Leakage-Resilient Easily Deployable and Efficiently Searchable Encryption (EDESE)

Jiaming Yuan
Yingjiu Li
Jun Li
jiamingy@uoregon.edu
yingjiul@uoregon.edu
lijun@cs.uoregon.edu
Department of Computer Science
University of Oregon
Eugene, USA

Daoyuan Wu
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Hong Kong, China
daoyuan@cse.ust.hk

Jianting Ning*
School of Cyber Science and Engineering
Wuhan University
Wuhan, China
Faculty of Data Science
City University of Macau
Macau, China
jtning88@gmail.com

Yangguang Tian
School of Computer Science and Electronic Engineering
University of Surrey
Guildford, UK
yangguang.tian@surrey.ac.uk

Robert H. Deng
School of Computing & Information Systems
Singapore Management University
Singapore, Singapore
robertdeng@smu.edu.sg

Abstract

Easily Deployable and Efficiently Searchable Encryption (EDESE) is a cryptographic primitive designed for practical searchable applications, offering efficient search and easy deployment. However, it remains vulnerable to Leakage-Abuse attacks, allowing adversaries to exploit keyword-matching processes to extract sensitive information. To address these vulnerabilities, we introduce Leakage-Resilient EDESE (LR-EDESE) with k-indistinguishability and controlled leakage functions. We then propose Volume Leakage-Resilient EDESE (VLR-EDESE), a new scheme to protect against both query and document volume leakage. Our experimental results demonstrate that at $k = 5000$ (maximum security setting), VLR-EDESE incurs an overhead of 63× compared to the baseline EDESE without leakage protection, outperforming state-of-the-art methods with 320× and 97× overhead, respectively. For smaller k values (10, 20, 50, 100), storage and communication overhead remain within 2× and 2.5× of the baseline EDESE, highlighting VLR-EDESE's flexibility. Finally, we present CloudSec, an implementation of VLR-EDESE that seamlessly integrates with cloud storage platforms, using OneDrive as an example.

CCS Concepts

• **Security and privacy → Cryptography**.

Keywords

Symmetric Searchable Encryption, Leakage-Abuse Attack Defense, Leakage-Resilient, EDESE

ACM Reference Format:
Jiaming Yuan, Yingjiu Li, Jun Li, Daoyuan Wu, Jianting Ning, Yangguang Tian, and Robert H. Deng. 2025. Leakage-Resilient Easily Deployable and Efficiently Searchable Encryption (EDESE). In *Proceedings of the 30th ACM Symposium on Access Control Models and Technologies (SACMAT '25), July 8–10, 2025, Stony Brook, NY, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3734436.3734445

*Jianting Ning is the corresponding author.

SACMAT '25, Stony Brook, NY, USA

ACM ISBN 979-8-4007-1503-7/2025/07
https://doi.org/10.1145/3734436.3734445

1 Introduction

Easily Deployable and Efficiently Searchable Encryption. The Easily Deployable and Efficiently Searchable Encryption (EDESE) is a promising cryptographic primitive designed specifically for practical symmetric searchable encryption (SSE) applications [3, 13, 20, 28, 31]. It offers two advantages: *easy deployment* and *efficient search*. Compared to existing SSEs, easy deployment makes EDESE the optimal choice for protecting existing searchable services.

The easy deployment of EDESE stems from minimal server involvement: the server only stores encrypted data and performs simple matching. When a client sends a query token for a keyword, the server matches it directly against stored ciphertexts using bitwise comparison—mirroring the process on unencrypted data. This compatibility enables seamless migration from existing systems to EDESE-based searchable services. It leads to significantly reduced deployment and maintenance costs, attracting both individual and enterprise customers.

Another key benefit of EDESE is its remarkable search efficiency on the server side, which leverages the inherent efficiency of SSE. Due to the streamlined matching process employed by EDESE, the search performance on the server side closely resembles that of searching over plaintext. In contrast, most existing searchable encryption schemes require the server to run complex test algorithms,

which may involve decryption or other intricate mathematical operations For example, PIR-protected schemes [1, 14] require complex computations on the server side. Consequently, EDESE outperforms other searchable encryption schemes as the most efficient searchable encryption solution in practical scenarios.

Leakage-Abuse Attack. EDESE is vulnerable to Leakage-Abuse attacks (LAAs), widely studied in numerous research works [2, 4, 12, 15, 19, 21, 23, 24, 26]. These attacks leverage leaked information from searchable encryption schemes, including EDESE, to infer sensitive data with the help of auxiliary information. In the context of searchable encryption literature, "leakage" refers to any information that an attacker can discern about query tokens, encrypted documents, and query responses. The auxiliary information consists of a partial or complete set of source documents or a set of documents with a distribution similar to that of the source documents.

LAAs encompass various types of leakage profiles, such as access patterns, search patterns, volume patterns, co-occurrence patterns, etc. Specifically, the access pattern leakage exposes which encrypted document is associated with a query token. Search pattern leakage implies whether two query tokens correspond to the same keyword. Volume pattern leakage refers to the number of encrypted documents associated with a query token (or vice versa, the number of query tokens associated with an encrypted document). Co-occurrence pattern leakage illustrates how many encrypted documents are associated with a pair of query tokens or how many query tokens are associated with a pair of encrypted documents.

However, it is important to note that EDESE is more vulnerable to LAAs than general searchable encryption (SE) schemes due to its more stringent requirements. Its matching process on the server restricts the data owners from employing complex approaches to mitigating LAAs, and the only option for addressing LAAs is to incorporate dummy messages into the server's responses as a means of addressing such attacks.

Existing Approaches. Existing approaches to safeguard general SE from LAAs cannot protect EDESE because these approaches require additional server operations except for matching operations or multiple communications between server and client, and cannot be applied to EDESE [1, 3, 17, 21, 25, 28]. On the other hand, existing countermeasures that fulfill the requirement of EDESE only focus on query volume leakage [2, 4, 5, 12, 15]. They are not secure against other leakages, such as document volume leakage [23] and query/document co-occurrence leakage [4, 12, 26].

1.1 Our Contribution

Leakage-Resilient EDESE. To effectively strengthen EDESE against LAAs while maintaining a balance between efficiency and security, we introduce a novel security concept: **Leakage-Resilient EDESE (LR-EDESE)**, characterized by k-indistinguishability.

To avoid confusion, we clarify that "leakage-resilient" in this paper differs from its conventional cryptographic meaning, which typically refers to resistance against side-channel attacks that leak memory or computation details. Here, it signifies that an EDESE scheme remains secure against LAAs even when partial information is exposed. Additionally, our notion of "k-indistinguishability" aligns more closely with k-anonymity, ensuring that each query or document volume is indistinguishable from at least k alternatives, thereby limiting leakage exploitation. We rigorously formalize LR-EDESE within a cryptographic framework under this definition.

Volume Leakage Hiding Methods. We propose a novel volume leakage hiding method called modulo-based document partition (MDBPar), which partitions document volumes based on a modulo operation to hide document volume leakage, and a query volume leakage hiding method, inspired by Bost et al.'s work [3], named keyword clustering (KClu), which groups keywords into clusters to obscure query volume patterns. Both approaches introduce *dummy documents* alongside real keywords to effectively mask volume leakage. We formally prove that MDBPar and KClu achieve k-indistinguishability, ensuring robust mitigation of both document and query volumes in EDESE.

Volume Leakage-Resilient EDESE. We construct a concrete Volume Leakage-Resilient EDESE (VLR-EDESE). The VLR-EDESE achieves the complete query response, concurrently thwarting LAAs in terms of query volume and document volume leakages through the incorporation of k-indistinguishability. The VLR-EDESE sequentially employs the proposed volume leakage hiding methods to generate encrypted documents. Initially, VLR-EDESE applies the MDBPar method to derive a set of semi-obfuscated documents from the source documents. Then it takes this set as input for the subsequent execution of the KClu method, producing a set of fully obfuscated documents. These obfuscated documents comprise both original and dummy documents, and the server storage overhead of VLR-EDESE is linear to the count of the obfuscated documents.

Evaluations. We implement a prototype of VLR-EDESE and evaluate its performance through both theoretical analysis and experimental evaluation focusing on setup computation overhead, server storage overhead, and communication overhead. Our theoretical analysis demonstrates that VLR-EDESE achieves a setup complexity of $O(m \log m + mn + km + k^2)$, a storage overhead of $O(km + n)$, and a communication overhead of $O(km + n)$, where m indicates the number of keywords, n denotes the number of documents, and k is the indistinguishability parameter. We conducted experiments using our prototype implementation on the Enron email dataset [27], a widely used benchmark in SSE research. The results highlight the efficiency of VLR-EDESE compared to state-of-the-art approaches, including PRT-EMM [16] and FP-EMM [25]. VLR-EDESE takes around 1 hour to produce encrypted documents based on the top 5,000 keywords and 37,172 source documents. With the strongest security setting ($k = 5000$), VLR-EDESE incurs an overall overhead of only 63× compared to the baseline EDESE without volume leakage mitigation, significantly lower than the 320× and 97× overheads of adapted state-of-the-art schemes. This maximum k value represents the theoretical security limit achievable by any EDESE or ORAM-based scheme. Furthermore, we test with smaller k values (specifically 10, 20, 50, and 100), significantly reducing storage and communication overhead to no more than 2 times and 2.5 times that of the baseline scheme respectively. This demonstrates VLR-EDESE's flexibility in balancing efficiency and security.

