCA model derived from the training process to transform the testing data.

Once D_{test} is obtained, the well-trained model is employed for risk assessment. Notably, unlike typical machine learning tasks, our task benefits from prior knowledge of these relationships. We are aware of which samples pertain to the same class (i.e., keyword), providing a basis for result aggregation. Moreover, the model often assigns high probabilities to correct categories even with occasional misclassifications, underpinning the effectiveness of result aggregation. Thus, we aggregate the prediction results for each class. In the testing phase, we sample $\lceil \tilde{\eta} \times t \rceil$ samples of each class, denoted as $\chi_j = \{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_{\lceil \tilde{\eta} \times t \rceil}\}$. For each sample $\mathbf{x}_i$, the prediction probability is denoted as p_i, which is a vector of size n.

Algorithm 3: Dynamic Keyword Clustering

Input: Volume information $\mathcal{V}$, query set $\mathcal{Q}$, predefined threshold list ψ, time intervals t, scaling factor ε
Output: Keyword clusters $\mathcal{G}$

$\mathsf{SV} \leftarrow$ sum volume V of entire time period;
$\mathcal{Q}' \leftarrow$ resort $\mathcal{Q}$ with SV in descending order;
foreach $q_k \in \mathcal{Q}$ **do**
  for $\mu \leftarrow 0$ **to** t **do**
    $\mathsf{R}_k^{(\mu)} \leftarrow$ get query ranks for q_k with V_μ;
  $a_k, s_k \leftarrow$ calculate mean and std of R_k;
$\theta \leftarrow \emptyset, \mathsf{cnt} \leftarrow 0$;
while $\theta \neq \emptyset$ **and** number of remaining queries $\geq \theta$[last_idx] **do**
  for $\psi_k \in \psi$ **do**
    $a_{\mathrm{avg},k} \leftarrow \mathrm{average}(a_{\mathsf{cnt}}, a_{\mathsf{cnt}+1}, \cdots, a_{\mathsf{cnt}+\psi_k})$;
    $s_{\mathrm{avg},k} \leftarrow \mathrm{average}(s_{\mathsf{cnt}}, s_{\mathsf{cnt}+1}, \cdots, s_{\mathsf{cnt}+\psi_k})$;
    $\theta_i \leftarrow \min\left\{\psi_k \;\middle|\; \begin{array}{l} a_{\mathrm{avg},k} - \varepsilon \cdot s_{\mathrm{avg},k} \geq \mathsf{cnt} \textbf{ and} \\ a_{\mathrm{avg},k} + \varepsilon \cdot s_{\mathrm{avg},k} \leq \mathsf{cnt} + \psi_k \end{array}\right\}$;
    $\mathsf{cnt} \leftarrow \mathsf{cnt} + \theta_i$ and add θ_i into θ;
  θ[last_idx] $\leftarrow \theta$[last_idx] $+ n - \mathsf{cnt}$; `// Add remaining queries to the last cluster`
return $\mathcal{G} \leftarrow$ split $\mathcal{Q}'$ into ξ groups with θ;

The aggregated predicted probability P_j and corresponding aggregated predicted query $\hat{y}_j$ are defined as:

$$\mathsf{P}_j = \frac{1}{\tilde{\eta}} \sum_{i=1}^{\tilde{\eta}} p_i \ , \ \hat{y}_j = \arg\max(\mathsf{P}_j).$$

The overall risk assessment map RM and recovered queries RQ for query space $\mathcal{Q}$ are denoted as $\mathsf{RM} = \{\mathsf{P}_1, \mathsf{P}_2, \cdots, \mathsf{P}_n\}$ and $\mathsf{RQ} = \{\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_n\}$, respectively.

Observe that the test data does include timestamps that are present in the training dataset in our design. This overlap was designed to reflect scenarios where models rely on historical data to predict or analyze patterns for the same or overlapping periods. Meanwhile, our approach assumes an adversary's auxiliary information is based on historical observations up to the test timestamps. This setup reflects practical use cases and does not involve using future data to make predictions, thereby preserving the logical consistency of causality. Specifically, the adversary can only observe the most recent database snapshot, ALERT processes these observations to align update features with the training data using the temporal information embedded in the encrypted files.

Remark. While the pipeline offers a way to assess the risk of queries performed over encrypted databases, as demonstrated later in Table 1, it requires approximately 100 seconds to perform an overall risk assessment, hindering the achievement of real-time risk assessment objectives. This is particularly due to the sheer quantity of classes (i.e., keywords), which significantly increases model complexity.

4.2 Facilitating Efficiency with Keyword Pre-classification

To address the efficiency limitation above, we introduce DCM, a dynamic keyword clustering mechanism for pre-classification. Specifically, ALERT employs a top-down strategy that first classifies the keywords into several clusters based on volume information, and then further classifies them into specific classes using more sophisticated leakage patterns. We need to note that DCM is not always necessary in the risk assessment process; it serves as an extension of ALERT, and it is utilized when the target dataset contains a sheer quantity of classes. The procedure of DCM is formally described in Algorithm 3. Specifically, we analyze queries' historical volume rankings to determine clustering thresholds. By examining the temporal variations in these rankings, we identify stability patterns across different queries. These stability patterns, in turn, guide the dynamic establishment of clustering boundaries, which reflect the stability variants of the queries.

Specifically, we utilize volume information $\mathcal{V}$ and query space $\mathcal{Q}$ contained in Aux for pre-classification. Our first step is to aggregate the volume information across the entire time interval to obtain the overall volume information, denoted as SV, where $\mathsf{SV} = \sum_{i=1}^{t} \mathsf{V}_i = \{sv_1, sv_2, \ldots, sv_n\}$, where sv_j represents the aggregated volume for the j-th query. Then, we resort to the queries $\mathcal{Q}$ in descending order based on SV. The reordered set of queries, $\mathcal{Q}'$, is defined as:

$$\mathcal{Q}' = \{q_{\pi(1)}, q_{\pi(2)}, \ldots, q_{\pi(n)}\},$$

where π is defined such that $sv_{\pi(1)} \geq sv_{\pi(2)} \geq \cdots \geq sv_{\pi(n)}$.

For query q_k, we log its ranking across t timestamps in the ranking vector R_k. Specifically, element $\mathsf{R}_k^{(\mu)} \in \mathsf{R}_k$ records the rank of q_k at timestamp μ based on volume descending sorting order. We extract the rankings of each query across all timestamps and store them as a set of ranking vectors $\mathcal{R} = \{\mathsf{R}_1, \mathsf{R}_2, \ldots, \mathsf{R}_n\}$. Next, we compute the mean $a_k = \frac{1}{t}\sum_{i=1}^{t} \mathsf{R}_k^{(i)}$ and standard deviation $s_k = \sqrt{\frac{1}{t}\sum_{i=1}^{t}(\mathsf{R}_k^{(i)} - a_k)^2}$ for each vector $\mathsf{R}_k \in \mathcal{R}$, which reflect the central tendency and variability of volume ranking of q_k over time. A larger standard deviation indicates more variability in the data, suggesting the need for a higher splitting threshold to ensure accurate pre-classification.

We then calculate the splitting threshold θ_i based on the mean and standard deviation values. Our goal is to minimize θ_i while ensuring that the weighted average standard deviation of the clustered queries remains within acceptable bounds. Specifically, θ_i is determined as follows:

$$\begin{aligned} \theta_i = \min_{\theta} \quad & \theta \\ \text{s.t.} \quad & \frac{1}{\theta}\sum_{j=\mathsf{cnt}}^{\mathsf{cnt}+\theta} a_j - \frac{\varepsilon}{\theta}\sum_{j=\mathsf{cnt}}^{\mathsf{cnt}+\theta} s_j \geq \mathsf{cnt}, \\ & \frac{1}{\theta}\sum_{j=\mathsf{cnt}}^{\mathsf{cnt}+\theta} a_j + \frac{\varepsilon}{\theta}\sum_{j=\mathsf{cnt}}^{\mathsf{cnt}+\theta} s_j \leq \mathsf{cnt}+\theta. \end{aligned}$$

where $\theta_0 = 0$. Here, $\mathsf{cnt} = \sum_{k=0}^{i-1} \theta_k$, representing the cumulative sum of all queries in the previous clusters. ε is a scaling factor that controls the permissible deviation of accumulated standard deviation values. A larger ε tends to a larger θ, including more keywords in each cluster. Considering the data transfer time, we predefined a set ψ, which contains several candidates' values of θ. It avoids the redundant overhead caused by iterating through all possible values. The mechanism will loop until the number of remaining queries is less than or equal to the value of the last θ, at which point the remaining queries will be added to the final cluster.

