Query Recovery Attack Based on Multi-Source Leakage and Semantic Embedding in Searchable Symmetric Encryption

Xiaogang Yuan, Xinle Yang, Dezhi An
School of Cyber Security, Gansu University of Political Science and Law, Lanzhou 730070, China

***Abstract*—With the rapid development of cloud computing and big data technologies, searchable encryption has become a research hotspot. To improve search efficiency, searchable encryption algorithms may leak redundant information, allowing adversaries to launch query recovery attacks by exploiting pattern leakage in searchable symmetric encryption and infer the underlying keywords of user queries. Existing query recovery attack methods only perform query recovery under multi-source leakage patterns, without considering the semantic level; they simulate user queries with plaintext rather than ciphertext and rarely take frequency as auxiliary leakage for attacks. Meanwhile, the weights between volume and frequency mostly rely on manual configuration, without dynamic allocation of the weights for volume and frequency. Therefore, this study proposes a novel attack method named refine_atk. First, a multi-layer perceptron is used to learn the weights of volume and frequency to accurately identify and recover distinctive queries. Next, co-occurrence information is employed to correct the queries recovered in the previous step. Finally, a cost matrix is constructed using the weighted co-occurrence matrix and the semantic embedding matrix obtained by the pre-trained language model MiniLM-L6, and the remaining queries are recovered in one pass via greedy graph matching. The proposed attack achieves an attack accuracy of 95% on the Enron and Lucene datasets. The attack performance remains robust even after removing part of the similar data. The attack execution efficiency of the proposed method is significantly superior to that of the traditional schemes, yielding better performance when attacking concrete searchable symmetric encryption schemes or evaluating their security. This work provides a reference for the security evaluation and defense mechanism design of searchable symmetric encryption.**

***Keywords*—Searchable symmetric encryption; query recovery attack; pattern leakage; multi-layer perceptron (MLP)**

I. Introduction

Searchable symmetric encryption (SSE) query recovery attacks mainly utilize the queries observed by the server and the returned results to infer the keywords of the user query. It is divided into active attack and passive attack. Active attack [1], is also known as injection attack. Suppose an attacker can upload a small number of malicious files, which will leak the content of encrypted queries, allowing the attacker to obtain the client query privacy. The file injection attack proposed by Zhang Y et al. [1] only requires the injection of a small amount of files. Even in the face of the SE scheme with less leakage, the query content of the client can be completely inferred. The attack proposed by Poddar et al. [2] can be recovered with just one query issued by the user. The attack exploits file injection and automatic query replay, combined with volume leakage, to recover user queries. Zhang X et al. [3] proposed two binary-based volume injection attacks, BVA and BVMA, which reduce the number of injected files by using known keywords and infer the query content by observing the query volume. Zhang L et al. [4] proposed a volume injection attack with the optimal injection size. By using the response length and size patterns, only O(logn) injection files are required, and the total size is O(nlogn).

Passive attacks can be classified into known data attacks and similar data attacks. A known data attack refers to an attacker who has full or partial knowledge of the documents stored on the server. This study focuses on the SSE solution for single-keyword query. Cash et al. [5] proposed a counting attack, which requires the number of identical files matched by any two keywords in a known document to recover the query based on the response volume (length of the access pattern). The IKK attack [6] uses access pattern leakage to recover user queries. It assumes that the attacker has some known queries and knows the probability that any two keywords appear in the same known document. The core principle is the simulated annealing algorithm. Blackstone et al. [7] proposed a subgraph attack that exploits the document volume pattern (the number of bytes each document occupies) to recover queries. The attack proposed by Ning et al. [8] simultaneously recovers user queries and databases.

Similar data attack refers to the situation where the attacker has documents with keyword distributions similar to the user real data, and the documents in the similar data do not need to be indexed. For example, public threat intelligence datasets [9] and knowledge graphs [10] can serve as sources of similar data. The attack proposed by Liu et al. [11] only relies on the leakage of search patterns. By comparing the search frequency of each keyword on known documents with the search frequency of user queries on real documents, the underlying keywords are recovered. Oya et al. [12] pointed out that by simultaneously utilizing the access pattern and the search pattern and using the optimal solver [13], the problem of linear allocation of keywords can be solved.Damie et al.[14] proposed a similar data attack that utilizes a small number of known queries as prior knowledge. Xu et al. [16] proposed an attack in a dynamic scenario, which utilizes the update volume and query response volume of the same keyword before and after the update, and combines query equality to recover the query. Gui et al. [17] redefined leakage from the perspective of SSE systems and pointed out that the vast majority of literature only considers the leakage of encrypted search indexes throughout the SSE system and does not take into account the leakage of encrypted documents. Attacks remain effective

against some SSE schemes that suppress leaks. Wang et al. [18] proposed a machine learning enhancement framework for real-time risk assessment in searchable encryption. In other fields of cybersecurity, machine learning is also widely applied to malware detection [19], [20] and threat detection [21]. In [12], [24], the authors indicate that similar data attacks can bypass the defense of pattern randomization [35], [15]. Additionally, active learning techniques [36] can be utilized to enhance entity recognition, which may be related to keyword extraction in such attacks.