CloudSec. We present CloudSec, which leverages the prototype implementation of VLR-EDESE to offer a user-friendly application tailored for the Windows platform. CloudSec enables seamless

integration with various cloud storage platforms that provide REST APIs, such as OneDrive, Dropbox, Google Drive, and IDrive. In this paper, we highlight CloudSec's integration with Microsoft OneDrive, demonstrating its ability to interact effectively with cloud services.

2 Related Work

In this section, we conducted a comprehensive review of existing symmetric searchable encryption (SSE) schemes against LAAs. Our examination reveals two limitations when these schemes are applied to protect EDESE against LAAs:

Limitations in Applying SSE Schemes to EDESE. Firstly, existing SSE schemes [1, 3, 5–8, 16, 17, 21, 25, 28] are often inadequate for protecting EDESE due to the nature of server operations extending beyond simple matching. SSE schemes designed to mitigate LAAs typically require additional server-side operations. For example, ORAM-protected SSE schemes [10, 14, 16, 30] introduce significant overhead through supplementary computations, state maintenance on the server, and increased client-server communication. However, EDESE primarily relies on the fundamental operations of a storage server, including storage and matching. In such cases, ORAM-protected SSE schemes fail to effectively safeguard EDESE. Designing appropriate EDESE schemes to defend LAAs is still challenging.

Limited Protection Coverage of Document Retrieval Phase. Many existing SSE schemes [3, 5, 7, 9, 16, 18, 22, 25, 29] are categorized as structured encryption (STE) schemes, where relationships between keywords and documents are represented as mappings to document indices. While STE secures the index retrieval phase—allowing users to retrieve document indices matching a searched keyword—it introduces a potential vulnerability during document retrieval, when users access documents based on their indices. STE schemes, such as encrypted multi-maps (EMMs) [6, 11, 16], enable outsourcing encrypted structural data to an untrusted server while allowing the data owner to perform operations on the encrypted data. In these schemes, the <keyword, document credential> relationships, known as search indexes, are outsourced, and the client queries encrypted keywords (trapdoors) to retrieve document indices. This process, known as index retrieval, is crucial for SSE systems.

However, the query phase in EDESE includes both index and document retrieval, requiring protection for each to prevent leakage. Thus, using STE alone is insufficient. If EDESE protects only the search index with STE but naively outsources encrypted documents, attackers can observe retrieved documents and infer index patterns. To prevent this, our EDESE schemes securely outsource encrypted documents to mitigate leakage during document retrieval.

3 Models and Definitions

In this section, we first introduce the essential pre-definitions for EDESE presentations. Then, we present the system model of EDESE along with the formal definition of its syntax. Next, we describe the adversarial models of LAAs. Finally, we provide the formal definition of leakage-resilient security.

3.1 Pre-defined Notions

To illustrate proposed approaches more clearly, there are three predefined notions.

- Given a set of documents D, $K(D)$ denotes the set of all keywords in D. Besides, if given a set of encrypted documents ED, $K(ED)$ denotes all query tokens associated with ED.
- $Vec(D, d)$ represents all keywords associated with document $d \in D$. $Vec(D, kw)$ is defined to represent all documents associated with keyword $kw \in K(D)$. Similarly, $Vec(ED, ed)$ indicates all query tokens associated with an encrypted document $ed \in ED$ and $Vec(ED, qt)$ indicates all encrypted documents associated with a query token qt. In one word, Vec represents four functions based on its inputs.
- A volume function Vol is defined to denote the number of keywords (resp. query tokens) associated with a document (resp. encrypted document) or the number of documents (resp. encrypted documents) associated with a keyword (resp. query tokens). This is referred to as query volume or document volume.

3.2 System Model of EDESE

EDESE involves two entities, a client and a server, and employs five core algorithms: key generation, setup, query token generation, matching, and final result algorithms. These algorithms are grouped into two distinct phases: setup phase and query phase, as shown in Fig. 1.

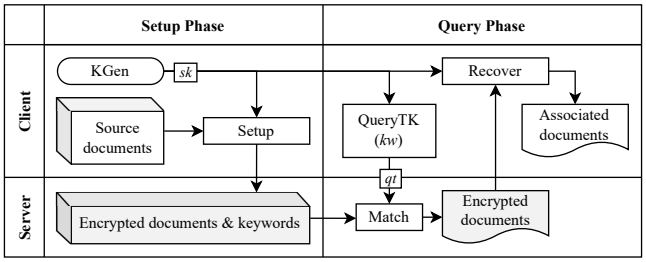


Figure 1: System model of EDESE.

In the setup phase, the client first runs the key generation algorithm, **KGen**, to produce a secret key sk, which is kept confidential. Subsequently, the client executes the setup algorithm, **Setup**, using a collection of source documents as input. This process generates a collection of encrypted documents, which are then sent to and stored by the server.

During the query phase, the client encrypts a queried keyword kw into a query token qt using the query token generation algorithm, **QueryTK**, together with the secret key sk. The query token is sent to the server, which uses the matching algorithm, **Match**, to identify all encrypted documents related to qt and returns them to the client. Finally, the client applies the recovery algorithm, **Recover**, with the secret key sk as input to retrieve all documents associated with the queried keyword from the received encrypted documents.

3.3 Algorithms of EDESE

An easily deployable and efficiently searchable encryption (EDESE) Σ_{EDESE}=(**KGen, Setup, QueryTk, Match, Recover**) is composed of five polynomial-time algorithms as below:

- **KGen**$(1^\lambda) \rightarrow sk$: This probabilistic algorithm, run by the client, takes as input a security parameter λ and outputs a secret key sk.
- **Setup**$(sk, D) \rightarrow ED$: It takes as inputs a secret key sk and a set of documents D, and outputs a set of encrypted documents ED combined with the encrypted keywords. It is a probabilistic algorithm run by the client who outsources ED to a server.
- **QueryTK**$(sk, kw) \rightarrow qt$: This deterministic algorithm takes as inputs the secret key sk and a keyword kw, and outputs a query token qt. This algorithm is run by the client.
- **Match**$(ED, qt) \rightarrow ct$: It takes as inputs the encrypted documents ED and a query token qt, and outputs a response ct, which is a subset of ED. This algorithm is run by the server.
- **Recover**$(sk, ct) \rightarrow pt$: It takes as inputs the secret key sk and the response ct, and outputs a set of documents pt. This algorithm is run by the client.

Correctness. For any kw, sk, and D, it calculates $ED \leftarrow$ **Setup**(sk, D), $qt \leftarrow$ **QueryTK**$(sk, kw), ct \leftarrow$ **Match**(ED, qt), and $pt \leftarrow$ **Recover**(sk, ct). Then, a Σ_{EDESE} is **correct** if and only if pt is identical to the search result $D(kw)$, which indicates the set of documents related to the keyword kw in D.

3.4 LAAs on EDESE

In recent years, numerous studies have demonstrated that underlying keywords of query tokens can be retrieved by exploiting the vulnerabilities of EDESE schemes, given prior knowledge about the targeted database [2, 4, 5, 12, 15, 23]. This prior knowledge includes either a partially or fully encrypted database being targeted, along with auxiliary information. Specifically, the targeted encrypted database is obtained from the server in the setup phase or observed via communications between the client and the server during the query phase.

Attack models can be categorized into two types based on the extent of knowledge contained in the auxiliary information: known-data attacks and inference attacks. In a stronger attack model [2], known as the known-data attack, the auxiliary information is partially or the entire underlying plaintext of the targeted encrypted database. In a weaker attack model, known as the inference attack, the auxiliary information is a similar database that has the same or close distribution as the targeted database.

In general, the attacker learns the leakage patterns, including access pattern, search pattern, volume pattern, co-occurrence pattern, etc., from the targeted encrypted database and the auxiliary information via leakage functions, which we will formalize the commonly used leakage functions at the end of this subsection. The attacker's objective is to establish the mapping relationships between the query tokens and the keywords.

In this paper, we consider a strong LAA model on EDESE, characterized by the complete observation of the targeted encrypted database and utilizing the entire underlying plaintext database as auxiliary information. This attack model encompasses both the setup phase and the query phase, as the target encrypted database is acquired through these stages. It is worth noting that during the query phase, certain dummy information can be filtered out.