Then, we proceed to partition the $\mathcal{Q}'$ into several clusters with the splitting threshold set θ. Given a splitting threshold set $\theta = \{\theta_1, \theta_2, \ldots, \theta_\xi\}$, the cluster G_i is defined as:

$$G_i = \{q_{\pi(\mathsf{cnt}+1)}, q_{\pi(\mathsf{cnt}+2)}, \cdots, q_{\pi(\mathsf{cnt}+\theta_i)}\},$$

for $i = 1, 2, \cdots, \xi$, where $\mathsf{cnt} = \sum_{k=0}^{i-1} \theta_k$. After forming the query clusters $\mathcal{G} = \{G_1, G_2, \cdots, G_\xi\}$, we proceed to cluster the columns of $\mathcal{COC}'$ based on $\mathcal{G}$. Once clustered, we perform RRS sampling in Equation 1 and later process the resulting submatrices. This mechanism allows us to partition $\mathcal{COC}'$ into smaller, more manageable submatrices, each of which can be used to train individual models.

5 Experimental Evaluation

To validate the proposed risk perception system, a comprehensive series of experimental evaluations is undertaken. Our goals are twofold: (1) to assess the efficacy of the ALERT by evaluating its accuracy and efficiency in predicting the queried keywords, and (2) to present empirical findings illustrating how risk levels change during database usage.[1]

5.1 Experiment Setup

We implement and evaluate ALERT on a server with two Intel Xeon Platinum 8383C CPUs and eight Nvidia RTX A6000 graphics cards running on Ubuntu 22.04. The detailed experiment settings are described below.

Adversarial Assumption. To further understand the impact of adversarial knowledge on risks, we explore two adversarial scenarios in our experiments:

- **Partially Known Dataset.** The adversary has precise knowledge of a subset of the database, with γ representing the fraction of data known to them. The difficulty of query recovery increases as this fraction decreases.
- **Sampled Dataset.** The dataset is split into two disjoint sets, Aux and Leak, with α representing the proportion of data assigned to Aux. The separation of data for training and testing intensifies the difficulty of query recovery.

[1] The code is publicly available at https://doi.org/10.5281/zenodo.14726862.

Target Datasets. We use three widely adopted public datasets from different domains: the Enron email dataset [2], the NYTimes article dataset [96], and the Wikipedia encyclopedia entries dataset [102]. Specifically, following the methodology in [82], we extract a fixed number of the most popular keywords from each dataset. For clarity, we denote the datasets containing n selected keywords and their corresponding files as En_n for Enron, Ny_n for NYTimes, and Wiki_n for Wikipedia. For Enron, we use approximately 5.03×10^5 documents spanning from January 2000 to July 2007. For NYTimes, we use 3.00×10^5 documents. For Wikipedia, we utilize 1.01×10^6 entries. The NYTimes and Wikipedia datasets we utilize are bag-of-words datasets that lack the inherent temporal information required for dynamic SSE. To address this limitation, we partition the NYTimes and Wikipedia data into 30 timestamps, aligning them with the Enron dataset's temporal distribution. Similar simulation approaches for deriving dynamic flows from static datasets have been adopted in prior works [61].

Exploited Leakages. Following previous work on encrypted database leakage abuse attacks [31, 76, 82], we assume that a passive attacker can observe certain leakage from queries to the encrypted dynamic database. Here, we consider two leakage patterns: volume pattern and co-occurrence pattern.

Metrics. In our evaluation, we use three metrics: *Risk Assessment Latency* (RAL), *Clustering Success Rate* (CSR), and *Query Recovery Rate* (QRR). RAL measures the time taken for the risk assessment system to update risk levels for all queries when a new query is submitted. CSR tracks the percentage of samples correctly assigned to their respective clusters during pre-classification. Lastly, QRR measures the proportion of test queries correctly classified as the target class, providing insight into the risk assessment accuracy.

Parameter Settings for ALERT. Recall that the selections of β, η, δ, can significantly influence ALERT's performance in terms of both QRR and RAL (see Appendix A for more details). We briefly introduce how we selected these parameters.

- β controls the selection of a portion of database snapshots, $\tilde{t}$, from the dynamic database and ranges from 0 to 1. A smaller β reduces computational overhead during test data preparation, lowering RAL, while a larger β provides richer database information. We set β to 1.0 during model training and 0.4 during testing by default.
- δ regulates the stability of each augmented data sample, with a recommended range of 0.2 to 0.6. A larger δ is preferable when the data contains many outliers, and a smaller δ works better with more stable data. We set the default value of δ to 0.1 for training and 0.4 for testing.
- η controls the redundancy of augmented data. A higher η improves system performance but also increases the number of samples to be predicted, leading to higher RAL. η is set to 2.0 for both training and testing to maintain a balance between performance and efficiency.

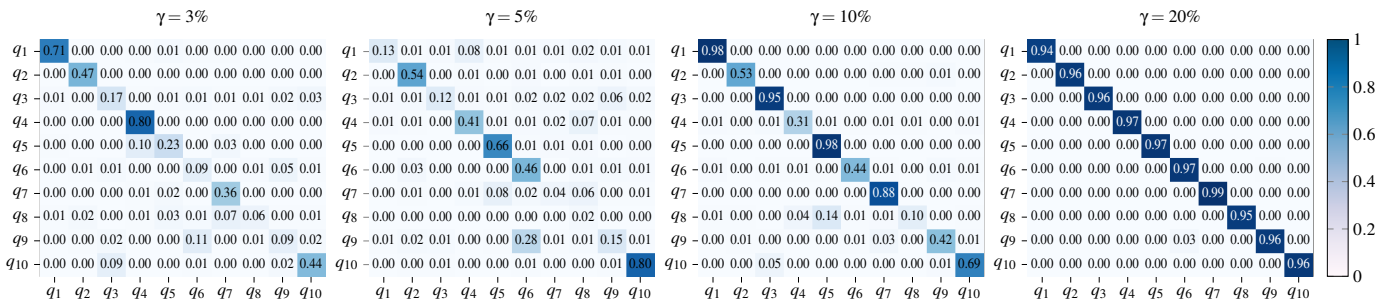


Figure 3: An example of a risk assessment map for 10 easily obfuscated keywords with varying conditions of γ. The x-axis represents the predicted labels, and the y-axis represents the ground truth (original) labels. For instance, (q_1, q_3) indicates the probability that the original label q_3 is predicted as q_1.

Besides these parameters, we set $\varepsilon = 5$ in DCM that balances between achieving sufficient clustering while preventing significant QRR reduction in datasets with diverse distributions.

Learning Algorithms. We selected Catboost [84], one of the most advanced GBDT models, as our base model. Further training parameters are provided in Appendix B.

5.2 Efficacy Analysis of ALERT

In this section, we demonstrate the overall effectiveness of ALERT. We begin by analyzing its single-query risk perception capability through an examination of prediction probabilities for selected queries. Then, we evaluate ALERT across various scenarios and configurations.

5.2.1 An Example of Recovery Map of ALERT

In this section, we present an example from the risk assessment map generated by ALERT, illustrating its ability to visualize and assess potential risks in the query process. The experiment is conducted in the partially known dataset scenario. We assess privacy risks across varying levels of database leakage by testing Ny with 10 easily obfuscated keywords. [2] To simulate varying levels of leakage severity, we adjust the fraction of known data rates, $\gamma \in \{3\%, 5\%, 10\%, 20\%\}$. Figure 3 reports the accessed risks for the selected queries. As expected, higher leakage levels (higher γ) lead to increased privacy risks. For example, for $\gamma = 3\%$, only four keywords show a risk probability greater than 0.4, whereas for $\gamma = 20\%$, all 10 keywords exhibit significantly elevated risk levels.

Interestingly, an increase in leakage does not always correspond to a higher risk. When γ increases from 3% to 5%, the probability of accurately recovering the query q_3 and q_7 decreases. This intriguing result suggests that while higher leakage levels generally expose more information, they may also introduce noise that obscures distinct patterns, making certain keywords harder to recover. Besides, the recovery map highlights ALERT's ability to identify queries with potential risks, even without exact keyword recovery. For q_9 under $\gamma = 5\%$, it shows a relatively high prediction probability despite not being recovered, indicating a potential risk.

[2] The 10 keywords are "lefthand", "setback", "runaround", "toler", "believ", "problema", "americana", "continu", "number", and "bigami."

5.2.2 Results on Partially Known Dataset Scenario

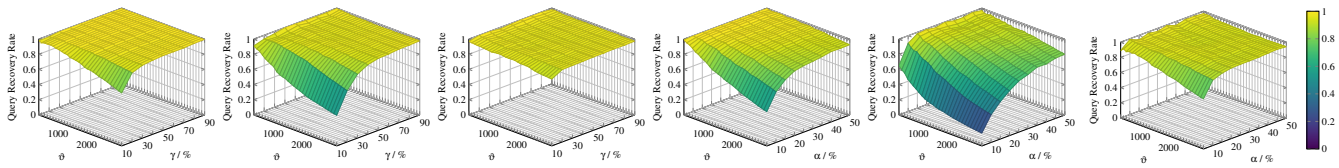
We present our evaluation of ALERT in the partially known dataset scenario. In this experiment, fixed portions of files are extracted from the dataset to form Aux, while the entire database is used to construct Leak. We simulate different levels of adversaries' knowledge by adjusting $\gamma \in \{10\%, 20\%, \ldots, 90\%\}$. To assess the effectiveness of ALERT in recovering various keywords, we vary the number of targeted keywords $\vartheta \in \{100, 200, \ldots, 3000\}$.