Damie et al. [14] indicated that accuracy is related to query volume (the number of documents matched by the query). Oya et al. [12] pointed out that high-frequency queries are more likely to be accurately recovered. In [7], the authors emphasize that high-volume queries are easier to recover. Nie et al. [22] pointed out that high-volume and high-frequency queries are easier to recover than other queries. Based on these queries, almost all remaining queries can be recovered. The volume and frequency of keywords in the database follow Zipf's law [23], that is, the lower the ranking of a keyword, the smaller its volume and frequency. The similar data attack RSA proposed by Damie et al. [14] can recover all queries by using only 10 known queries. Passive query recovery attacks are classified into two types based on the prior knowledge utilized by the attacks: known data attacks and similar data attacks. Table I lists several known data attacks and similar data attacks.

Inspired by Nie et al.[22], prioritizing the recovery of high-volume and high-frequency queries brings significant advantages to subsequent query recovery. Fig. 1 shows the overview of the refine_atk (Ours) operation pipeline.The main contributions of this study are as follows:

- Most existing query recovery attacks simulate user queries using plaintext queries, whereas this study employs ciphertext queries for user query simulation. The statistical features used throughout the attack are modeled based on ciphertext queries. The volume, frequency, and co-occurrence information of ciphertext queries remain consistent with those in the plaintext state. Since the indices of the underlying semantic embeddings of queries are fixed, semantic vectors can still be extracted via these indices, enabling effective ciphertext query recovery and facilitating better analysis and validation of attack performance.
- This study uses a multi-layer perceptron (MLP) to dynamically allocate weights for volume and frequency. The weights learned by the MLP cannot be directly applied to distinctive query recovery; instead, a coefficient α for distinctive query recovery is derived with the help of a small number of known queries. The weights assigned to volume and frequency, as well as coefficient α vary in each simulated attack.
- This study constructs a TF-IDF weighted co-occurrence matrix from unrecovered queries and unpaired keywords. The pre-trained language model MiniLM-L6 is used to encode unrecovered queries and unpaired keywords to obtain a semantic embedding submatrix. These two components form a cost matrix, and the remaining queries are recovered in one pass via greedy graph matching.
- The proposed method (Ours) remains robust even when the attacker possesses limited auxiliary knowledge. The first module of the proposed attack achieves promising Top-10 performance, meaning that the top 10 candidates in the attack results all correspond to the recovery of distinctive queries, which is critical to the overall effectiveness of our method.

II. SSE Scheme and Attack Model

A. SSE

Searchable encryption schemes are divided into five steps. Generate key: Enter a security parameter and output the system key. The data owner encrypts the file with a key and uploads it to the cloud server. Index establishment: Extract keywords from the database, encrypt as user document indexes, and upload to the cloud server. Generate trapdoor: The data owner uses the key pair to generate trapdoors for the keywords and send to the cloud server. Search: The cloud server matches the search trapdoor with the encrypted index stored locally. If the match is successful, the cloud server will send the ciphertext document mapped by the encrypted index to the user. File decryption: Legitimate users decrypt ciphertext documents.

Bost et al. [25] proposed a forward-privacy SSE scheme named Sophos, which only relies on trapdoor permutation and does not require a structure similar to ORAM. Subsequently, Bost [26] proposed a framework for constructing forward privacy using range-constrained pseudo-random functions. Cash and Tessaro[27] discussed the balance issue between "server storage size" and "memory access locality" in SSE. The size of the encrypted index is O(NlogN), and only O(logN) reads are required during search. Chamani et al. [28] proposed three DSSE schemes, addressing the balance between "security and performance" in backward privacy and providing efficient solutions for strong privacy requirements. Kamara and Moataz [29] proposed a non-interactive and efficient SSE scheme that can handle arbitrary disjunctive queries and boolean queries. It features sublinear search efficiency in the worst-case scenario and optimal communication complexity. Patel et al. [30] proposed a scheme that supports dynamic sharing and unsharing of documents among users, with small cross-user leakage and slow growth as queries increase. Sun et al. [31] proposed the cryptographic primitive of symmetric deletable encryption, which features fast search speed and low server storage cost. Xu et al. [32] introduced the "key-updatable pseudo-random function" to construct the ROSE scheme, which simultaneously satisfies robustness, forward privacy, backward privacy and practical performance.

B. Pattern Leakage

Searchable encryption can leak the document identifier (access pattern) returned by the query. By observing and analyzing the access patterns, attackers attempt to infer the private information queried by users. The search pattern refers to which two queries in a query sequence are equal. SSE using deterministic encryption algorithms will expose the same access pattern when repeatedly searching for the same keyword. Taking advantage of this, attackers can infer whether two queries correspond to the same keyword. The volume pattern refers to the number of documents returned by each query.

TABLE I. COMPARISON OF THE EXISTING PASSIVE ATTACKS[1]

Attack	Leak	KPK		SPK		ACC
		Doc	Query	Doc	Fre	
IKK[6]	AP	■	▲	□	□	∼80%
Count[5]	AP,VP	■	□	□	□	∼90%
SubgraphID[7]	AP	▲	□	□	□	∼90%
LEAP[8]	AP	▲	□	□	□	∼100%
RSA[14]	AP	□	▲	■	□	∼85%
IHOP[24]	AP,SP	□	□	■	■	∼90%
Jigsaw[22]	AP,SP,VP	□	□	■	▲	∼90%
Ours(refine_atk)	AP,SP,VP	□	▲	■	▲	**∼ 95%**

[1] "AP" stands for access pattern, "SP" for search pattern, "VP" for volume pattern, "KPK" for known prior knowledge, "SPK" for similar prior knowledge. "Doc" stands for document, "Query" for known query, and "Fre" for frequency. "■" indicates that the attacker requires all known data or highly depends on similar data. "▲" indicates that the attacker requires some known data or relatively depends on similar data. "□" indicates that the attacker does not need known data or similar data. The first four are known data attacks, and the last four are similar data attacks. Although our attack requires some known queries to assist in attacks, in actual scenarios, it is "treated as" not knowing any known queries.