This paper focuses solely on addressing volume leakage, leaving aside search, access, and co-occurrence patterns. This choice aligns with the nature of EDESE, as methods to mitigate these other patterns would undermine its efficiency and easy deployment. For example, ORAM-based searchable encryption can hide multiple patterns but introduces significant communication overhead, compromising EDESE's efficiency. Similarly, randomized query token generation could help with search and access pattern leakage, but requires additional mapping and mathematical computations, which counteract the ease of deployment of EDESE. Moreover, preventing volume pattern leakage alone is adequate to protect EDESE against LAAs [2, 4, 5, 12, 15, 23].

Leakage Functions. To better demonstrate leakage functions, our common math functions are defined, including: 1) $Eq(a, b) = 1$ if $a = b$; otherwise, $Eq(a, b) = 0$; 2) given a matrix $M_{m\times n}$, $Row(M, i) = (M_{i,1}, ..., M_{i,n})$; 3) $Col(M, i) = (M_{1,i}, ..., M_{m,i})$; 4) and given a vector or an array V, $|V|$ counts the number of elements in V.

- **Search pattern leakage function.** The search pattern leakage function $\mathcal{L}_S$ identifies whether two queries qt_1, qt_2 refer to the same keyword, i.e. $\mathcal{L}_S(qt_1, qt_2) = Eq(qt_1, qt_2)$.
- **Access pattern leakage function.** The access pattern leakage function $\mathcal{L}_A$ produces a matrix $M^{(m,n)}$, reflecting the relationships between a set of n encrypted documents ED and a set of m query tokens Q, such that $M_{i,j} = 1$ if $qt_i \in Vec(ED, ed_j)$; otherwise $M_{i,j} = 0$, where $qt_i \in Q, ed_j \in ED, i \in [1, m], j \in [1, n]$.
- **Volume pattern leakage function.** These functions capture the number of associations in the access pattern matrix M. The query token volume leakage function $\mathcal{L}_{QV}$ (response volume) computes the number of encrypted documents linked to each query token: $\mathcal{L}_{QV}(qt_i) = |Row(M, i)|$, where $i \in [1, m], qt_i \in Q$. Similarly, the document volume leakage function $\mathcal{L}_{DV}$ outputs the number of query tokens linked to each encrypted document: $\mathcal{L}_{DV}(ed_j) = |Col(M, j)|$, where $j \in [1, n], ed_j \in ED$.
- **Co-occurrence pattern leakage function.** These functions measure overlap between query tokens or documents. The query token co-occurrence leakage $\mathcal{L}_{QC}$ yields a matrix $M^{(QC)} \in \mathbb{Z}^{m\times m}$ where for all $j_1, j_2 \in [1, n]$, $M^{(QC)}_{i_1,i_2} = |Row(M, i_1) \cap Row(M, i_2)|$. The document co-occurrence leakage $\mathcal{L}_{DC}$ similarly produces $M^{(DC)} \in \mathbb{Z}^{n\times n}$, with $M^{(DC)}_{j_1,j_2} = |Col(M, j_1) \cap Col(M, j_2)|$.

3.5 Leakage-Resilient EDESE

We demonstrate the Leakage-Resilient EDESE (LR-EDESE) with k-indistinguishability and leakage function $\mathcal{L}$ in Definition 3.1.

Definition 3.1 (k-ind $\mathcal{L}$-resilient EDESE). Let $\Sigma =$ (**KGen**, **Setup**, **QueryTK**, **Match**, **Final**) be an EDESE scheme, $\lambda \in \mathbb{N}$ be the security parameter, k be a threshold number, $\mathcal{L}$ be a polynomial time leakage function, $\mathcal{A}$ be a probabilistic polynomial time adversary, and C be a polynomial time challenger with the response capability of the following oracle.

- $O_{\mathbf{QTGen}}(\cdots)$: $\mathcal{A}$ submits a keyword $kw_i \in K(D)$, where $i \in [1, |K(D)|]$. C returns a deterministic query token $qt_{(i,\alpha')}$, where $\mathcal{L}(qt_{(i,1)}, ED) = \cdots = \mathcal{L}(qt_{(i,m)}, ED) = \mathcal{L}(qt_i, ED)$, $\alpha' = \alpha + l$

mod m, m indicates the total number of query tokens satisfying the equivalent leakage to qt_i, $m \geqslant k$, $qt_{(i,\alpha)} = qt_i$, and $qt_i \leftarrow \mathbf{QueryTK}(sk, kw_i)$.

We consider the following experiment $\mathbf{LR}^{\Sigma}_{\mathcal{A}}(\lambda, k, \mathcal{L})$:

Experiment $\mathbf{LR}^{\Sigma}_{\mathcal{A}}(\lambda, k, \mathcal{L})$**:**

$sk \leftarrow \mathbf{KGen}(1^{\lambda}), \quad l \in_R [0, k-1]$
$ED \leftarrow \mathbf{Setup}(sk, D)$
where $O_{\mathbf{QTGen}}(\cdot)$ on input kw_i :
if $i \notin [1, |K(D)|]$, **return** $\perp$
else $i \leftarrow i+1$, **return** $qt_{(i,\alpha')}$
$l' \leftarrow \mathcal{A}^{O_{\mathbf{QTGen}}}(D, ED)$
if $l' = l$, **return** 1
else, return 0

We say that Σ is a k-ind $\mathcal{L}$-resilient EDESE if for all probabilistic polynomial time $\mathcal{A}$,

$$\mathbf{Adv}^{\mathbf{LR}}_{\mathcal{A}}(\lambda, k, \mathcal{L}) = \left| \mathbb{P}[\mathbf{LR}^{\Sigma}_{\mathcal{A}}(\lambda, k, \mathcal{L}) = 1] - \frac{1}{k} \right| \leqslant negl(\lambda).$$

For Definition 3.1, the challenger C runs **Setup** algorithm to produce obfuscated and encrypted documents ED based on the documents D generated by $\mathcal{A}$. Then, C chooses a random number l from 0 to $k-1$. In the query phase, $\mathcal{A}$ queries the oracle O_{QTGen} multiple times by all keywords in D. O_{QTGen} will output a query token qt corresponding to the given keyword kw based on the secret key sk, ED, the **QueryTK** algorithm, and the chosen number l, such that qt have the same leakage as the query token based on kw. In the guess phase, $\mathcal{A}$ guesses l' for l. If $l = l'$, $\mathcal{A}$ wins the security game, which means $\mathcal{A}$ can distinguish a query token based on a given keyword from a randomly chosen query token when both of them have the same leakage regarding the given leakage function. Otherwise, then we say the EDESE scheme is k-ind $\mathcal{L}$-Resilient secure.

4 VLR-EDESE

An EDESE scheme qualifies as a Volume Leakage-Resilient EDESE (VLR-EDESE) when it achieves k-ind $\mathcal{L}_{QV}$ resilient security and k-ind $\mathcal{L}_{DV}$ resilient security simultaneously. Thus, a VLR-EDESE must integrate both a query volume leakage hiding method and a document volume leakage hiding method.

The remainder of this section introduces a novel document volume leakage hiding method and a query volume leakage hiding method inspired by Bost et al.'s work [3]. We then present a concrete construction of VLR-EDESE along with its security analysis. Furthermore, we discuss padding strategies for optimizing VLR-EDESE's efficiency and security.

4.1 Document Volume Hiding Method

A modulo-based partition approach is defined to hide document volume pattern leakages, which achieves the intermediate computational complexity and the intermediate number of obfuscated documents (duplicated documents + dummy documents) compared to 1) no partitioning approach and 2) a flexible partition approach which partitions keywords associated with a document into flexible sized disjoint subsets.

Modulo-Based Document Partition (MBDPar) Algorithm. Given a set of source documents D_n and the set of associated keywords K_m, the algorithm produces a set of obfuscated documents $D'_{n'}$ and the corresponding modulus p^*, which is shown in Algorithm 1.

Algorithm 1: Modulo-based document partition algorithm MBDPar

Data: a set of source documents D_n and the set of corresponding keywords K_m.
Result: a set of obfuscated documents $D'_{n'}$, the optimal modulus p^*.

computes the maximum document volume value $v^* = \max_{d \in D_n} Vol(d)$;
forall $p \in [1, v^*]$ **do**
  initialize $O_p \leftarrow \{0\}_p, O'_p \leftarrow \{0\}_p$;
  forall $i \in [1, n]$ **do**
    $O_p[p] \leftarrow O_p[p] + \lfloor \frac{Vol(D,d_i)}{p} \rfloor$;
    if $\gamma > 0$, **then**, $O_p[\gamma] \leftarrow O_p[\gamma] + 1$, where $\gamma = Vol(D, d_i) \bmod p$;
  end
  if $O_p[v] > 0$, **then**, set $O'_p[v] \leftarrow max(k - O_{p^*}[v], 0)$;
   otherwise, $O'_p[v] \leftarrow 0$, where $\forall v \in [1, p]$;
  compute $n'_p \leftarrow \sum_{v=1}^{p} O_p[v] + O'_p[v]$;
end
select the p^* such that n'_{p^*} is minimum;
initialize $D' \leftarrow \emptyset$;
forall $d_i \in D_n$ **do**
  compute $u_i \leftarrow \lceil \frac{Vol(D,d_i)}{p^*} \rceil$; set $V \leftarrow Vec(D, d_i)$;
  forall $i' \in [1, u_i]$ **do**
    $d_{i,i'} \leftarrow_{duplicate} d_i$;
    if $i' < u_i$, **then**, update $Vol(D', d_{i,i'}) \leftarrow p^*$;
     otherwise,
     $Vol(D', d_{i,i'}) \leftarrow Vol(D, d_i) - (i-1) \times p^*$;
    update $Vec(D', d_{i,i'})$ with first $Vol(D', d_{i,i'})$ elements of V; $V \leftarrow V \setminus Vec(D', d_{i,i'})$;
  end
end
forall $v \in [1, p^*]$, **do**, generate $O'_p[v]$ dummy documents $dd_1, \cdots, dd_{O'_p[v]}$ and updates $Vec(D, *)$ and $Vol(D, *)$ functions for each of them with randomly selected v non-repeated keywords from K_m, then put them into D';
output p^* and $D'_{n'}$, where $n' \leftarrow n'_{p^*}$.