The results presented in Figure 4a, 4b, and 4c demonstrate the overall effectiveness of ALERT. In general, ALERT accurately reflects the leakage risk across varying adversarial capacities. For $\gamma = 30\%$, the QRR is 96.1% for En_{3000} and 84.3% for Ny_{3000}, while Wiki_{3000} achieves 95.9%, similar to En_{3000}. Even for $\gamma = 10\%$, En_{3000} and Wiki_{3000} maintain QRRs of 69.8% and 88.1% respectively. This shows ALERT's robustness across different γ. Additionally, keywords with lower popularity are harder to recover due to data distribution instability from having few related documents. For $\gamma = 10\%$, the QRR for Ny_{100} is 91.0%, but drops to 40.3% for Ny_{3000}. It underscores the fact that query recovery relies on a stable distribution formed by a large number of auxiliary documents.

5.2.3 Results on Sampled Dataset Scenario

The sampled dataset scenario presents a greater challenge compared to the partially known dataset scenario, as Aux and Leak contain entirely distinct subsets of files. We divided the files of the dynamic database into two parts under different sampling rates $\alpha \in \{5\%, 10\%, \ldots, 50\%\}$.

The detailed results are shown in Figures 4d, 4e, and 4f. For popular keywords, ALERT maintains a high QRR even with a small portion of the data sampled - achieves a QRR of 99.0% with just $\alpha = 5\%$ in En_{100}. However, the overall recovery performance requires more auxiliary documents to achieve stable results. For En_{3000} and Ny_{3000}, reducing the sampling



(a) En with different γ. (b) Ny with different γ. (c) Wiki with different γ. (d) En with different α. (e) Ny with different α. (f) Wiki with different α.

Figure 4: Query recovery results on efficacy analysis over the partial known dataset and sampled dataset scenario with different data partial known rate γ and sampling data rate α, where γ = 20% indicates that 80% files are randomly selected from the dataset as Aux, all files are utilized as Leak. α = 30% means 30% files serves as the Aux and the rest 70% serves as the Leak.

rate from α = 10% to α = 5% results in sharp QRR drops of 21.5% and 19.7%, respectively. Relatively, Wiki_{3000} demonstrates greater robustness at lower α values due to its larger document numbers. The QRR remains stable from 90.7% to 86.2% even as α decreases from 15% to 10%, showing no significant accuracy degradation. Nonetheless, when α > 20%, all datasets show relatively stable QRR performance, demonstrating ALERT's ability to recover effectively from a stable distribution under the sampled dataset scenario.

5.3 Efficacy Analysis of Keyword Clustering

One of the key designs of ALERT is the utilization of DCM, which employs volumetric information for keyword clustering. We analyze its performance from two perspectives: (1) the efficacy of keyword clustering with fixed parameters and (2) the efficacy of the dynamic clustering mechanism.

Keyword Clustering Results with Fixed Parameters. The purpose of this experiment is to evaluate the effectiveness of keyword clustering using volume information. We consider three key parameters: number of targeted keywords ϑ, file sampling rate α, and keyword clustering threshold θ. Our evaluation encompasses the datasets En_{3000} and Ny_{3000}. We evaluate the results based on the CSR.

The results in Figure 5a show that the CSR initially remains stable as the keyword space increases, but then begins to decline for larger targeted keyword space. Specifically, the CSR for En_{500} and Ny_{500} remains relatively stable, while it drops significantly from 96.4% to 63.6% for En_{3000} and from 92.8% to 60.7% for Ny_{3000}. The reason for this decline is that as the keyword space expands, the disparity in query volume information for each keyword diminishes, making it harder to distinguish. Setting α from 5% to 50%, DCM exhibits robust CSR performance across both datasets (Figure 5b), demonstrating the effectiveness of keyword clustering under varying sampling rates. Figure 5c shows that the CSR improves as the splitting threshold θ increases, but these gains diminish after a certain point. For Ny_{1000}, CSR rises from 28.5% to 93.2% when θ increases from 5 to 100, whereas further increasing θ to 500 yields only an additional 6.2% improvement. These results indicate that careful splitting threshold calibration is key for optimal CSR performance.

Keyword Clustering Results with Dynamic Thresholds. Previous experiments prove the effectiveness of keyword clustering with volume information. Building on the results, we set the predefined clustering threshold as $\psi \in \{50, 100, 200, 500\}$. We then conducted a comparative experiment to evaluate the impact of DCM by comparing QRR and RAL between systems using DCM versus not using it.

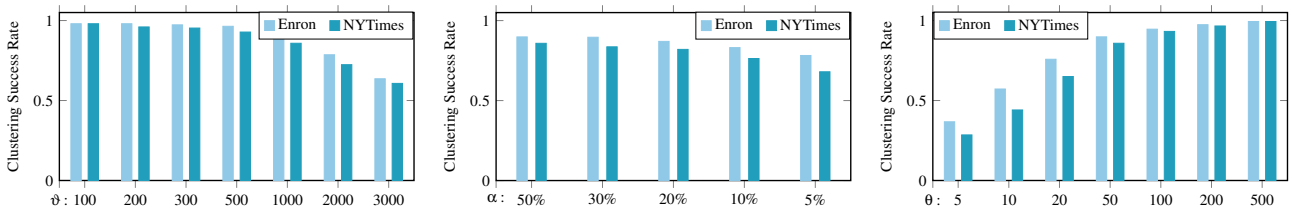
Table 1 presents the results. Even without the dynamic clustering mechanism, ALERT achieves a high QRR of 93.8% under α = 50%, while maintaining a RAL of 101.6s, which is already faster than other LAA methods. This validates the effectiveness of ALERT in accurately assessing query risk. When the DCM mechanism is applied, the QRR decreases by only 0.4%, but the RAL decreases significantly by approximately 65.2%, from 101.6s to 35.4s. This substantial reduction in latency highlights the advantages of using DCM in latency-sensitive scenarios. Further analysis in more challenging scenarios showed that DCM can improve QRR, particularly for larger keyword spaces. This phenomenon highlights the benefits of reducing model complexity through DCM. In summary, while ALERT functions effectively without DCM, the inclusion of the DCM is essential for latency-sensitive applications and scenarios with large keyword spaces.

5.4 Performance across FP/BP-DSSE

Besides generic DSSE schemes, database owners often utilize forward and backward privacy constructions to improve security guarantees for DSSE systems, so we accordingly discuss the ALERT's robustness facing FP/BP-DSSE [17, 24, 48, 95].

Forward privacy ensures that newly added file/keyword pairs cannot be linked by previous query tokens, preventing query tokens from being used to search the newly added entries. Since in ALERT's simulation, the adversary only accesses the most recent database state, our evaluation setting is naturally compatible with forward privacy requirements.

For backward privacy, which protects the privacy of deleted documents, we follow Bost et al.'s [17] classification of three security levels, providing different security and efficiency trade-offs. As illustrated by Xu et al. [106], volume and co-occurrence leakage patterns remain consistent across these levels. Thus, we consider the most challenging backward privacy level in the experiment, where file information deleted prior to the accessed snapshot is completely hidden. To evaluate this setting, we simulate FP/BP-DSSE behavior by varying deletion rates (λ) from 0% to 50% in Leak.



(a) Different ϑ, $\alpha = 50\%$, $\theta = 50$ (b) Different α, $\vartheta = 1000$, $\theta = 50$ (c) Different θ, $\alpha = 50\%$, $\vartheta = 1000$

Figure 5: Keyword clustering results with different parameters. We use CSR to represent the ratio of classifying the keywords into the correct clusters. The experiment is conducted under the sampled dataset scenario. Let $\vartheta \in \{100, 200, 300, 500, 1000, 2000, 3000\}$ be the range of keywords need to be recovered, $\alpha \in \{5\%, 10\%, 20\%, 30\%, 50\%\}$ be the file sampling rates, and $\theta \in \{5, 10, 20, 50, 100, 200, 500\}$ be the splitting thresholds in the experiment. In each experiment, we fix two parameters while varying the third to isolate and show the effect of a single parameter on the results.

Table 1: Performance comparisons of the basic and improved ALERT running over En_{3000} and Ny_{3000} datasets. Here α denotes the sampling rates. To maximize the difference between the difference in QRR and RAL, let $\tilde{\beta} = 1$.