Searchable encryption can reveal the number of documents returned by the server.

C. Attacker

Honest, yet curious servers follow the SSE protocol, but attempt to recover user queries by exploiting leaks and prior knowledge. The server can access all encrypted documents. In searchable symmetric encryption, only the document contents and indexes are encrypted, while the semantic information of keywords (e.g., word meaning, contextual relations) naturally exists in plaintext space. Although an attacker cannot directly obtain the plaintext of user queries, they can extract the keyword space W_s from auxiliary data documents and independently perform semantic embedding on these keywords. Since semantic embedding relies solely on public corpora and pre-trained models and does not require any user private data, the attacker is fully capable of incorporating this information into the attack process. Relying solely on volume, frequency, and co-occurrence matrices, the attacker struggles to distinguish between queries with similar statistical features (e.g., two low-frequency, low-volume keywords may have nearly identical volume and frequency values). This fully demonstrates the ambiguity inherent in statistical information, whereas semantic embedding provides orthogonal discriminative capability. Query recovery essentially aligns observed ciphertext queries (with observable volume, frequency, and co-occurrence patterns) with plaintext keywords. Semantically similar words often exhibit similar document occurrence patterns (e.g., "apple" and "fruit" may appear in similar document contexts). Therefore, semantic embedding and co-occurrence-based statistical features are naturally complementary.

Attackers use leaks to infer the frequency, volume and co-occurrence information of queries. Suppose a user issues s queries, represented as Q^s, and l different queries are identified by using the search frequency. Mark the query list identified by the attacker as $Q_r = [q_1, q_2,..., q_l]$. The attacker observed the documents returned by the query $[D(q_1), D(q_2),..., D(q_l)]$. The server knows the total number of documents and records it as $|D|$. For each query q, normalize the query volume: $v_{qi} = |D(q_i)| / |D|$, $V_r = [v_{q1}, v_{q2},..., v_{ql}]$ is the volume vector for all queries in the Q_r, with the component numerator being the number of documents matched by the query. The attacker uses the search pattern to know the frequency of q in Q^s. The frequency of q is: $f_q = Cnt(q)/|Q^s|$, where $Cnt(q)$ is the number of q in Q^s. $F_r = [f_{q1}, f_{q2},..., f_{ql}]$ is the frequency vector for all queries in Q_r. Based on the access pattern, an attacker can construct an $l \times |D|$ matrix ID_r, where the rows are queries and the columns are documents. If the search results of q_i include d_j, then $ID_r[i, j] = 1$; otherwise 0. Finally, construct the $l \times l$ co-occurrence matrix $C_r = ID_r ID_r^T / |D|$.

Suppose the attacker has a list of similar documents $D_s = [d_1, d_2,..., d_k]$, and uses the same algorithm to extract the keywords. Attackers can easily obtain a keyword space similar to the user's document[3], denoted as $W_s = [w_1, w_2,..., w_m]$, and construct a volume vector $V_s = [v_{w1}, v_{w2},..., v_{wm}]$, with $v_{wi} = |D_s(w_i)| / |D_s|$, where $|D_s(w_i)|$ is the number of documents containing the keyword w_i in D_s. The attacker constructs an ID_s from W_s and D_s, with the co-occurrence matrix $C_s = ID_s ID_s^T / |D_s|$. For W_s, the attacker obtained the similar search frequency $F_s = [f_{w1}, f_{w2},..., f_{wm}]$ through Google Trends [37], which is used to display the keyword trends and changes searched by users in various regions around the world on the Google search engine.

III. REFINE_ATK

A. Partition Query

Fewer keywords in the database have high volume and frequency. Users may frequently query these keywords, so it is necessary to first distinguish between high-volume and high-frequency queries. Fig. 2 and Fig. 3 shows the results of the queries ranked by volume and frequency. This study selects the top 1000 keywords without stopping words as W. The query is divided into four quadrants: HVHF, HVLF, LVHF and LVLF. "L" and "H" represent "low" and "high", while "V" and "F" represent "volume" and "frequency". High-volume (frequency) queries refer to those ranking in the top 10% in terms of volume (frequency), while low-volume (frequency) queries account for the remaining 90%.

Rank the queries based on volume and frequency, and display the queries in different quadrants according to the ranking. The HVHF query ranking in the upper right corner is high, but the quantity is small. The ranking of the LVLF query in the lower left corner is low, but the number is large.