The algorithm first computes the optimal modulus value p^* such that the total number of result documents is minimal, which is represented by n'_p. Then, the MBDPar algorithm partitions the keyword vector associated with each source document to produce multiple disjoint subsets associated with multiple duplicated documents. There is no more than one sub-vector whose size is less than p^* while the sizes of others are p^*, following the modulo calculation. After that, the MBDPar algorithm associates keywords in each sub-vector with a duplication of the source document.

Next, the algorithm counts occurrences of values from 1 to p^* in document volume values. If the occurrence of a value v is less than

k and greater than 0, the algorithm creates new dummy documents associated with v keywords randomly selected from K_m to reach k. Finally, it outputs the obfuscated document $D'_{n'}$, which contains all duplicated documents and dummy documents, where n' represents the number of documents in D'.

Computational Complexity and Boundaries. The computational complexity of the MDBPar algorithm is $O\left(\frac{mn}{p^*} + n + p^*(k-1)\right)$.

The number of obfuscated documents produced by the MBDPar algorithm is bounded by $O\left(\frac{mn}{p^*} + n + p^*k\right)$. The total number of keyword-document pairs in the obfuscated documents is bounded by $O(mn + kp^{*2})$.

For further details, refer to A.1.

4.2 Query Volume Hiding Method

There are many query volume hiding algorithms, such as rounding padding, group padding, and bucket padding approaches. To achieve k-ind query volume resilient secure, recall the definition 3.1, we need to guarantee each qurery volume occurs at list k times, which is exactly idea of group padding.

Algorithm 2: keyword clustering algorithm KClu

Data: a set of source documents D_n, the set of corresponding keywords K_m, and a dummy document capacity p^*.
Result: a set of obfuscated documents $D'_{n'}$.

1 sort kw in K_m by the ascending order of the $Vol(D_n, kw)$ to obtain K'_m;
2 construct a weighted directed graph $G = (V, E, W)$, where $V = [0, m], E = \{(e_1, e_2) | e_1 \in [0, m-k], e_2 \in [e_1 + k, \min(e_1 + 2k - 1, m)]\}$, and the weighting function is defined as $W(e_1, e_2) = \sum_{e=e_1+1}^{e_2}(Vol(D_n, kw_{e_2}) - Vol(D_n, kw_e)))$;
3 calculate the shortest weighted path P^* from the start vertex 0 to the end vertex m by Dijkstra's Algorithm, where $P^* = \{e_0, e_1, \cdots, e_\mu\}$ and $e_0 = 0, e_\mu = m$;
4 the optimal collection of keyword groups is generated as $\mathbb{G} = \{g_i\}_{i\in[1,\mu]}$, where $g_i = \{kw_e\}_{e\in[e_{i-1}+1, e_i]}$;
5 calculate $O[e] = Vol(kw_{e_i}) - Vol(kw_e)$ for $kw_e \in g_i$;
6 initialize $D' \leftarrow D_n, T \leftarrow \emptyset$;
7 **forall** $i \in [1, \mu], e \in [e_{i-1} + 1, e_i]$ **do**
8 **if** $O[e] > |T|$, **then** generate total $(O[e] - |T|)$ dummy documents and append them into T and D';
9 select the first $O[e]$ dummy documents from T to form a set T' and update $Vec(D', kw_e) \leftarrow Vec(D, kw_e) \cup T', Vol(D', kw_e) \leftarrow Vol(kw_{e_i})$;
10 **forall** $d \in T'$, **if** $Vol(T', d) = p^*$, **then**, $T \leftarrow T \setminus \{d\}$;
11 **end**
12 output $D'_{n'}$, where $n' = \sum_{i=1}^{\mu}(e_i - e_{i-1})Vol(kw_{e_i})$.

VLR-EDESE employs a group padding algorithm, called keyword clustering algorithm (KClu), inspired by [3] and enhanced with our padding strategy. Given a keyword set K_m, a document set D_n, and a dummy document capacity p^* (i.e., the maximum number of keywords that can be associated with a dummy document), the KClu algorithm outputs a set of obfuscated documents $D'_{n'}$, as described in Algorithm 2.

KClu computes an optimal grouping of keywords to minimize the number of dummy documents needed while ensuring that all keywords within the same group share the same volume. To achieve this, it constructs a weighted directed graph based on sorted keywords where 1) nodes represent keywords plus a start node, 2) edges indicate keyword groups, and 3) the weight function calculates the number of keyword-document pairs required to be padded for a group. The algorithm then applies Dijkstra's shortest path algorithm to identify the optimal grouping P^*. Each group contains at least k and at most $2k - 1$ keywords, leading to a total of $(m - 2k)k + \frac{k(k-1)}{2}$ edges in the graph.

Once the optimal grouping is determined, KClu pads dummy documents with capacity p^* to form the obfuscated document set $D'_{n'}$, including both dummy and source documents, where n' denotes the total number of obfuscated documents.

Computational Complexity and Boundaries. The computational complexity of the KClu algorithm is $O\left(m \log m + km + k^2\right)$.

The total number of obfuscated documents generated by the KClu algorithm is $O(\frac{mn}{p^*} + n)$. Additionally, the number of keyword-document pairs is bounded by $O(mn)$.

For further details, refer to A.2.

4.3 Construction of VLR-EDESE

We apply the proposed volume-hiding methods to construct a concrete VLR-EDESE scheme. VLR-EDESE employs a symmetric encryption scheme $\Sigma_{SE} = (\text{KGen}, \text{Enc}, \text{Dec})$ for document encryption and decryption, along with a collision-resistant hash function $\mathcal{H}$ to generate query tokens. The detailed algorithms of VLR-EDESE are presented in Figure 2.

VLR-EDESE utilizes Σ_{SE} to generate the secret key sk, encrypt and decrypt documents, and process query tokens. To mitigate volume pattern leakage, the **Setup** algorithm first applies the MDBPar algorithm to transform the source document set D_n into a semi-obfuscated document set $D'_{n'}$. Next, the KClu algorithm further obfuscates $D'_{n'}$ to produce the final document set $D''_{n''}$. The **Setup** algorithm then encrypts documents in $D''_{n''}$ using Σ_{SE}.Enc and hashes the associated keywords using $\mathcal{H}$.

4.4 Security Analysis

We prove the security of VLR-EDESE using an expanding strategy. First, we prove that the scheme achieves k-ind $\mathcal{L}_{DV}$ resilience over a set of k documents in Lemma B.1. Next, we extend this result by demonstrating that a collection of such disjoint document sets remains k-ind $\mathcal{L}_{DV}$ resilient secure in Lemma B.2.

Similarly, we prove the k-ind $\mathcal{L}_{QV}$ resilience for query volume leakage. By combining these two results, we establish that the constructed VLR-EDESE satisfies k-ind volume leakage resilient secure, ensuring robust security against LAAs.

For further details, refer to B.