Dataset	En_{3000}					
α	50%		30%		10%	
DCM	×	✓	×	✓	×	✓
QRR (%)	93.8	93.4	86.8	89.9	60.1	66.7
RAL (s)	101.6	35.4	111.0	34.1	141.9	37.2
Dataset	Ny_{3000}					
α	50%		30%		10%	
DCM	×	✓	×	✓	×	✓
QRR (%)	83.7	82.5	68.3	71.4	23.3	36.4
RAL (s)	126.0	33.5	127.9	36.6	121.8	40.1

We conduct an experiment under sample setting ($\alpha = 50\%$) on En_{3000} and Ny_{3000} and demonstrate the results in Figure 6. As λ increases from 0% to 50%, the QRR on En_{3000} drops from 93.1% to 85.3%. And for Ny_{3000}, the QRR reduces from 78.9% to 67.7%. The results show that while increased deletion rates do reduce access pattern leakage, the leakage persists. This shows that even with forward and backward privacy guarantees, DSSE may remain vulnerable to volume (co-occurrence)-based attacks in practical deployments.

5.5 Comparisons with Prior Alternatives

To demonstrate the effectiveness of ALERT, we compare it with three existing methods that, similar to our ALERT, primarily exploit access pattern leakage and achieve state-of-the-art performance: the IHOP attack [82], the Jigsaw attack [76], and the RSA attack [31]. The comparison is performed in the sampled dataset scenario, aligned with the evaluation scenarios described in the original works [31, 76, 82]. Unlike previous LAAs, which primarily focus on query recovery rate, our risk assessment system prioritizes query risk assessment latency while maintaining competitive accuracy in query recovery. To comprehensively evaluate these aspects, we conduct four experiments: (1) restricting the algorithm runtime for each LAA method to compare the performance under low-latency scenarios; (2) comparing the performance without runtime constraints to evaluate the query risk assessment capabilities of each method; (3) evaluating ALERT against Jigsaw with larger keyword universe sizes dataset under similar runtime constraints to evaluate ALERT's scalability on larger datasets; (4) comparing the performance of different methods against SSE countermeasures to evaluate the robustness of each method.

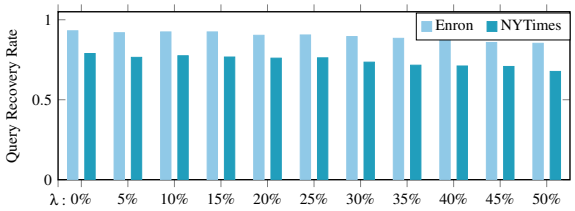
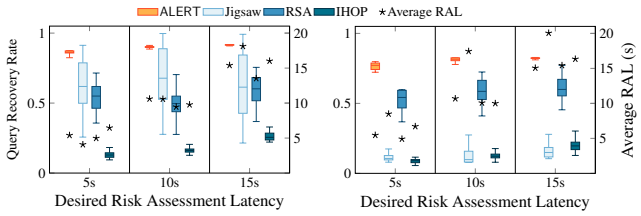


Figure 6: Query recovery results of ALERT with different data deletion rates (λ). Let $\lambda \in \{0\%, 5\%, \cdots, 50\%\}$, where $\lambda = 20\%$ indicates that 20% files are deleted from Leak.

Parameter Settings for Other LAAs. We configure the parameters for the baseline methods following the default settings provided in their respective papers [31, 76, 82]. For the Jigsaw attack, we set $\alpha = 0.3$ and $\beta = 0.9$, with *BaseRec* and *ConfRec* set to 45 and 35, respectively. To control the algorithm runtime, Jigsaw utilizes the *Refspeed* parameter, which we set to 15 in our experiments. In the IHOP attack, we assign p_{free} to 0.25 and n_{iters} to 500. For the RSA attack, we set *refinespeed* to 15 to control the converge runtime and randomly select 5 queries as the ground truth. To ensure a fair comparison, we establish three principles. (1) We disable the frequency information used in both Jigsaw and IHOP, so that all methods access the same auxiliary information under the same leakage conditions. (2) Both ALERT and baseline approaches implement the optimized update strategy defined in Equation 2 to ensure fairness in data preprocessing. (3) All baseline methods are evaluated using only the most in-



(a) En_{3000} (b) Ny_{3000}

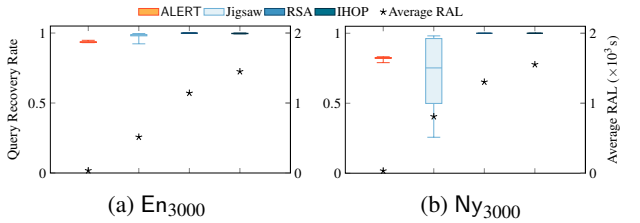
Figure 7: Comparison experiment for En_{3000} and Ny_{3000} under low-latency scenario. Specifically, three desired RAL values are set to {5s, 10s, 15s} for both datasets. To make the RAL of each method close to the desired RALs, we adjust the parameter that controls the convergence speed for each method while keeping other parameters as default. We set Jigsaw's *RefSpeed* to $\{2100, 1800, 1500\}$, RSA's *RefSpeed* to $\{2150, 2070, 2000\}$, IHOP's n_{iters} to $\{2, 3, 5\}$. The stars indicate the average RAL achieved by each method.

formative database snapshot (most recent), as while temporal changes could be utilized by executing static LAAs at different timestamps, existing static methods lack a well-defined methodology for aggregating recovery results across multiple timestamps. Further, such an approach would significantly increase computational overhead, compromising real-time assessment objectives.

5.5.1 Comparisons Under Low-latency Scenario

In this experiment, we compare the QRR of different methods under the low-latency scenario to assess their ability for real-time query risk assessment. We set various RAL expectations and adjust the RALs of each method to achieve similar runtime by tuning specific parameters that influence algorithm runtime while keeping other parameters at their default settings. For Jigsaw and RSA, we adjust *RefSpeed*, while for IHOP, we adjust n_{iters}. This adjustment is necessary because, under their default settings, these SOTA LAA methods are unable to meet the real-time risk assessment objectives. Figure 7 demonstrates the results, where the boxplots represent the distribution of QRR over 30 runs for each method, and the stars indicate the actual average RAL.

Our findings reveal that under the low-latency scenario, only our proposed method, ALERT, consistently achieves a high QRR with stable convergence. This is attributed to the system's design, by learning features from database leakage during this offline training phase, ALERT is equipped to directly predict the risk for each sample during online testing. It mitigates the variability and randomness compared with statistical-based methods in query recovery. Specifically, for En_{3000}, ALERT attains a median QRR of 86.3% with a narrow interquartile range of just 1.5% at a RAL of 5.4s, demonstrating both effectiveness and reliability. In contrast, while Jigsaw does reach a maximum QRR of 91.2% in one instance, its overall median QRR is at 61.9%, and it exhibits a substantial



(a) En_{3000} (b) Ny_{3000}

Figure 8: Comparison experiment for En_{3000} and Ny_{3000} without runtime constraints. Let α be 50%, all other parameters of other LAAs are set as default. Specifically, we set $\tilde{\beta} = 1$ to achieve the maximum query recovery rate of ALERT.

interquartile range of 28.9%. This variability is likely due to the randomness introduced by its iterative recovery step, which causes inconsistent performance across runs. Further, when tested on the more challenging Ny_{3000} scenario, ALERT continues to show robust performance, achieving a median QRR of 77.2% at an average RAL of 5.5s. In contrast, Jigsaw's performance deteriorates sharply in this scenario. This decline is due to Jigsaw's reliance on its volume-based initialization for rapid convergence, which becomes less effective in the scenario where query volumes show higher similarity. While RSA has shown consistent performance in both datasets, it achieves both lower QRR and stability compared to ALERT. IHOP performs poorly in the comparison experiments compared to other methods, likely due to the additional iterations required for convergence caused by its reliance on a random keyword fixing strategy. Moreover, when we further reduced the time limit, ALERT still maintains a 66.8% QRR with a RAL of 1.8s on En_{3000}, whereas all other methods are hard to converge stably under this setting.

5.5.2 Comparisons without Time Constraints

Figure 8 presents the risk assessment results without runtime constraints, demonstrating that ALERT achieves competitive query recovery performance compared to other LAA methods. When compared to Jigsaw, the baseline method achieving the optimal balance between QRR and runtime, ALERT demonstrates a 7.3% higher median QRR on Ny_{3000} while achieving a 5.2% lower median QRR on En_{3000}. More importantly, ALERT exhibits better stability in QRR compared to Jigsaw, which is crucial for a reliable risk estimator. In terms of risk assessment runtime, Jigsaw requires 513.6 seconds to assess the risk of an updated query under the default setting on En_{3000}, which is significantly slower than ALERT's 35.4 seconds, demonstrating a 14.5× speed-up. Furthermore, ALERT achieves a 31.1× speedup with only a 2.0% accuracy loss under the default setting ($\tilde{\beta} = 0.4$). While RSA and IHOP eventually attain high QRR, they require substantially longer runtime to achieve stable convergence - over 1100 and 1400 seconds on En_{3000}. Additionally, RSA operates under a stronger adversarial assumption by requiring known queries for its analysis.