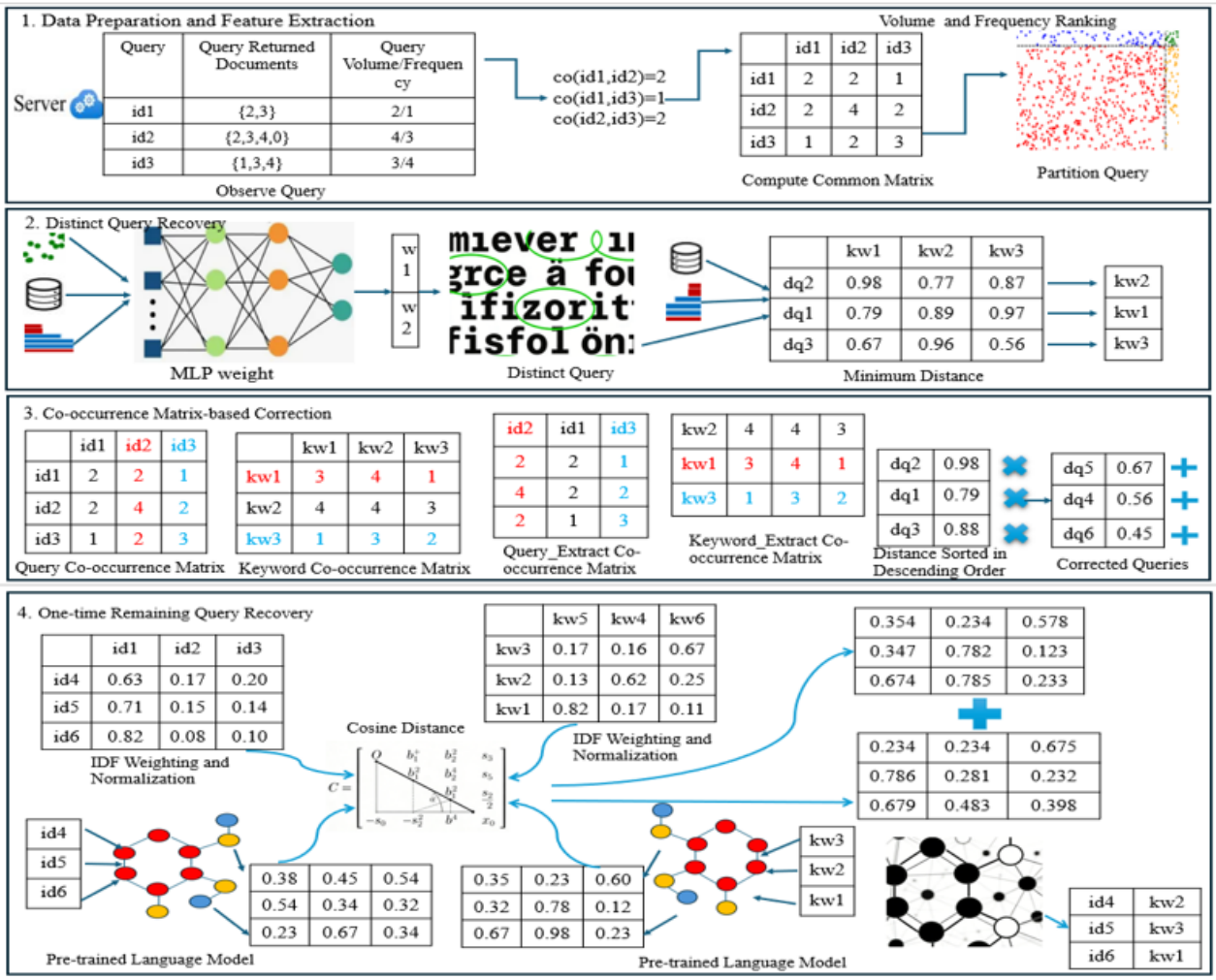

Fig. 1. An overview of the refine_atk (Ours) operation pipeline.

The attack accuracy rates in the HVLF quadrant in Enron and Lucene were 63% and 86%, respectively, 84% and 72% for LVHF, 26% and 34% for LVLF, and all were over 90% for HVHF. It indicates that the LVLF query is difficult to recover.

B. Identify and Recover Distinctive Queries

Algorithm 1 inputs the Td_r, V_r, F_r of the leaked exported data in the SSE query and similar data W_s, V_s, V_s. Output the *BaseRec* predictions. *BaseRec* refers to the number of queries recovered by this module. The *BaseRec* should not be too large. If the *BaseRec* is too large, there will be more predictions for non-HVHF queries, and the accuracy rate will decline.

First, the HVHF, HVLF, LVHF, and LVLF queries are divided. MLP uses the HVHF query, F_r and V_r learn the weights of volume and frequency, which are respectively denoted as w_V and w_F. Transpose F_r and V_r into column vectors and perform a logarithmic transformation (The frequency is processed in the same manner) [see Eq. (1)]:

$$\log V = \ln(V + \epsilon) \tag{1}$$

Element-by-element multiplication and division [see Eq. (2)]:

$$V \times F = V \odot F \tag{2}$$

$$V_{divF} = \frac{V + \epsilon}{F + \epsilon} \tag{3}$$

Calculate the ascending ranking for F_r and V_r, respectively. The smallest element ranks 0, the second smallest ranks 1, and so on. The largest ranking is n-1, where n is the total number of queries. Normalize the ranking vector to the interval [0,1]:

$$\text{ranks}_V = \frac{\text{argsort}(\text{argsort}(V))}{n-1} \tag{4}$$

Concatenate the features V, F, $logV$, $logF$, $V \times F$, V_{divF}, F_{divV} [Swap the numerator and denominator, see Eq. (3)], $ranks_V$ and $ranks_F$ [Replace the numerator with the frequency see Eq. (4)] by column. If the query $\in$ HVHF, the label is marked as 1; otherwise 0. If the number of HVHF queries

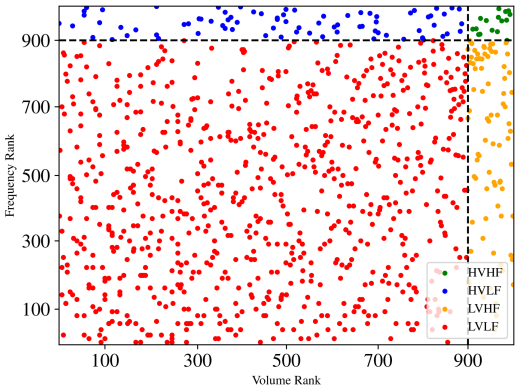

Fig. 2. Query distribution divided into different quadrants based on volume and frequency rank in Enron.