4.5 Padding Strategy

Padding with dummy query tokens becomes ineffective because an attacker can easily filter them out by observing that those dummy

Figure 2: Concrete construction of the VLR-EDESE

- **KGen$(1^\lambda) \rightarrow sk$**:
  KGen takes a security parameter λ. Then it runs $sk \leftarrow \Sigma_{SE}.KGen(1^\lambda)$ and outputs sk.
- **Setup$(sk, D) \rightarrow ED$** :
  (1) It initializes $K_m \leftarrow K(D), D_n \leftarrow D$ where $m = |K(D)|, n = |D|$.
  (2) Then, for each document $d \in D_n$, it extracts a set of keywords S and updates $Vec(d) \leftarrow S, Vec(kw) \leftarrow Vec(kw) \cup \{d\}, \forall kw \in S$.
  (3) Next, it computes $D'_{n'}, p^* \leftarrow \text{MBDPar}(D_n, K_m), D''_{n''} \leftarrow \text{KClu}(D'_{n'}, K_m, p^*)$.
  (4) After that, the algorithm encrypts D'' into ED as $ED \leftarrow \{ed|ed = r|\widehat{ed}|qt_1|\cdots|qt_{Vol(d)}, r \xleftarrow{R} \mathbb{R}, \widehat{ed} = \Sigma_{SE}.Enc(sk|r, d), qt_i = \mathcal{H}(sk|kw_i), kw_i \in Vec(d), i \in [1, Vol(d)], \forall d \in D''\}$.
- **QueryTk$(sk, kw) \rightarrow qt$** :
  The algorithm encrypts kw into a query token qt. It calculates $qt = \mathcal{H}(sk|kw)$ and outputs it.
- **Match$(ED, qt) \rightarrow ct$** :
  The server finds the set of encrypted documents ct associated with the given qt from ED.
- **Final$(sk, ct) \rightarrow pt$** :
  The client decrypts all encrypted documents in ct and filters out dummy ones. It initializes $pt \leftarrow \emptyset$. Afterwards, for each $ed \in ct$, it parses ed to $r|\widehat{ed}|qt_1|\cdots|qt_{Vol(d)}$ and then obtain the document as $d = \Sigma_{SE}.Dec(sk|r, \widehat{ed})$. If d is a real document, it put d in pt; otherwise, do nothing. Finally, pt will only hold all documents related to the queried keyword kw.

query tokens are never queried. In addition, padding with neither dummy documents nor dummy query tokens places an additional burden on the client to identify whether a responded document associates with the query token. Taking into account the above reasons, the proposed VLR-EDESE adopts a padding strategy that exclusively utilizes dummy documents, ensuring that only query responses are padded with dummy entries.

5 Evaluation

5.1 Overhead of EDESE

There are three critical overheads of EDESE, including setup computational overhead, server storage overhead, and communication overhead.

- **Setup computational Overhead.** In EDESEs, computational overhead refers to the computational costs associated with the setup phase. It is determined by the computational complexity of the **KGen** and **Setup** algorithms.
- **Server Storage Overhead.** In EDESEs, server storage overhead indicates the data size of encrypted documents. In this paper, we use the number of encrypted documents instead.
- **Communication Overheads.** The communication overheads include client communication (query) overhead and server communication (response) overhead. For EDESE, client communication overhead is represented by the average length of a search request. Server communication overhead is represented by the average associated encrypted documents of a search request. It can be calculated by (the number of document-query_token mappings)/(the number of query tokens).

5.2 Theoretical Evaluation

We analyze the complexities of the proposed VLR-EDESE in terms of computational, storage, and communication overheads in Section 5.1. The setup computation complexity is $O\left(m \log m + mn + km + k^2\right)$. The server storage complexity is $O(km + n)$. The client communication complexity remains constant at $O(1)$, while the server communication complexity is $O(km + n)$.

For further details, refer to Section A.3.

5.3 Experimental Evaluation

In our experiments, we utilize the Enron email dataset [27], following the methodology of previous studies [4, 23, 26, 32]. This dataset comprises 37,470 emails exchanged by 150 employees of the Enron Corporation between 2000 and 2002. To preprocess the data, we adopt the approach used in [4, 23, 32], extracting the top 5,000 keywords after removing stop words. We then implement VLR-EDESE using these 5,000 keywords and the complete set of 37,470 emails as source documents.

Evaluation with Varied Parameters. We evaluate the prototype implementation of the proposed VLR-EDESE using varying percentages of source documents and varying values of k. The percentage of source documents ranges at 0.1%, 0.5%, 1%, 5%, 10%, 50%, and 100%. Similarly, the k values are set to 10, 20, 50, and 100. Our evaluation focuses on three key aspects: computation overhead, server storage overhead, and communication overhead.

Table 1 and Figure 3 show that VLR-EDESE is particularly effective for large-scale datasets, especially when k is not too large. Specifically, as highlighted in the blue row of Table 1, for $k \in \{10, 20, 50, 100\}$ and the completed dataset (37,470 documents), VLR-EDESE achieves the following.

- Storage overhead (number of documents) is limited to at most 2 times the source dataset size.
- Communication overhead (number of indices) does not exceed 2.5 times the number of source indices.
- The computational overhead (run-time) is approximately 1 hour.
- Memory consumption does not exceed 8 GB, fitting comfortably within the typical 16 GB RAM of modern PCs.

Furthermore, as demonstrated by the trend lines in Figures 3a and 3b, **the communication and storage overheads of VLR-EDESE**

decrease as the number of source documents increases. Meanwhile, Figures 3c and 3d show that runtime and memory requirements scale almost linearly with the number of source documents.

Moreover, Figures 3e, 3f, and 3h indicate that the rates of obfuscated keyword-document pairs (resp. obfuscated documents) relative to their original counterparts, as well as memory usage, exhibit a linear relationship with k. Finally, Figure 3g confirms that the value of k does not impact the runtime of the setup phase.

The flexibility of VLR-EDESE allows for tunable security parameters that balance privacy and efficiency. By selecting a moderate k value (e.g., 100), overheads can be significantly reduced. This adaptability enables VLR-EDESE to surpass other schemes in providing adjustable security guarantees alongside optimized storage, communication, and computational efficiency.

Comparison with Existing Schemes. This section presents a comparative analysis of the VLR-EDESE against two state-of-the-art SSE schemes, PRT-EMM [16] and FP-EMM [25], along with the baseline EDESE scheme. To adapt these SSE schemes to the EDESE setting and preserve their security guarantees within the EDESE framework, we apply a straightforward transformation: replacing encrypted indices in each scheme with encrypted documents, ensuring that identical encrypted indices map to the same encrypted document. This adaptation does not account for additional computational overhead incurred on the server. Besides, the baseline EDESE scheme is the EDESE without any leakage protection.

As shown in Table 2, **VLR-EDESE outperforms PRT-EMM and FP-EMM in terms of overall efficiency**, which encompasses server storage, query communication, and server response overheads, as well as setup complexity. Compared to PRT-EMM and FP-EMM, while VLR-EDESE incurs a storage overhead of at most 138× compared to the baseline EDESE, it significantly reduces setup costs and achieves superior efficiency in query processing. Specifically, VLR-EDESE matches the baseline EDESE in client query communication overhead while achieving the lowest server response overhead, equivalent to that of PRT-EMM.

The compared schemes are implemented as follows.

- **Baseline EDESE.** The baseline EDESE encrypts each document along with its associated query tokens without incorporating dummy or duplicate documents. This scheme serves as a baseline for evaluating the efficiency of other schemes.
- **PRT-EMM.** We adopt the most storage-efficient parameter configuration from the original PRT-EMM work with $\alpha = 0.5$.
- **FP-EMM.** FP-EMM employs a full padding strategy, ensuring that all query response lengths are padded to the maximum observed query response length. Each query requires two token transmissions—one for each table—while the server responds twice, returning the maximum possible document set.

Both PRT-EMM and FP-EMM achieve k'-query volume RL security, where $k' = 5000$. To ensure a fair comparison, VLR-EDESE is configured to provide the same k'-indistinguishability in both query volume and document volume pattern leakage. This k' value represents the theoretical security limit achievable by any EDESE or ORAM-based scheme. Each encrypted document is standardized to a size of 1KB to isolate the impact of document encryption.

6 CloudSec

In this section, we introduce CloudSec,a real-world application that integrates the prototype implementation of VLR-EDESE, enabling secure keyword searches on popular cloud storage services.

6.1 System Design

CloudSec consists of three modules, including the user interface management module (UIM module), the cloud service module (CS module), and the encryption module, as illustrated in Figure 4.

- **UIM module** manages interactions between UIs, including the cloud platform management UI, file upload UI, search UI, file viewer, and menu UI. Specifically, the cloud platform management UI allows users to configure connections to cloud storage services. The file upload UI enables users to upload directories to a specified cloud storage platform. The search UI receives search keyword and selected storage platforms from users. The file viewer displays files from the search result. The menu UI provides access to all functionalities.
- **CS module** handles cloud storage interactions, including user authentication and file operations with the cooperation of the cloud service platform. File operations include uploading, keyword search, and downloading.
- **Encryption module** integrates the prototype implementation of VLR-EDESE and manages user secret keys, ensuring secure keyword search functionality.

6.2 User Workflow

Similar to EDESE, CloudSec operates in two phases: setup and query. The workflow is illustrated with OneDrive.

In setup phase, when launching CloudSec, the encryption module runs `KGen` to generate and store the user's secret key. Before uploading files, the user authorizes CloudSec to access OneDrive. The UIM module redirects the "Add Cloud → OneDrive" action to the CS module, which invokes OneDrive's authentication. After the user grants access, the CS module stores the resulting token. The user selects a folder, the encryption module encrypts the files, and the CS module uploads them using the stored token.