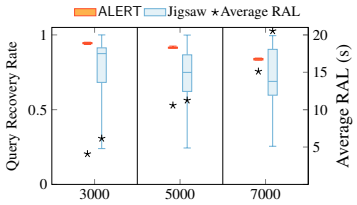
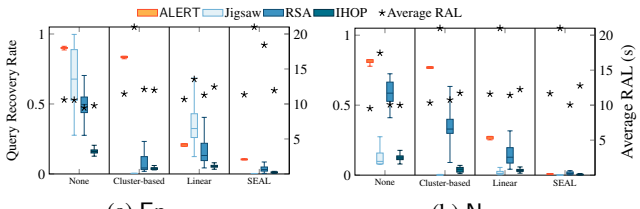


Figure 9: Comparison experiment between ALERT and Jigsaw on the Wikipedia dataset under similar time constraints. The x-axis shows different keyword universe sizes. We set Jigsaw's *RefSpeed* to $\{2500, 4500, 6500\}$ for keyword sizes of $\{3000, 5000, 7000\}$, respectively.



(a) En_{3000} (b) Ny_{3000}

Figure 10: Comparison experiment for En_{3000} and Ny_{3000} against padding countermeasures under similar time constraints. Specifically, to make the RAL of each method close, we adjust the parameter that controls the convergence speed for each method while keeping other parameters as default. We set Jigsaw's *RefSpeed* to 1800, RSA's *RefSpeed* to 2070, IHOP's n_{iters} to 3.

5.5.3 Comparisons in Large Keyword Universe Sizes under Similar Time Constraints

Based on previous experimental results, Jigsaw shows the best performance among LAA methods under real-time risk analysis scenarios besides ALERT. To further validate the robustness of ALERT across different scenarios, we compare its performance against Jigsaw under similar risk assessment latencies for new incoming queries. Based on the stable data distribution of the Wikipedia dataset, we set ALERT with $\tilde{\beta} = 0.2, \tilde{\delta} = 0.2, \tilde{\eta} = 1$ to provide a faster risk assessment, and accordingly adjust Jigsaw's *RefSpeed* parameter to achieve similar runtimes between the two systems. As shown in Figure 9, when scaling the keyword universe from 3000 to 7000, ALERT maintains consistent performance in both efficiency and effectiveness. In contrast, Jigsaw fails to maintain comparable runtime performance even with *RefSpeed* adjusted to near-keyword-size values, preventing further scaling experiments. Moreover, while Jigsaw's median QRR decreases roughly from 87.5% to 68.9% as the keyword universe expands, ALERT demonstrates superior stability, maintaining median QRR between 94.5% and 83.9%, outperforming Jigsaw by roughly 7-15%.

5.5.4 Comparisons Against Countermeasures

We evaluate ALERT and other LAAs against three access-pattern countermeasures: clustering-based padding [98], linear padding [20], and SEAL [33]. Specifically, SEAL is only considered as a padding strategy in our comparison rather than the complete SEAL strategy based on padding and ORAM. Padding countermeasures [20, 33, 98] inject fake documents to increase the query volume according to different strategies. Clustering-based padding [98] specifies α as one cluster unit, and pads all query volumes within the same cluster unit to match the maximum query volume in the cluster. Linear padding [20] injects fake documents to increase the query volume to the nearest multiple of k. SEAL [33] employs a different approach by padding the volume of keywords to the nearest power of an integer x. We conduct experiments on both En_{3000} and Ny_{3000} on sample settings ($\alpha = 50\%$). Similar to the experiment in Section 5.5.1, we set Jigsaw's *RefSpeed* to 1800, RSA's *RefSpeed* to 2070, and IHOP's n_{iters} to 3, testing the real-time recovery capacities of different methods against countermeasures.

Figure 10 shows the QRR of different methods against three padding countermeasures: linear padding ($k = 500$), cluster-based padding ($\alpha = 8$), and SEAL padding ($x = 2$). For cluster-based padding, ALERT achieves a median QRR of 83.3% and 77.0% for En_{3000} and Ny_{3000}, significantly outperforming other methods. This superior performance stems from its analysis of co-occurrence distribution rather than relying on high-volume queries as initial results. While cluster-based padding effectively hides access patterns of high-volume queries within clusters, the relatively consistent volumes of low-volume queries make them more susceptible to co-occurrence analysis. For linear padding [20], ALERT demonstrates consistent performance across both datasets, achieving median QRR of 20.8% and 27.3% on En_{3000} and Ny_{3000} respectively. In comparison, Jigsaw shows more variation between datasets - while achieving a median QRR of 32.5% on En_{3000}, its performance decreases to 1.2% median QRR on Ny_{3000}, suggesting that ALERT may offer more stable performance across different data distributions. For SEAL padding [33], ALERT achieves a modest median QRR of 10.5% on En_{3000}, while other methods show median recovery rates below 5%. The performance gap narrows on Ny_{3000}, where all methods, including ALERT, achieve median QRR around or below 1%, highlighting SEAL's effectiveness in protecting access patterns. This can be attributed to SEAL's padding strategy that extensively pads both high-volume and low-volume queries, though this strong protection comes at the cost of significantly higher storage and communication overhead compared to other countermeasures [76]. It should be noted that our experimental results focus on non-adaptive scenarios, where attackers lack prior knowledge of the applied countermeasures. These results provide a foundation for assessing baseline attack performance. Moreover, prior

works [76, 82] have demonstrated that adaptive attacks, which utilize auxiliary databases processed with countermeasures, can significantly enhance effectiveness. Similarly, ALERT could be improved by retraining on data processed with countermeasures. We acknowledge that evaluation results may vary under adaptive attacks and leave this investigation for future work.

6 Discussion on Future Directions

The above evaluation demonstrates ALERT's ability to provide a reliable real-time risk assessment. ALERT primarily leverages volumetric leakage, such as response sizes and co-occurrence counts of point (aka single keyword-based) queries, to estimate the likelihood of the keyword underlying a query on datasets with keyword-based inverted index. This characteristic makes ALERT applicable for end-to-end encrypted systems (e.g. SSE) that preserve volume information after encryption. However, it would make random guesses among all candidates for systems if all such features are hidden (e.g., volume-hiding ORAM).

Leakage Analysis with Alternative Warning Models. We acknowledge that there are other types of queries and corresponding leakages in the realm of SSE. For example, conjunctive and boolean queries also reveal query-specific leakage. While our current model focuses on point queries, it is extensible to these query types, as they similarly expose volumetric or co-occurrence-based leakage. Meanwhile, we note that most current encrypted search algorithms reveal query equality, which denotes whether two queries are for the same keyword. By treating this leakage as features analogous to response size or co-occurrence, our framework can be generalized to support a broader range of scenarios.

Additionally, SSE schemes that extend beyond inverted index-based datasets, such as order-revealing encryption (ORE) applied to non-indexed datasets, introduce distinct leakage patterns. For instance, ORE-based schemes may leak information about the relative order of encrypted values, which can be exploited by adversaries to infer plaintext relationships. While our model could theoretically adapt to these scenarios by treating order properties as features, how to effectively model and represent these properties as input features remains unclear and requires further exploration. We leave this as an open problem for future research.

Leakage Analysis with Different Data Sensitivity. In this paper, we treat all keywords as having equal importance in our analysis like other LAAs. In practice, different keywords carry varying levels of data criticality. Some keywords may be associated with highly sensitive information (e.g., financial data, personal identifiers), while others might represent less critical keywords. Future research should systematically incorporate these varying sensitivity levels of keywords into the risk assessment framework.

Leakage Analysis with Different Auxiliary Assumptions. This paper follows the common assumption in SSE that the adversary possesses a similar plaintext dataset (auxiliary dataset) to the target dataset. In fact, the auxiliary dataset itself may be sensitive and subject to strict access restrictions. The limited query access may lead to large inconsistency between auxiliary and target query space, affecting the reliability of risk assessment results. Future research should focus on analyzing the impact caused by different auxiliary dataset assumptions and how to mitigate this impact.

Leakage Analysis with Other Models. In this work, we assess query leakage using GBDT models. Extending our analysis to other advanced models could yield further insights. For example, language models are able to provide semantic explanations for specific query leakages, offering deeper understanding and enabling more targeted client interactions. Another potential direction is the adaptation of our system to different data representations. For instance, databases formulated as graphs [12] might benefit from the applications of graph neural networks (GNNs) [86] for leakage analysis.

7 Related Works

Dynamic Searchable Symmetric Encryption. SSE is proposed for secure and efficient query searching in encrypted databases. The SSE for basic query is proposed by Song et al. [93] and Curtmola et al. [30]. Then, some SSE schemes are proposed for range query [14, 23, 35], boolean query [22, 52] and specific query type [51, 55, 66]. To support dynamic databases, a series of studies have focused on dynamic searchable symmetric encryption (DSSE), particularly on higher efficiency [15, 56, 57, 72] , stronger security [58, 75, 97, 97, 100] and specific structure [35, 39, 40, 66].