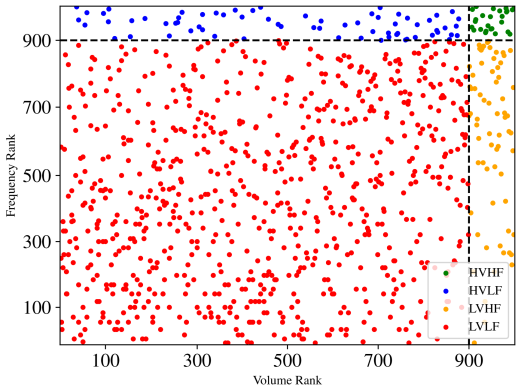

Fig. 3. Query distribution divided into different quadrants based on volume and frequency rank in Lucene.

Algorithm 1 Locate and recover distinctive queries

procedure LocRecDQ($\mathrm{Td}_r, \mathrm{V}_r, \mathrm{F}_r, \mathrm{W}_s, \mathrm{V}_s, \mathrm{F}_s$, params)
// Outline: learn weights using MLP, compute the differential distance, select the distinctive queries, and then perform recovery.Params include: BaseRec, nb_queries.
// 1. Partition the quadrants and take HFHV (or another subset) as the training set.
$(\mathrm{HFHV}, \mathrm{HFLV}, \mathrm{LFHV}, \mathrm{LFLV}) \leftarrow$ plot_histogram_and_classify($\mathrm{F}_r, \mathrm{V}_r$)
// 2. Use MLP to learn the weight vector on the selected subset (output denoted w).
TrainSet $\leftarrow$ HFHV
w $\leftarrow$ learn_weight_mlp(TrainSet, $\mathrm{F}_r, \mathrm{V}_r$)
// 3. Compute the differential distance vector with the learned weights and select the distinctive queries.
$\mathrm{D_{FV}} \leftarrow$ calculate_dVF(w)
// 4. Obtain known query pairs and learn the recovery coefficient α.
known_pairs $\leftarrow$ get_known_queries(nb_queries = params.nb_queries)
$\alpha \leftarrow$ learn_recover_weight_alpha(known_pairs)
// 5. Perform recovery mapping on the distinctive queries.
Pred $\leftarrow$ RecoverByVF(HFHV, $\mathrm{W}_s, \mathrm{V}_s, \mathrm{F}_s, \alpha$)
return Pred
end procedure

is less than 5% of the total number of queries, oversampling is performed. Specifically, the query index for HVHF is pos_idx, and the query index for the rest is neg_idx. The number of HVHF queries is the maximum value between 50 and 10% of the total number of queries. The number of repetitions, repeat_times, is the number of HVHF queries divided by the length of pos_idx, repeat the HVHF query features and labels repeat_times, and concatenate them with the remaining queries. Finally, shuffle the order. Divide the training set and the validation set (80% training and 20% validation). Use StandardScaler (The validation set is treated in the same way) [see Eq. (5)]:

$$X_{\text{train}_r} = \frac{X_{\text{train}} - \mu_{\text{train}}}{\sigma_{\text{train}}} \tag{5}$$

The hidden layer size of MLPClassifier is (64, 32), the activation function is relu, the solver is adam, an early stop strategy is set, and the maximum iteration is 500 times. Repeatedly shuffle a certain feature column on the validation set to see how much the model performance drops. The more the performance drops, the more important this feature is. Each feature is shuffled 20 times and the average is taken. Each feature has an importance score. Aggregate feature importance to *V* and *F*. For each feature *i*, its importance is I_i, and it is assigned according to the feature name (Apply the same processing to the frequency) [see Eq. (6)]:

$$\mathrm{imp}_V = \sum_{\text{only } V} I_i + \sum_{V \text{ and } F} \frac{I_i}{2} + \sum_{\text{others}} \frac{I_i}{2} \tag{6}$$

Only *V* refers to a feature *i*, where the feature name contains only *V*, with an importance of I_i. *V* and *F* indicates that the feature name includes both *V* and *F*, with an importance of $\frac{I_i}{2}$. Others refers to all other features, with an importance of $\frac{I_i}{2}$. The weight calculation for volume and frequency is simplified as follows [see Eq. (7) to Eq. (9)]:

$$w = \left[\frac{\mathrm{imp}_V}{|\mathrm{imp}_V| + |\mathrm{imp}_F|}, \frac{\mathrm{imp}_F}{|\mathrm{imp}_V| + |\mathrm{imp}_F|} \right] \tag{7}$$

$$w_V = \frac{\mathrm{imp}_V}{|\mathrm{imp}_V| + |\mathrm{imp}_F|} \tag{8}$$

$$w_F = \frac{\mathrm{imp}_F}{|\mathrm{imp}_V| + |\mathrm{imp}_F|} \tag{9}$$

Calculate the differential distance to identify unique queries from all the queries. The differential distance d_{qi} of query q_i is [see Eq. (10)]:

$$d_{q_i} = \min_{q_j \in Q_r \wedge j \neq i} w_V \cdot |v_{q_i} - v_{q_j}| + w_F \cdot |f_{q_i} - f_{q_j}| \tag{10}$$

w_V and w_F denote the weights associated with volume and frequency, respectively. The queries are sorted in descending order based on their differential distance. The top *BaseRec* queries constitute the distinctive queries. A small set of known queries is utilized to learn the weight parameter α, which is used to recover distinctive queries. The specific value of α is determined by the statistical characteristics (volume and frequency) of the known queries and their correspondence with keywords in the auxiliary data. Known queries are used solely as a tool for internal parameter estimation and are not directly leveraged as prior knowledge for computing the final attack accuracy, thereby ensuring fairness in comparison with other baseline attacks (e.g., RSA, Jigsaw). This study experimentally validates (see Fig. 20) that with a small number of known queries (1, 2, 5), Algorithm 1 can still achieve relatively stable accuracy (although accuracy fluctuates more significantly when the number is extremely small). This demonstrates that the assumption is reasonable and feasible in practice. Attacker only needs a very small number of random known queries to effectively estimate α.

Following the definition of distance between queries and keywords from[22], each query is paired with its nearest keyword. The distance $s(q_i, w_j)$ between a true query q_i and a keyword $w_j \in W_s$ is defined as Eq. (11):

$$s(q_i, w_j) = \alpha \cdot |v_{q_i} - v_{w_j}| + (1 - \alpha)|f_{q_i} - f_{w_j}| \tag{11}$$

recovery the query q_i as Eq. (12):

$$\text{Pred}(q_i) = \arg \min_{w_j \in W_s} s(q_i, w_j) \tag{12}$$

By dynamically allocating the weights of volume and frequency, this module is capable of recovering unique queries.

C. Correction Through Co-occurrence Matrix

The query-keyword pairs output by the co-occurrence matrix correction of Algorithm 1 is given. The accurate recovery of HVHF queries is crucial for the recovery of subsequent queries.

The input of Algorithm 2 is the co-occurrence matrices C'_r and C'_s, corresponding respectively to the query and keyword in *Pred*, *BaseRec*, and *Pred* recovered by Algorithm 1. That is, in *Pred*, the columns of the query index in the co-occurrence matrix C_r are taken as C'_r, and the rows of the keyword index in the co-occurrence matrix C_s are taken as C'_s. Then, each row of C'_r and C'_s is divided by the sum of the rows to normalize. *ConfRec*($\leq$ *BaseRec*) refers to the number of queries correctly recovered in *Pred*. The smaller the *ConfRec* is, the higher the accuracy rate will be and the fewer queries can be recovered. For the correct prediction (q_i, w_i) in *Pred*, C'_r*[i]* and C'_s*[i]* are similar, and their volume and frequency are similar; Conversely, q_i and w_i are only similar in volume and frequency. The reverse confidence revconf of (q_i, w_i) is Eq. (13):

$$\text{revconf} = \|C'_r[i] - C'_s[i]\| \tag{13}$$

Algorithm 2 Verification based on co-occurrence matrix

procedure VERIFY($Pred, C'_r, C'_s, BaseRec, ConfRec$) //
// Extract the columns and rows corresponding to the queries and keywords in the co-occurrence matrices C_r and C_s of Algorithm 1 $Pred$, respectively, to construct C'_r and C'_s.
  $TempPred \leftarrow Pred$
  $Td' \leftarrow Pred.td$
  $Revconf \leftarrow \emptyset$
  for $i \in [0, |Td'| - 1]$ **do**
    revconf $\leftarrow \|C'_r[i] - C'_s[i]\|$
    append($Td'[i]$, revconf) to $Revconf$
  end for
  sort $Revconf$ in descending order by the revconf value;
  for $i \in [0, BaseRec - ConfRec - 1]$ **do**
    $(td, revconf) \leftarrow Revconf[i]$
    remove the prediction for td from $TempPred$
  end for
  return $TempPred$
end procedure

The greater the *revconf*, less reliable it is for q_i to predict w_i, that is, the plaintext keywords at the bottom of q_i are very likely not w_i. Calculate the *revconf* of each query in *Pred* and add it to the *Revconf*. Sort the *Revconf* in descending order. Remove the first *BaseRec-ConfRec* queries from *Pred*. Return *ConfRec* query predictions.

D. Recovering the Remaining Queries Globally

The input of Algorithm 3 include: the query-keyword pair *Pred* recovered by Algorithm 2, the co-occurrence matrix C_r between the unrecovered and recovered query, all queries T_r, the similar keyword space W_s, and the co-occurrence matrix C_s between the unpaired and paired keyword. U_{td} stands for unrecovered queries, and U_{kw} stands for unpaired keywords.