In query phase, the user inputs a keyword via the UIM search box and selects OneDrive. The encryption module generates a query token, which the CS module sends along with the stored token to OneDrive's search interface. Upon receiving results, the CS module downloads the files via returned URLs. The encryption module then decrypts and filters the files before displaying them in File Explorer.

6.3 Implementation

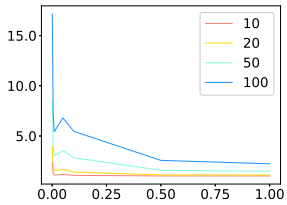
This section outlines key details of the CloudSec implementation. We developed CloudSec in C# for the Windows platform, integrating OneDrive as a cloud storage service. CloudSec interacts with OneDrive via its REST API for seamless file management. To enhance the user experience, CloudSec is designed as a tray application, enabling convenient menu access and robust hotkey functionality for streamlined operations.

Encryption module. The encryption module of CloudSec implements VLR-EDESE using AES-CBC-256 for symmetric encryption and HMAC-SHA-256 for query token generation. Specifically,

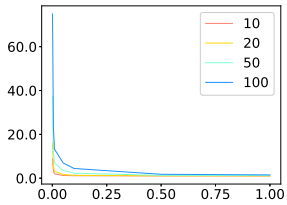
Table 1: Performance of VLR-EDESE.

Data Pct.	No. of Indices[1]					No. of Documents					Memory(GB)[2]				Runtime(s)[3]			
	Source[4]	ks[5]				Source	ks				ks				ks			
		10	20	50	100		10	20	50	100	10	20	50	100	10	20	50	100
0.001	2246	5286 (2.35)	8826 (3.93)	19438 (8.65)	38522 (17.15)	37	328 (8.86)	594 (16.05)	1384 (37.41)	2772 (74.92)	0.01	0.02	0.05	0.09	24.5	23.74	22.36	20.15
0.005	11995	16350 (1.36)	24207 (2.02)	47817 (3.99)	87173 (7.27)	187	613 (3.28)	1020 (5.45)	2251 (12.04)	4321 (23.11)	0.09	0.09	0.18	0.35	336.42	338.12	332.03	317.61
0.01	19761	21899 (1.11)	31133 (1.58)	59789 (3.03)	107504 (5.44)	374	745 (1.99)	1193 (3.19)	2600 (6.95)	4961 (13.26)	0.35	0.35	0.35	0.47	564.96	561.03	545.09	529.66
0.05	112242	130253 (1.16)	185181 (1.65)	398322 (3.55)	762503 (6.79)	1873	2369 (1.26)	3071 (1.64)	6603 (3.53)	12991 (6.94)	0.47	0.47	0.86	1.72	1454.34	1514.67	1450.9	1421.78
0.1	221540	239690 (1.08)	313172 (1.41)	627262 (2.83)	1209040 (5.46)	3747	4224 (1.13)	4955 (1.32)	8513 (2.27)	16638 (4.44)	1.72	1.72	1.72	2.17	1676.92	1670.76	1655.18	1613.32
0.5	1102810	1127398 (1.02)	1235006 (1.12)	1744237 (1.58)	2831602 (2.57)	18735	19498 (1.04)	20409 (1.09)	24442 (1.3)	33467 (1.79)	2.84	3.06	3.49	4.43	2927.11	2906.1	2828.16	2858.69
1	2193545	2245102 (1.02)	2437223 (1.11)	3253320 (1.48)	4901463 (2.23)	37470	38411 (1.03)	39776 (1.06)	45263 (1.21)	56648 (1.51)	5.81	5.96	6.53	7.79	3743.62	3752.6	3696.84	3661.42

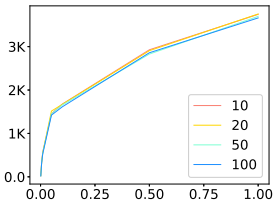
[1] "Indices" indicates keyword-document pairs.
[2] "Memory(GB)" represents the peak of memory consumed in gigabytes.
[3] "Runtime(s)" stands for the runtime to setup VLR-EDESE in seconds.
[4] "Source" indicates the numbers of original indices and documents without any hiding approaches.
[5] "ks" denotes various k values, including 10, 20, 50, 100.



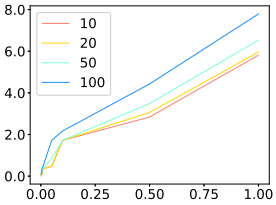
(a) Rates of keyword-document pairs to ones in the baseline EDESE.



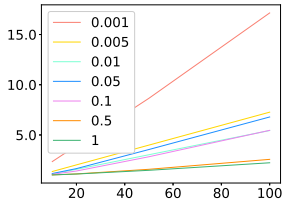
(b) Rates of the obfuscated documents to the source documents.



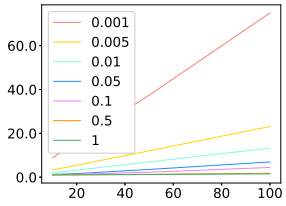
(c) Run times.



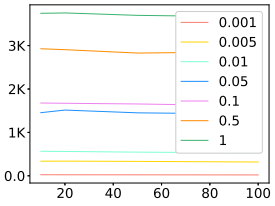
(d) Memories.



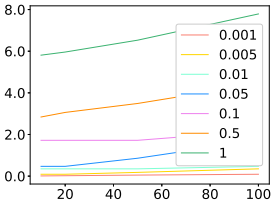
(e) Rates of keyword-document pairs to ones in the Baseline EDESE.



(f) Rates of the obfuscated documents to the source documents.



(g) Run times.



(h) Memories.

Figure 3: Experiment performances in the number of keyword-document pairs, the number of obfuscated documents, setup running times, and memory peaks setup required. (a, b, c, d) indicate performance over percentage numbers of source documents. (e, f, g, h) demonstrate performances over k values.

Table 2: Comparisions of state-of-art SSE schemes and the VLR-EDESE

Scheme	Setup Complexity	Storage (Server)	Query Communication		Overall*
			Client (query)	Server (response)	
Baseline	$O(mn)$	36.6 MB (1x)	32 B (1x)	438 KB (1x)	1
PRT-EMM	$O(vn^2)$	30.7 GB (860x)	1.6 MB (50Kx)	22 MB (50x)	320
FP-EMM	$O((1+\alpha)mn^2)$	3.3 GB (94x)	3 KB (99x)	42 MB (99x)	97
VLR-EDESE	$O\left(m\log m + mn + km + k^2\right)$	5.1 GB (138x)	32 B (1x)	22 MB (50x)	63

overall rate = (server storage rate + client query rate + server response rate) / 3.

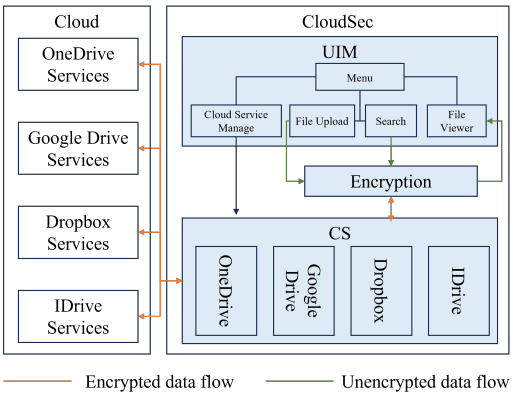


Figure 4: The Architecture of CloudSec.

AES-CBC-256 ensures secure encryption and decryption of documents, while HMAC-SHA-256functions as a keyed hash mechanism for generating query tokens. In addition, a cryptographically secure random number generator (RNG) is employed for secret key generation. All cryptographic operations utilize the System.Security.Cryptography namespace in C#.

The encryption module encrypts both file names and content to ensure data confidentiality. Consequently, all files stored on cloud platforms retain encrypted filenames and contents. To optimize efficiency, the module decrypts only file names to filter out dummy and duplicate entries, eliminating unnecessary file content decryption.

UIM. The UIM module manages the menu UI, cloud service management UI, file upload UI, and search UI, ensuring seamless user interaction. Specifically, the menu UI, accessible via the tray application's right-click menu, provides options for cloud service management, file upload, search, and exit. The cloud service management UI allows users to add or remove connections to cloud storage platforms. The file upload UI includes cloud storage platform selection and a directory selection dialog for specifying files to upload. The search UI presents a search box containing a text input field, checkboxes for selecting target cloud platforms, and a submission button for executing searches.

CS. The CS module facilitates cloud storage integration by utilizing REST APIs, which employ standard HTTP methods for resource operations. Each API request is stateless, meaning it contains all necessary information for execution, as the server does not retain client context between requests. REST APIs are widely used in cloud service development, e.g., OneDrive, Dropbox, IDrive, and Google Drive.

For OneDrive integration, CloudSec employs OneDrive REST APIs with JSON as the data exchange format, which is efficiently handled in C# using Json.NET. The CS module requests OneDrive user authentication through the HTTP URL:
login.microsoftonline.com/common/oauth2/v2.0/authorize.
For small file (<4 MB) uploading, the CS module sends request to [PUT /me/drive/items/parent-id:/filename:/content] while [POST /me/drive/items/itemId/createUploadSession] for large file uploading. The searching REST API is [GET /me/drive/root/search(q='query-token')]. By leveraging these APIs, the CS module enables secure authentication, efficient file management, and keyword search functionality within CloudSec.