Leakage Analysis on SSE. Though SSE schemes provide efficient solutions for encrypted search, they are proven to be vulnerable to leakage-abuse attacks (LAAs). A series of LAAs [20, 37, 43–46, 50, 63–65, 74, 77, 81–83, 105] have been developed against different adversarial assumptions and SSE schemes. Being aware of these attacks, recent works aim to theoretically analyze the impact of revealing these leakages in reality. Kornaropoulos et al. [62] introduced leakage inversion, using the entropy of the reconstruction space to provide insights into the security of SSE schemes. Moreover, Kamara et al. [54] employed Bayesian networks to demonstrate the persistence of leakage in SSE schemes and established generic bounds for both target and auxiliary databases. Boldyreva et al. [13] analyzed the SSE scheme's security by quantifying adversarial success, and they offered diverse functions to express more fine-grained goals on designing SSE schemes.

Countermeasures Against LAAs in SSE. Several works have been proposed to thwart LAAs by suppressing leakage, focusing on protecting specific subsets of leakage patterns, such as volume and co-occurrence patterns. A series of studies [9, 16, 20, 53] has concentrated on concealing the volume pattern in encrypted databases. Additionally, some works

focus on protecting the co-occurrence leakage pattern [26, 40, 88, 104] by inserting virtual documents in database construction or querying process. Beyond these countermeasures, some works have introduced oblivious RAM (ORAM) [11, 25, 34, 36, 47, 55, 67, 71, 94], which can fully hide the query access patterns. While given significant computation and communication overhead, recent hardware-supported implementations [36, 47] have achieved relatively acceptable performance while maintaining strong security guarantees, making them a viable option for security-critical scenarios. In contrast to defense approaches, ALERT operates as an opt-in plugin that, without modifying SSE schemes, analyzes leakage and alerts clients to potential vulnerabilities in real-time.

Machine Learning-Enhanced Encrypted Flow Analysis. Encrypted traffic pattern analysis aims to construct fingerprints for clients from encrypted traffic signals. Currently, there has been a long series of works [8, 10, 18, 28, 29, 38, 49, 70, 73, 89–92, 99, 103] in many scenarios. Schuster et al. [87] accurately fingerprint the encrypted video streamings on video sites under an open-world assumption. ALERT reveals that auxiliary information in encrypted database querying can be modeled as fingerprints, leading us to introduce flow analysis methods for query risk assessment.

8 Conclusion

We propose ALERT, a real-time risk assessment system for SSE schemes that clusters keywords into several groups and generates risk assessment maps for queries within each cluster. By formulating risk assessment as a multi-classification problem and leveraging a novel machine learning-enhanced pipeline, ALERT delivers accurate risk evaluations in real-time, assessing potential leakage in seconds, thereby enhancing the feasibility of integrating leakage analysis tolls into the practical use of SSE schemes. Our experimental results demonstrate ALERT's robust performance across various datasets and adversary assumptions. Under the low-latency scenario, ALERT consistently outperforms existing leakage analysis approaches [31, 76, 82]. Even with a stringent time limitation of 1.8 seconds, ALERT maintains a recovery rate of 66.8%, showcasing its efficiency. Moreover, ALERT's performance advantage persists even when facing various countermeasures, particularly with cluster-based padding, where it shows only a 6.8% decrease in median recovery rate compared to scenarios without countermeasures.

Acknowledgement

The authors sincerely thank the reviewers for their invaluable feedback. This work was supported in part by the National Natural Science Foundation of China under Grant No.62202228, by the Youth Science and Technology Talents Lifting Project of Jiangsu Association of Science and Technology JSTJ-2024-163, by the Fundamental Research Funds for the Central Universities No.30923011023, by the Research Grants Council of Hong Kong under Grants CityU 11218322, 11219524, R6021-20F, R1012-21, RFS2122-1S04, C2004-21G, C1029-22G, C6015-23G, and N_CityU139/21 and in part by the Innovation and Technology Commission of Hong Kong (ITC) under Mainland-Hong Kong Joint Funding Scheme (MHKJFS) under Grant MHP/135/23. This work was also supported by the InnoHK initiative, the Government of the HKSAR, and the Laboratory for AI-Powered Financial Technologies (AIFT). Cong Wang is the corresponding author of this paper.

Ethics Considerations

Throughout our study, we have conducted a thorough ethical assessment of our research methodology and potential impacts. Our investigation does not involve human participants, personal information, or any sensitive data. The research relies exclusively on three publicly available datasets for experimental evaluation, neither of which contains any personally identifiable information. Therefore, there are no ethical concerns in this paper.

Open Science Statement

As part of our commitment to open science, we have released our code and materials for artifact evaluation. Our materials include:

- Full source code of ALERT, covering data preprocessing, training, and risk assessment modules. The "src" folder contains all components of our system, including data preprocessing, training, risk assessment, and data post-processing modules.
- Comprehensive training and test datasets. We would provide all needed dataset files about three datasets (Enron, NYTimes, and Wikipedia).
- Detailed configuration files and instructions for result reproduction. A comprehensive README is provided to guide users through the setup and reproduction process, ensuring full functionality of our main findings. The README.md can be found in the project root directory.
- Analysis scripts for the complete pipeline: preprocessing, model training, risk assessment, and result analysis. The "scripts" folder contains all the necessary scripts for reproducing our experimental results. For instance, "main_recovery.sh" is used to reproduce the results presented in Figure 4. The README file provides a detailed mapping between scripts and paper results.

References

[1] Aws database encryption sdk. https://docs.aws.amazon.com/database-encryption-sdk/latest/devguide/searchableencryption.html.

[2] Enron Email Dataset. http://www.cs.cmu.edu/ enron/.

[3] Google safe browsing. https://safebrowsing.google.com.

[4] macos malware protection. https://support.apple.com/guide/security/sec469d47bd8/web.

[5] Microsoft defender smartscreen. https://learn.microsoft.com/en-us/windows/security/operating-system-security/virus-and-threat-protection/microsoft-defender-smartscreen/.

[6] Queryable encryption: Mongodb manual. https://www.mongodb.com/docs/manual/core/queryable- encryption/.

[7] Windows user account control. https://learn.microsoft.com/en-us/windows/security/application-security/application-control/user-account-control/.

[8] Iman Akbari, Mohammad A. Salahuddin, Leni Ven, Noura Limam, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau, and Stéphane Tuffin. A look behind the curtain: Traffic classification in an increasingly encrypted web. *Proc. ACM Meas. Anal. Comput. Syst.*, 5(1):04:1–04:26, 2021.

[9] Ghous Amjad, Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption. *Proc. Priv. Enhancing Technol.*, 2023(1):417–436, 2023.

[10] Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello, Fernando M. V. Ramos, and André Madeira. Flowlens: Enabling efficient flow classification for ml-based network security applications. In *Proc. of NDSS*, 2021.

[11] Vincent Bindschaedler, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, and Yan Huang. Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward. In *Proc. of ACM CCS*, 2015.

[12] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *Proc. of NDSS*, 2020.

[13] Alexandra Boldyreva, Zichen Gui, and Bogdan Warinschi. Understanding leakage in searchable encryption: a quantitative approach. *Proc. Priv. Enhancing Technol.*, 2024(4):503–524, 2024.

[14] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *Proc. of EUROCRYPT*, 2015.

[15] Angèle Bossuat, Raphael Bost, Pierre-Alain Fouque, Brice Minaud, and Michael Reichle. SSE and SSD: page-efficient searchable symmetric encryption. In *Proc. of CRYPTO*, 2021.

[16] Raphael Bost and Pierre-Alain Fouque. Thwarting leakage abuse attacks against searchable encryption - A formal approach and applications to database padding. *IACR ePrint.*, 2017.

[17] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proc. of ACM CCS*, 2017.

[18] Jiahao Cao, Zijie Yang, Kun Sun, Qi Li, Mingwei Xu, and Peiyi Han. Fingerprinting SDN applications via encrypted control traffic. In *Proc. of RAID*, 2019.

[19] Shaosheng Cao, Xinxing Yang, Cen Chen, Jun Zhou, Xiaolong Li, and Yuan Qi. Titant: Online real-time transaction fraud detection in ant financial. *Proc. VLDB Endow.*, 12(12):2082–2093, 2019.

[20] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proc. of ACM CCS*, 2015.

[21] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Proc. of NDSS*, 2014.

[22] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Proc. of CRYPTO*, 2013.

[23] David Cash, Feng-Hao Liu, Adam O'Neill, Mark Zhandry, and Cong Zhang. Parameter-hiding order revealing encryption. In *Proc. of ASIACRYPT*, 2018.

[24] Javad Ghareh Chamani, Dimitrios Papadopoulos, Mohammadamin Karbasforushan, and Ioannis Demertzis. Dynamic searchable encryption with optimal search in the presence of deletions. In *Proc. of USENIX Security*, 2022.

[25] T.-H. Hubert Chan, Kai-Min Chung, Bruce M. Maggs, and Elaine Shi. Foundations of differentially oblivious algorithms. In *Proc. of SIAM SODA*, 2019.