Calculate TF-IDF weight to each row of $\widetilde{C}_r$ and $\widetilde{C}_s$ [see Eq. (14) and Eq. (15)]:

$$idf(j) = \log \frac{N}{df(j) + 1} \tag{14}$$

$$r_w[j] = r[j] \times idf(j) \tag{15}$$

N represents the number of unrecovered queries, df is the number of non-zero elements in each column of the $\widetilde{C}_r$ matrix, j is the index, and r is the $\widetilde{C}_r$ matrix. Calculate the cosine distance between the two weighted co-occurrence matrices [see Eq. (16)]:

$$D_{co}[i, j] = 1 - \frac{\tilde{C}_r[U_{td}[i]] \cdot \tilde{C}_s[U_{kw}[j]]^T}{\|\tilde{C}_r[U_{td}[i]]\| \cdot \|\tilde{C}_s[U_{kw}[j]]\|} \tag{16}$$

The MiniLM-L6 pre-trained language model has 6 hidden layers and 12 attention heads. The dimension of the hidden layer is 384 dimensions, the maximum sequence length is 128 tokens, and mean pooling is adopted. MiniLM is a

lightweight variant of BERT, with fewer parameters and a more streamlined structure than the standard BERT. Encode the unrecovered queries and unpaired keywords using MiniLM-L6, normalize the generated embedding vectors, and convert them into numpy matrices. In this matrix, the embedding vectors of unrecovered queries and unpaired keywords are concatenated column-wise, respectively, and their indices remain fixed throughout the entire attack process, which is critical to query recovery. For the indexes of unrecovered queries and unpaired keywords, extract the corresponding embedding matrix row vectors to form new embedding submatrices, and calculate the cosine distance between the two embedding submatrices [see Eq. (17)]:

$$D_{emb}[i,j] = 1 - \frac{E_{td}[i] \cdot E_{kw}[j]^T}{\|E_{td}[i]\| \cdot \|E_{kw}[j]^T\|} \tag{17}$$

E_{td} and E_{kw} are embedding submatrices for unrecovered queries and unpaired keywords. The cost matrix is a linear combination of D_{co} and D_{emb} [see Eq. (18)]:

$$M[i,j] = \beta \cdot D_{co}[i,j] + (1-\beta) \cdot D_{emb}[i,j] \tag{18}$$

M is the cost matrix, β is the weight of $D_{co}[i, j]$, and 1-β is the weight of $D_{emb}[i, j]$. This study uses a greedy graph to match unrecovered queries and unpaired keywords. M_{td} and M_{kw} are two empty sets. The row index and column index of matric *M* element, as well as the element values, are added to flat as tuples to sort each element of *M* in ascending order. For each tuple's row index and column index, if neither is in M_{td} or M_{kw}, take the value of the row index in U_{td}, denoted as *td*, and take the value of the column index in U_{kw}, denoted as *kw*. (*td*, *kw*) is added as a matching pair in P_{final} [see Eq. (19)]:

$$P_{final}[td] = kw \tag{19}$$

The M_{td} and M_{kw} collections, respectively add row index and column index [see Eq. (20)]:

$$\begin{cases} M_{td} \leftarrow M_{td} \cup \{i\}, & i \notin M_{td},\ j \notin M_{kw} \\ M_{kw} \leftarrow M_{kw} \cup \{j\}, & j \notin M_{kw},\ i \notin M_{td} \end{cases} \tag{20}$$

The known queries in Algorithm 1 are not the high-volume known queries in RSA [14], but are randomly extracted from the user query sequence and the keyword space.

E. Experimental Data and Evaluation Metrics

The experimental dataset consists of Enron and Lucene. The Enron email Corpus [33] was collected from 2000 to 2002, containing a total of 30,109 emails. The Lucene mailing list consists of 66,491 emails from 2001 to 2020 [34]. This study uses the preprocessed version provided in reference [12].

The search frequency of keywords in Enron and Lucene is generated using Google Trends, which contains Google search trends for approximately 260 weeks from October 2016 to October 2021. The attacker auxiliary knowledge is the sum of the search frequency for each keyword within 1 to 50 weeks. Divide by the total to normalize the frequency of each keyword, denoted as F_s. The user query is simulated and generated based on the total frequency *F* from 1 + τ to 50 + τ. τ is the time gap between the attacker auxiliary knowledge and the user query.

Randomly shuffle all the documents and divide them into two disjoint subsets of equal size. One subset serves as the user's encrypted database D_r, and the other subset as similar data D_s. The attacker generates W_s, V_s and C_s based on D_s, and observes all user queries to obtain Q_r, V_r, F_r and C_r. Each

Algorithm 3 Recover all queries using co-occurrence and embedding.