6.4 Achievements

In this section, we outline the key achievements of **CloudSec** in terms of security, usability, and flexibility.

- **Security.** As an application of VLR-EDESE, CloudSec preserves the same level of security, specifically volume leakage-resilient EDESE. In alignment with the EDESE design, CloudSec only outsources encrypted files and encrypted file names to cloud storage platforms. All encryption and decryption processes are confined to the local environment, ensuring data confidentiality.
- **Usability.** CloudSec offers a seamless user experience of secure keyword searches on cloud storage platforms. Firstly, the application presents a search textbox, which is similar to existing file search tools, such as Everything and Listary, and minimizes the user learning curve. Then, files of the search result are presented in a File Explorer window as if searching on plaintext files, providing a seamless and intuitive experience for users.
- **Flexibility.** CloudSec's design prioritizes adaptability and ease of integration with existing systems. On one hand, the CS module is capable of connecting with popular cloud storage platforms through REST APIs, enabling it to work seamlessly with current cloud services without complex modifications. On the other hand, following the VLR-EDESE model, CloudSec requires only basic operations—uploading, downloading, and searching—on cloud storage platforms. No additional platform-specific configurations are needed, simplifying implementation and ensuring compatibility with all existing cloud storage providers.

7 Conclusion

In this paper, we introduced a novel leakage resilience security concept for EDESE, designed to prevent leakage abuse attacks while preserving the key advantages of EDESE, considering the trade-off between storage and communication overheads. Then, we proposed a novel document volume hiding approach called modulo-based document partitioning and integrated it with a group padding approach for query document volume hiding. Using these methods, we constructed VLR-EDESE, a volume-leakage-resilient EDESE. We have implemented the prototype of VLR-EDESE and conducted experiments to evaluate its performance. Our experiments and evaluations demonstrated the suitability of VLR-EDESE for large-scale EDESE by effectively balancing privacy with computation, storage, and communication overheads, achieving flexible k indistinguishability. Furthermore, we integrated the prototype into CloudSec for real-world use, enabling secure keyword searches on popular cloud storage services, such as OneDrive, Google Drive, Dropbox, and IDrive, via REST APIs.

Acknowledgments

We thank anonymous reviewers for helpful comments. Jianting Ning was supported in part by the National Natural Science Foundation of China (Grant No. 12441101, 62372108, 62425205). Yingjiu Li was supported in part by the Ripple University Blockchain Research Initiative.

References

[1] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. 2018. PIR with compressed queries and amortized query processing. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 962–979.

[2] Laura Blackstone, Seny Kamara, and Tarik Moataz. 2019. Revisiting leakage abuse attacks. *Cryptology ePrint Archive* (2019).

[3] Raphael Bost and Pierre-Alain Fouque. 2017. Thwarting Leakage Abuse Attacks against Searchable Encryption–A Formal Approach and Applications to Database Padding. *Cryptology ePrint Archive* (2017).

[4] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 668–679.

[5] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. 2013. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference. Proceedings, Part I*. Springer, 353–373.

[6] Melissa Chase and Seny Kamara. 2010. Structured encryption and controlled disclosure. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer, 577–594.

[7] Guoxing Chen, Ten-Hwang Lai, Michael K Reiter, and Yinqian Zhang. 2018. Differentially private access patterns for searchable symmetric encryption. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 810–818.

[8] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*. 79–88.

[9] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel Rosu, and Michael Steiner. 2015. Rich queries on encrypted data: Beyond exact matches. In *Computer Security–ESORICS 2015: 20th European Symposium on Research in Computer Security, Proceedings, Part II 20*. Springer, 123–145.

[10] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. 2016. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In *Annual International Cryptology Conference*. Springer, 563–592.

[11] Marilyn George, Seny Kamara, and Tarik Moataz. 2021. Structured Encryption and Dynamic Leakage Suppression. In *Advances in Cryptology – EUROCRYPT 2021*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer International Publishing, Cham, 370–396.

[12] Zichen Gui, Kenneth G Paterson, and Sikhar Patranabis. 2023. Rethinking searchable symmetric encryption. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1401–1418.

[13] Warren He, Devdatta Akhawe, Sumeet Jain, Elaine Shi, and Dawn Song. 2014. Shadowcrypt: Encrypted web applications for everyone. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 1028–1039.

[14] Thang Hoang, Attila A Yavuz, Betul F Durak, and Jorge Guajardo. 2017. Oblivious dynamic searchable encryption via distributed PIR and ORAM. *Cryptology ePrint Archive* (2017).

[15] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access pattern disclosure on searchable encryption: ramification, attack and mitigation.. In *Ndss*, Vol. 20. Citeseer, 12.

[16] Seny Kamara and Tarik Moataz. 2019. Computationally volume-hiding structured encryption. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part II 38*. Springer, 183–213.

[17] Seny Kamara and Charalampos Papamanthou. 2013. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*. Springer, 258–274.

[18] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. 2012. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 965–976.

[19] Steven Lambregts, Huanhuan Chen, Jianting Ning, and Kaitai Liang. 2022. Val: Volume and access pattern leakage-abuse attack with leaked documents. In *European Symposium on Research in Computer Security*. Springer, 653–676.

[20] Billy Lau, Simon Chung, Chengyu Song, Yeongjin Jang, Wenke Lee, and Alexandra Boldyreva. 2014. Mimesis Aegis: A Mimicry Privacy Shield-A System's Approach to Data Privacy on Public Cloud. In *23rd usenix security symposium (USENIX Security 14)*. 33–48.

[21] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu-an Tan. 2014. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences* 265 (2014), 176–188.

[22] Muhammad Naveed, Manoj Prabhakaran, and Carl A Gunter. 2014. Dynamic searchable encryption via blind storage. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 639–654.

[23] Jianting Ning, Xinyi Huang, Geong Sen Poh, Jiaming Yuan, Yingjiu Li, Jian Weng, and Robert H Deng. 2021. LEAP: leakage-abuse attack on efficiently deployable, efficiently searchable encryption with partially known dataset. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2307–2320.

[24] Simon Oya and Florian Kerschbaum. 2022. IHOP: Improved Statistical Query Recovery against Searchable Symmetric Encryption through Quadratic Optimization. In *31st USENIX Security Symposium (USENIX Security 22)*. 2407–2424.

[25] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. 2019. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 79–93.

[26] David Pouliot and Charles V Wright. 2016. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 1341–1352.

[27] Jitesh Shetty and Jafar Adibi. 2004. The Enron Email Dataset Database Schema and Brief Statistical Report. https://api.semanticscholar.org/CorpusID:59919272

[28] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. 2000. Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE symposium on security and privacy. S&P 2000*. IEEE, 44–55.

[29] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. 2013. Practical dynamic searchable encryption with small leakage. *Cryptology ePrint Archive* (2013).

[30] Zhiqiang Wu and Rui Li. 2023. OBI: a multi-path oblivious RAM for forward-and-backward-secure searchable encryption.. In *NDSS*.

[31] Jiaming Yuan, Yingjiu Li, Jianting Ning, and Robert H Deng. 2022. M-EDESE: Multi-Domain, Easily Deployable, and Efficiently Searchable Encryption. In *International Conference on Information Security Practice and Experience*. Springer, 606–623.

[32] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2016. All your queries are belong to us: the power of File-Injection attacks on searchable encryption. In *25th USENIX Security Symposium (USENIX Security 16)*. 707–720.

A Complexities and Boundaries

A.1 MDBPar Algorithm

Computational Complexity. The computational complexity of the MDBPar algorithm includes two procedures, optimizing the modulus and padding.

The optimization process consists of two nested iterations, in a computational complexity of $O(v^*n)$, which is bounded by $O(mn)$. The padding process firstly partitions document volumes and update Vec, Vol functions for them. The complexity of this process is given by $O\left(\sum_{i=1}^{n}\left\lceil\frac{Vol(d_i)}{p^*}\right\rceil\right)$. Since $\left\lceil\frac{Vol(d_i)}{p^*}\right\rceil \leqslant \frac{v^*}{p^*}+1$, we can further bound the complexity as $O\left(\sum_{i=1}^{n}\left\lceil\frac{Vol(d_i)}{p^*}\right\rceil\right) \leqslant O\left(\frac{mn}{p^*}+n\right)$. Additionally, the padding process includes the addition of dummy documents and Vec, Vol function updating with a computational complexity of $O\left(\sum_{v=1}^{p^*} O'_{p^*}[v]\right)$. Recall that the calculation is given by $O'_{p^*}[v] \leftarrow max(k-O_{p^*}[v], 0)$, which ensures that $0 \leqslant O'_{p^*}[v] \leqslant k-1$. As a result, the computational complexity of adding dummy documents is $O\left(\sum_{v=1}^{p^*} O'_{p^*}[v]\right) \leqslant O(p^*(k-1))$.