[26] Guoxing Chen, Ten-Hwang Lai, Michael K. Reiter, and Yinqian Zhang. Differentially private access patterns for searchable symmetric encryption. In *Proc. of IEEE INFOCOM*, 2018.

[27] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proc. of ACM KDD*, 2016.

[28] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. Online website fingerprinting: Evaluating website fingerprinting attacks on tor in the real world. In *Proc. of USENIX Security*, 2022.

[29] Scott E. Coull and Kevin P. Dyer. Traffic analysis of encrypted messaging services: Apple imessage and beyond. *Comput. Commun. Rev.*, 44(5):5–11, 2014.

[30] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proc. of ACM CCS*, 2006.

[31] Marc Damie, Florian Hahn, and Andreas Peter. A highly accurate query-recovery attack against searchable encryption using non-indexed documents. In *Proc. of USENIX Security*, 2021.

[32] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. Dynamic searchable encryption with small client storage. In *Proc. of NDSS*, 2020.

[33] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. SEAL: attack mitigation for encrypted databases via adjustable leakage. In *Proc. of USENIX Security*, 2020.

[34] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. SEAL: attack mitigation for encrypted databases via adjustable leakage. In *Proc. of USENIX Security*, 2020.

[35] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos N. Garofalakis. Practical private range search revisited. In *Proc. of ACM SIGMOD*, 2016.

[36] Saba Eskandarian and Matei Zaharia. Oblidb: Oblivious query processing for secure databases. *Proc. VLDB Endow.*, 13(2):169–183, 2019.

[37] Francesca Falzon, Evangelia Anna Markatou, Akshima, David Cash, Adam Rivkin, Jesse Stern, and Roberto Tamassia. Full database reconstruction in two dimensions. In *Proc. of ACM CCS*, 2020.

[38] Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. Realtime robust malicious traffic detection via frequency domain analysis. In *Proc. of ACM CCS*, 2021.

[39] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In *Proc. of CRYPTO*, 2016.

[40] Marilyn George, Seny Kamara, and Tarik Moataz. Structured encryption and dynamic leakage suppression. In *Proc. of EUROCRYPT*, 2021.

[41] Eleonora Giunchiglia and Thomas Lukasiewicz. Coherent hierarchical multi-label classification networks. In *Proc. of NeurIPS*, 2020.

[42] Nayanaba Pravinsinh Gohil and Arvind D. Meniya. Click ad fraud detection using xgboost gradient boosting algorithm. In *Proc. of COMS2*, 2021.

[43] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *Proc. of ACM CCS*, 2018.

[44] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *Proc. of IEEE S&P*, 2019.

[45] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *Proc. of IEEE S&P*, 2017.

[46] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. Encrypted databases: New volume attacks against range queries. In *Proc. of ACM CCS*, 2019.

[47] Thang Hoang, Muslum Ozgur Ozmen, Yeongjin Jang, and Attila A. Yavuz. Hardware-supported ORAM in effect: Practical oblivious search and update on very large dataset. *Proc. Priv. Enhancing Technol.*, 2019(1):172–191, 2019.

[48] Thang Hoang, Attila A. Yavuz, and Jorge Guajardo. A secure searchable encryption framework for privacy-critical cloud storage services. *IEEE Trans. Serv. Comput.*, 14(6):1675–1689, 2021.

[49] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. New directions in automated traffic analysis. In *Proc. of ACM CCS*, 2021.

[50] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Proc. of NDSS*, 2012.

[51] Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Outsourced symmetric private information retrieval. In *Proc. of ACM CCS*, 2013.

[52] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Proc. of EUROCRYPT*, 2017.

[53] Seny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In *Proc. of EUROCRYPT*, 2019.

[54] Seny Kamara and Tarik Moataz. Bayesian leakage analysis: A framework for analyzing leakage in encrypted search. *IACR ePrint.*, 2023.

[55] Seny Kamara, Tarik Moataz, and Olga Ohrimenko. Structured encryption and leakage suppression. In *Proc. of CRYPTO*, 2018.

[56] Seny Kamara, Tarik Moataz, Andrew Park, and Lucy Qin. A decentralized and encrypted national gun registry. In *Proc. of IEEE S&P*, 2021.

[57] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Proc. of FC*, 2013.

[58] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proc. of ACM CCS*, 2012.

[59] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Proc. of NeurIPS*, 2017.

[60] Samira Khodabandehlou and Alireza Hashemi Golpayegani. Fifraud: Unsupervised financial fraud detection in dynamic graph streams. *ACM Trans. Knowl. Discov. Data*, 18(5):111:1–111:29, 2024.

[61] Christoph Koch, Yanif Ahmad, Oliver Kennedy, Milos Nikolic, Andres Nötzli, Daniel Lupei, and Amir Shaikhha. Dbtoaster: higher-order delta processing for dynamic, frequently fresh views. *VLDB J.*, 23(2):253–278, 2014.

[62] Evgenios M. Kornaropoulos, Nathaniel Moyer, Charalampos Papamanthou, and Alexandros Psomas. Leakage inversion: Towards quantifying privacy in searchable encryption. In *Proc. of ACM CCS*, 2022.

[63] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution. In *Proc. of IEEE S&P*, 2020.

[64] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. Response-hiding encrypted ranges: Revisiting security via parametrized leakage-abuse attacks. In *Proc. of IEEE S&P*, 2021.

[65] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *Proc. of IEEE S&P*, 2018.

[66] Shangqi Lai, Sikhar Patranabis, Amin Sakzad, Joseph K. Liu, Debdeep Mukhopadhyay, Ron Steinfeld, Shifeng Sun, Dongxi Liu, and Cong Zuo. Result pattern hiding searchable encryption for conjunctive queries. In *Proc. of ACM CCS*, 2018.

[67] Kasper Green Larsen and Jesper Buus Nielsen. Yes, there is an oblivious RAM lower bound! In *Proc. of CRYPTO*, 2018.

[68] Ranran Li, Zhaowei Liu, Yuanqing Ma, Dong Yang, and Shuaijie Sun. Internet financial fraud detection based on graph learning. *IEEE Trans. Comput. Soc. Syst.*, 10(3):1394–1401, 2023.

[69] Xiaoliang Ling, Weiwei Deng, Chen Gu, Hucheng Zhou, Cui Li, and Feng Sun. Model ensemble for click prediction in bing search ads. In *Proc. of ACM WWW*, 2017.

[70] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. Fs-net: A flow sequence network for encrypted traffic classification. In *Proc. of IEEE INFOCOM*, 2019.

[71] Matteo Maffei, Giulio Malavolta, Manuel Reinert, and Dominique Schröder. Privacy and access control for outsourced personal records. In *Proc. of IEEE S&P*, 2015.

[72] Brice Minaud and Michael Reichle. Dynamic local searchable symmetric encryption. In *Proc. of CRYPTO*, 2022.

[73] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *Proc. of NDSS*, 2018.

[74] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *Proc. of ACM CCS*, 2015.

[75] Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. Dynamic searchable encryption via blind storage. In *Proc. of IEEE S&P*, 2014.

[76] Hao Nie, Wei Wang, Peng Xu, Xianglong Zhang, Laurence T. Yang, and Kaitai Liang. Query recovery from easy to hard: Jigsaw attack against SSE. In *Proc. of USENIX Security*, 2024.

[77] Jianting Ning, Xinyi Huang, Geong Sen Poh, Jiaming Yuan, Yingjiu Li, Jian Weng, and Robert H. Deng. LEAP: leakage-abuse attack on efficiently deployable, efficiently searchable encryption with partially known dataset. In *Proc. of ACM CCS*, 2021.

[78] US Department of Health and Human Services. New health insurance portability and accountability act (hipaa) regulations. https://www.hipaajournal.com/new-hipaa-regulations/.

[79] National Institute of Standards and Technology. Risk management guide for information technology systems. https://www.hhs.gov/sites/default/files/ocr/privacy/hipaa/administrative/securityrule/nist800-30.pdf.

[80] Aomar Osmani, Massinissa Hamidi, and Pegah Alizadeh. Clustering approach to solve hierarchical classification problem complexity. In *Proc. of AAAI*, 2022.

[81] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In *Proc. of USENIX Security*, 2021.

[82] Simon Oya and Florian Kerschbaum. IHOP: improved statistical query recovery against searchable symmetric encryption through quadratic optimization. In *Proc. of USENIX Security*, 2022.

[83] Yanjun Pan, Alon Efrat, Ming Li, Boyang Wang, Hanyu Quan, Joseph S. B. Mitchell, Jie Gao, and Esther M. Arkin. Data inference from encrypted databases: a multi-dimensional order-preserving matching approach. In *Proc. of ACM MobiHoc*, 2020.

[84] Liudmila Ostroumova Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In *Proc. of NeurIPS*, 2018.

[85] Farzad Samie, Lars Bauer, and Jörg Henkel. Hierarchical classification for constrained iot devices: A case study on human activity recognition. *IEEE Internet Things J.*, 7(9):8287–8295, 2020.

[86] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.

[87] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *Proc. of USENIX Security*, 2017.

[88] Zhiwei Shang, Simon Oya, Andreas Peter, and Florian Kerschbaum. Obfuscated access and search patterns in searchable encryption. In *Proc. of NDSS*, 2021.

[89] Meng Shen, Yiting Liu, Liehuang Zhu, Xiaojiang Du, and Jiankun Hu. Fine-grained webpage fingerprinting using only packet length information of encrypted traffic. *IEEE Trans. Inf. Forensics Secur.*, 16:2046–2059, 2021.

[90] Meng Shen, Jinpeng Zhang, Liehuang Zhu, Ke Xu, and Xiaojiang Du. Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. *IEEE Trans. Inf. Forensics Secur.*, 16:2367–2380, 2021.

[91] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proc. of ACM CCS*, 2018.

[92] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proc. of ACM CCS*, 2019.

[93] Dawn Xiaodong Song, David A. Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proc. of IEEE S&P*, 2000.

[94] Emil Stefanov, Marten van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. *J. ACM*, 65(4):18:1–18:26, 2018.

[95] Shifeng Sun, Ron Steinfeld, Shangqi Lai, Xingliang Yuan, Amin Sakzad, Joseph K. Liu, Surya Nepal, and Dawu Gu. Practical non-interactive searchable encryption with forward and backward privacy. In *Proc. of NDSS*, 2021.

[96] New York Times. Nytimes article dataset. https://archive.ics.uci.edu/ml/datasets/bag+of+words.

[97] Viet Vo, Shangqi Lai, Xingliang Yuan, Shifeng Sun, Surya Nepal, and Joseph K. Liu. Accelerating forward and backward private searchable encryption using trusted execution. In *Proc. of ACNS*, 2020.

[98] Viet Vo, Xingliang Yuan, Shi-Feng Sun, Joseph K. Liu, Surya Nepal, and Cong Wang. Shielddb: An encrypted document database with padding countermeasures. *IEEE Trans. Knowl. Data Eng.*, 35(4):4236–4252, 2023.

[99] Tao Wang. High precision open-world website fingerprinting. In *Proc. of IEEE S&P*, 2020.

[100] Tianhao Wang and Yunlei Zhao. Secure dynamic SSE via access indistinguishable storage. In *Proc. of ACM AsiaCCS*, 2016.

[101] Jonatas Wehrmann, Ricardo Cerri, and Rodrigo C. Barros. Hierarchical multi-label classification networks. In *Proc. of ICML*, 2018.

[102] Wikipedia. Wikipedia encyclopedia entries dataset. https://zenodo.org/records/12683869.

[103] Renjie Xie, Jiahao Cao, Enhuan Dong, Mingwei Xu, Kun Sun, Qi Li, Licheng Shen, and Menghao Zhang. Rosetta: Enabling robust TLS encrypted traffic classification in diverse network environments with tcp-aware traffic augmentation. In *Proc. of USENIX Security*, 2023.

[104] Lei Xu, Huayi Duan, Anxin Zhou, Xingliang Yuan, and Cong Wang. Interpreting and mitigating leakage-abuse attacks in searchable symmetric encryption. *IEEE Trans. Inf. Forensics Secur.*, 16:5310–5325, 2021.

[105] Lei Xu, Xingliang Yuan, Cong Wang, Qian Wang, and Chungen Xu. Hardening database padding for searchable encryption. In *Proc. of IEEE INFOCOM*, 2019.

[106] Lei Xu, Leqian Zheng, Chengzhi Xu, Xingliang Yuan, and Cong Wang. Leakage-abuse attacks against forward and backward private searchable symmetric encryption. In *Proc. of ACM CCS*, 2023.

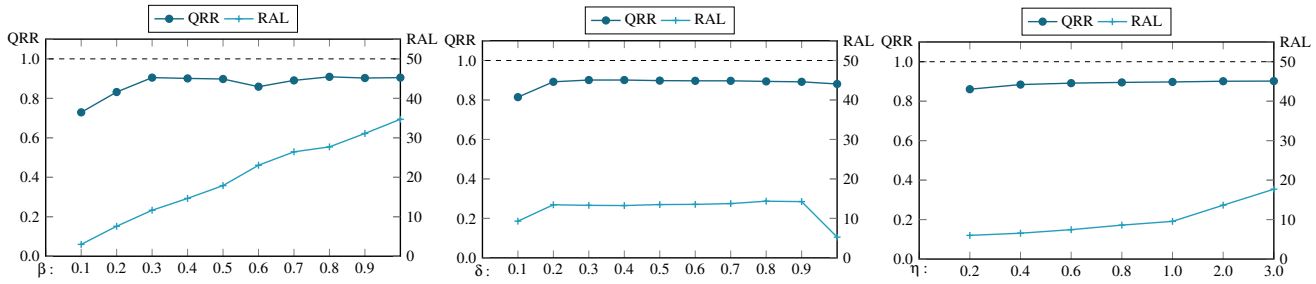
[107] Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. Hierarchically structured meta-learning. In *Proc. of ICML*, 2019.

A Parameter Analysis

To validate the optimality of our configurations, we conducted a comprehensive analysis of the parameters of ALERT. Specifically, we focused on three parameters in our risk assessment process: the proportion of selected database snapshots (β), the proportion of data selected for each sample in augmentation (δ), and the augmentation factor representing the total number of augmented samples generated for each class (η). For the experiment, we adjust the parameters in the risk assessment process while keeping the parameters as default in the classifier training. We repeat each experiment 30 times and take the average as the final result.

A.1 Proportion of Selected Database Snapshots β

The selected portion of database snapshots can affect the representativeness of the raw data used for risk assessment. If the β value is set too low, ALERT may produce inaccurate risk assessments because the raw data distribution may not accurately reflect the actual database distribution. Conversely, a high β value increases risk assessment latency, negatively impacting the real-time performance of the risk assessment. To evaluate the influence of β on RAL and average QRR, we experimented with various portion values. Specifically, we let $\beta \in \{0.1, 0.2, \ldots, 1.0\}$. Figure 11a shows the results. We can find that β and RAL increase approximately linearly because it affects both the time of data preparation and classifier prediction. Our experiments identified that a β value between 0.3 and 0.5 provides an optimal balance between QRR and RAL, leading us to select 0.4 as the default setting.



(a) Different β, $\delta = 0.4$, $\eta = 2$. (b) Different δ, $\beta = 0.4$, $\eta = 2$. (c) Different η, $\beta = 0.4$, $\delta = 0.4$.

Figure 11: Results on different parameter (β, δ, η) settings under sampled dataset scenario ($\alpha = 50\%$) on En_{3000}.

A.2 Proportion of Data Selected for Each Sample in Augmentation δ

The proportion of data selected plays a crucial role in mitigating the impact of noisy data. Due to the inherent volatility of keyword distribution over different timestamps, the absence of appropriate sampling techniques may lead to degraded model performance due to the influence of noisy data. To comprehensively elucidate the effect of the selected proportion, we systematically varied δ across a range of percentages. We empirically evaluated several predefined values of $\delta = \{0.1, 0.2, \ldots, 1.0\}$ and present the results in Figure 11b. Our findings indicate that an excessively low value of δ fails to effectively filter noisy data, while an excessively high δ unnecessarily prolongs data preprocessing time. We observed that setting δ between 0.2 and 0.6 yielded optimal performance in our experimental context. Consequently, we set δ at 0.4 in our large-scale experiments.

A.3 Augmentation Factor: Number of Samples Generated Per Class η

In the Random Repeated Sampling (RRS) process, η is a critical parameter that determines the extent of data augmentation. Specifically, it represents the augmentation factor, indicating the multiple of augmented data generated relative to the original raw data. In our experiment, we systematically vary η across a range of predefined values $\eta \in \{0.2, 0.4, 0.6, 0.8, 1.0, 2.0, 3.0\}$ and present the results in Figure 11c. Our analysis reveals that larger eta leads to stable QRR improvement but also increases the classifier prediction time. Therefore, we set η to 2, which can stabilize the convergence to optimal effect under a low latency scenario. If the latency requirements are more stringent, the client may choose to reduce the value of η accordingly.

B Machine Learning Model Training

In this paper, we train multiple classifiers to serve as risk estimators. Here, we detail the specific training parameters employed. Our classifier is configured with 1×10^3 iterations, a learning rate of 0.1, and a tree depth of 6 to balance performance and model complexity. Regularization is applied using L2 leaf regulation with a parameter value of 3, and numerical features are discretized into 128 bins for more efficient processing. We introduce randomness into the training process with a random strength of 1 and control the bagging intensity with a temperature of 0.8. To ensure that each leaf has sufficient data, a minimum of 5 samples per leaf is enforced. For each classifier, we randomly shuffle the training samples and apply a 75%-25% train-validation split. We utilize the early stop strategy to stop training if the validation loss does not decrease after 5 epochs. Additionally, we accelerate training using GPU resources and employ a softmax cross-entropy loss function.