Combining these computational complexities, the computational complexity of MDBPar is $O\left(\frac{mn}{p^*}+n+p^*(k-1)\right)$.

Boundaries. The number of obfuscated documents produced by the MBDPar algorithm is proportional to the number of computations required for updating the Vec and Vol functions, the complexity of which was previously calculated as $\sum_{i=1}^{n}\left\lceil\frac{Vol(d_i)}{p^*}\right\rceil+\sum_{v=1}^{p^*} O'_{p^*}[v]$. This is bounded by $O\left(\frac{mn}{p^*}+n+p^*k\right)$. Additionally, The total number of keyword-document pairs in the obfuscated documents is given by $\sum_{i=1}^{n} Vol(d_i)+\sum_{v=1}^{p^*} vO'_{p^*}[v]$, which is bounded by $O(mn+kp^{*2})$.

A.2 KClu Algorithm

Computational Complexity. The computational complexity of the KClu algorithm includes two procedures, choosing optimal groups forming and padding. The KClu algorithm utilizes Dijkstra's algorithm to find the optimal groups, whose complexity is $O\left((m+1)\log(m+1)+(m-2k)k+\frac{k(k-1)}{2}\right) \leqslant O(m\log m+km+k^2)$ and is not effected by the parameter p^*. Then, the algorithm adds dummy documents and updates Vec, Vol functions for each keywords. The number of updates is m.

In a conclusion, the computational complexity of the KClu algorithm is calculated as $O\left(m\log m+km+k^2\right)$.

Boundaries. The total number of obfuscated documents generated by the KClu algorithm can be determined by dividing the total number of keyword-document pairs in obfuscated documents by the capacity of a dummy document p^*, expressed as $O(\frac{mn}{p^*}+n)$. Besides, the number of keyword-document pairs is bounded by $O(mn)$.

A.3 VLR-EDESE

Setup Computation Complexity. The setup computation complexity equals the sum of the complexities of the **KGen** and the **Setup** algorithms.

Since the **KGen** algorithm runs the Σ_{SE}.KGen algorithm, its computation complexity is identical to the complexity of Σ_{SE}.KGen, denoted as $O(1)$. Recall the complexity of the MDBPar algorithm is and the complexity of the KClu algorithm in section 4.1 and 4.2, the complexity of the setup computation is calculated as:

$$O\left(\frac{mn}{p^*}+n+p^*(k-1)\right)+O\left(m\log m+km+k^2\right),$$

Considering p^* is an optimal value such that the number of documents in MDBPar's output, the first part is bounded by $O(km+n)$ where $p^*=m$. Thus, the setup complexity is:

$$O\left(m\log m+mn+km+k^2\right).$$

Server Storage Complexity. The server storage complexity equals the number of obfuscated documents generated by the KClu algorithm based on the output of the MBDPar algorithm. Therefore, the number of documents from MBDPar is substituted in the calculation of server storage complexity, shown as:

$$O\left(\frac{m\cdot O\left(\frac{mn}{p^*}+n+p^*k\right)}{p^*}+O\left(\frac{mn}{p^*}+n+p^*k\right)\right)\leqslant O(km+n).$$

Communication Complexity. The client communication complexity is $O(1)$, while the server communication complexity is calculated as:

$$O\left(m\cdot O\left(\frac{mn}{p^*}+n+p^*k\right)\cdot m^{-1}\right)\leqslant O(km+n).$$

B Security Analysis

Lemma B.1. *Given a set $S^{(ed)}$ of k encrypted documents, it is k-ind $\mathcal{L}_{DV}$ resilient secure if and only if $Vol(ed_1)=Vol(ed_2)=\cdots=Vol(ed_k)$ where $ed_1, ed_2, \cdots, ed_k \in S^{(ed)}$.*

Proof. Since the document volume leakage function $\mathcal{L}_{DV}(d)$ calculates $Vol(ed)$, we have $v^*=\mathcal{L}_{DV}(ed_1)=\mathcal{L}_{DV}(ed_2)=\cdots=\mathcal{L}_{DV}(ed_k)$. Consequently, the adversary $\mathcal{A}$ cannot leverage the leakage function $\mathcal{L}_{DV}$ to extract any information beyond v^*. Given an CPA secure symmetric encryption scheme Σ_{SE}, such as AES-CBC $\mathcal{A}$ cannot distinguish the order of encrypted documents in $S^{(ed)}$. Thus, $S^{(ed)}$ maintains k-ind $\mathcal{L}_{DV}$ resilient security.

Lemma B.2. *Given a collect $\mathbb{C}^{(ed)}$ of encrypted document sets, it is k-ind $\mathcal{L}_{DV}$ resilient secure if and only if $S_i^{(ed)}$ is at least k-ind $\mathcal{L}_{DV}$ resilient secure and $S_i^{(ed)}\cap S_j^{(ed)}=\emptyset$ where $\forall i,j\in[1,u]$ with $i\neq j$ and u indicates the number of encrypted document sets in $\mathbb{C}^{(ed)}$.*

Proof. According to Definition 3.1, since each set of encrypted documents are k-ind $\mathcal{L}_{DV}$ resilient secure, adversary $\mathcal{A}$ can not identify α produced for $\forall S^{(ed)}\in\mathbb{C}^{(d)}$ with known α'. Therefore, $\mathcal{A}$ has no ability of inferring l for all $S^{(ed)}\in\mathbb{C}^{(d)}$. In such case, the probability to win the security game is $\mathbb{P}[\mathbf{LR}^{\Sigma}_{\mathcal{A}}(\lambda,k,\mathcal{L}_{DV})=1]=\mathbf{Adv}^{\mathbf{CPA}}_{\mathcal{A},\Sigma_{SE}}(\lambda)+\frac{1}{k}$. Then, $\mathbf{Adv}^{\mathbf{LR}}_{\mathcal{A}}(\lambda,k,\mathcal{L}_{DV})\leqslant negl(\lambda)$.

Corollary B.2.1. *If a set of encrypted documents can be divided into a number of disjoint subsets and each subset is k-ind $\mathcal{L}_{DV}$ resilient secure, then this set of document is k-ind $\mathcal{L}_{DV}$ resilient secure.*

Proof. Suppose a set $\hat{S}^{(ed)}$ of encrypted documents can be divided into a collect $\mathbb{C}^{(ed)}$ of disjoint subsets, such that $\hat{S}^{(ed)}=\mathbb{U}_{i\in[1,u]}S_i^{(ed)}$ and $S_i^{(ed)}\cap S_j^{(ed)}=\emptyset$ where $\forall i,j\in[1,u]$with $i\neq j$. According to Lemma B.2, if $\forall S^{(ed)}\in\mathbb{C}^{(ed)}$ is k-ind $\mathcal{L}_{DV}$ resilient secure, $\mathbb{C}^{(ed)}$ is k-ind $\mathcal{L}_{DV}$ resilient secure. Consequently, $\hat{S}^{(ed)}$ is also k-ind $\mathcal{L}_{DV}$ resilient secure.

Corollary B.2.2. *If a set of query tokens can be divided in to a number of disjoint subsets and each subset is k-ind $\mathcal{L}_{QV}$ resilient secure, then this set of query tokens is k-ind $\mathcal{L}_{QV}$ resilient secure.*

The Corollary B.2.2 can be proved following the same proof processing. Note that k-ind $\mathcal{L}_{QV}$ resilient security relies on cryptographic hash function, which is modeled as a random oracle. This means that the hash function is assumed to provide unpredictable and consistent outputs.

Theorem B.3. *The constructed VLR-EDESE scheme is k-ind volume leakage resilient secure.*

Proof. Recall the construction of VLR-EDESE in Figure 2, MDBPar algorithm outputs the semi-obfuscated documents $D'_{n'}$, such that each document volume value occurs at least k times (described in Algorithm 1). Therefore, $D'_{n'}$ is k-ind $\mathcal{L}_{DV}$ resilient secure.

After that, KClu algorithm produces the obfuscated documents $D''_{n''}$ based on $D'_{n'}$. KClu algorithm only adds new dummy documents associated with existing query tokens, therefore, no new document volume values emerges (limited by the modulus p as Algorithm 2). Consequently, $D''_{n''}$ is also k-ind $\mathcal{L}_{DV}$ resilient secure.

In addition, KClu algorithm produces $D''_{n''}$ such that each keyword volume value occurs at least k times. Therefore, $D''_{n''}$ is k-ind $\mathcal{L}_{QV}$ resilient secure.

Since $D''_{n''}$ is k-ind $\mathcal{L}_{DV}$ resilient secure and k-ind $\mathcal{L}_{QV}$ resilient secure, the encryption of $D''_{n''}$ is k-ind volume leakage resilient secure. Hence, the proposed VLR-EDESE is a volume leakage-resilient EDESE with k indistinguishability.