procedure RECOVERALL($Pred, T_r, W_s, C_r, C_s$)
Input: $Pred$: Algorithm 2 predictions (map td $\rightarrow$ kw); T_r: all trap set; W_s: similar keyword universe; C_r: co-occurrence matrix (rows: unrecovered vs recovered queries); C_s: co-occurrence matrix (rows: unpaired vs paired keywords); $\beta \in [0,1]$, use_emb (flag), optional encoder $\mathcal{E}(\cdot)$.
$P_{final} \Leftarrow Pred$
$U_{td} \Leftarrow T_r \backslash \text{dom}(P_{final})$
$U_{kw} \Leftarrow W_s \backslash \text{range}(P_{final})$
$\widetilde{C}_r \Leftarrow C_r$, $\widetilde{C}_s \Leftarrow C_s$
$idf \Leftarrow$ compute IDF vector over columns of $\widetilde{C}_r$ and $\widetilde{C}_s$
while each row r of $\widetilde{C}_r$ **do**
    $r \Leftarrow r \odot \text{idf}$
    $r \Leftarrow r / \|r\|_2$
end while
while each row s of $\widetilde{C}_s$ **do**
    $s \Leftarrow s \odot \text{idf}$
    $s \Leftarrow s / \|s\|_2$
end while
$D_{co} \Leftarrow$ pairwise cosine-distance between rows of $\widetilde{C}_r[U_{td}]$ and $\widetilde{C}_s[U_{kw}]$
if $use\_emb = true$ and $\mathcal{E}$ available **then**
    $E_{td} \Leftarrow \mathcal{E}(U_{td})$
    $E_{td} \Leftarrow$ row-normalize(E_{td})
    $E_{kw} \Leftarrow \mathcal{E}(U_{kw})$
    $E_{kw} \Leftarrow$ row-normalize(E_{kw})
    $D_{emb} \Leftarrow$ pairwise cosine-distance between E_{td} and E_{kw}
else
    $D_{emb} \Leftarrow 0$ (same shape as D_{co})
end if
$M \Leftarrow \beta D_{co} + (1-\beta) D_{emb}$
$\mathcal{M}_{td} \Leftarrow \emptyset$, $\mathcal{M}_{kw} \Leftarrow \emptyset$
$L \Leftarrow$ list of tuples (i, j, M_{ij}) for all valid i, j
sort L ascending by third element (cost)
while each (i, j, ...) in L **do**
    if $i \notin \mathcal{M}_{td}$ and $j \notin \mathcal{M}_{kw}$ **then**
        $id \Leftarrow$ the i-th element of U_{td}
        $kw \Leftarrow$ the j-th element of U_{kw}
        $P_{final}[id] \Leftarrow kw$
        $\mathcal{M}_{td} \Leftarrow \mathcal{M}_{td} \cup \{i\}$
        $\mathcal{M}_{kw} \Leftarrow \mathcal{M}_{kw} \cup \{j\}$
    end if
end while
return P_{final}
end procedure

combination of experimental parameters was independently simulated 30 times. Each simulation randomly selects half of the documents as D_s and generates queries based on the frequency F.

Aggressiveness can be evaluated by accuracy rate, recovery rate and attack time. The recovery rate is the proportion of queries that have been recovered among all queries. Accuracy refers to the queries that have been correctly recovered among the recovered queries. The attack time is the time spent executing Algorithm 1 to Algorithm 3, excluding the time for generating the co-occurrence matrix, query generation, generating the keyword-document matrix, and loading the pre-trained language model [see Eq. (21) to Eq. (23)]:

$$\mathrm{ACC} = \frac{|\mathrm{Recovered}(Q^s)|}{|Q^s|} \tag{21}$$

$$\mathrm{QRR} = \frac{|\mathrm{CorrectRec}(Q^s)|}{|\mathrm{Recovered}(Q^s)|} \tag{22}$$

$$\mathrm{Atk\_time} = \mathrm{Alg1.time} + \mathrm{Alg2.time} + \mathrm{Alg3.time} \tag{23}$$

F. Performance of Algorithm 1 and Algorithm 2

In Table II, the parameter experiments are conducted, showing the accuracy, recovery rate and the number of correct recovery of Algorithm 1 and Algorithm 2 in Enron and Lucene when *BaseRec* $\in \{25, 50, 100, 200, 400\}$ and *ConfRec*/*BaseRec* $\in \{1, 0.5, 0.2\}$. part = 1 indicates that Algorithm 2 retains all recovery results of Algorithm 1. The first two columns of data under *ConfRec*/*BaseRec* $\in \{1, 0.5, 0.2\}$ are represented as decimals.

The accuracy rate of various parameter combinations exceeds 65%. As *BaseRec* and *ConfRec*/*BaseRec* decrease, the number of queries recovered by Algorithm 2 decreases, the recovery rate drops, but the accuracy rate increases. In Lucene, when *BaseRec* = 50 and *ConfRec*/*BaseRec* = 0.2, the accuracy rate drops by 1%. When *BaseRec* = 25 and *ConfRec*/*BaseRec* = 0.2, Algorithm 2 achieved recovery rates of approximately 0.66% and 2.04% in Enron and Lucene, respectively, but with extremely high accuracy. The attack proposed in this study outperforms the accuracy and the number of correct recovery achieved in reference [22]. *BaseRec* = 400, *ConfRec*/*BaseRec* = 1, the accuracy rate increased by 23% and 17%, respectively, the number of correct recovery increased by 63 and 57.

Algorithm 1 mainly involves MLP learning the weight of volume and frequency through HVHF query, calculating the differential distance, and identifying unique queries. A coefficient α for recovering unique queries is obtained by using a small number of known queries, and finally the unique queries are recovered by volume and frequency. The attack parameter *BaseRec* represents the total number of queries $|Q_r|$. Each dataset independently simulates 30 experiments, collecting the weights, α , accuracy of Algorithm 1, and attack time of each experiment. The collected weight and α array are divided into buckets with a width of 0.05. The mean and standard deviation of the accuracy within each bucket are calculated. The bucking is done to ensure that all values in both arrays are covered and to average the attack processing time. Test the influence of different weights and α on the accuracy and attack time of Algorithm 1.

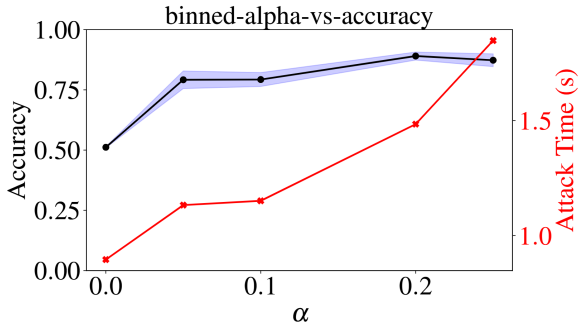

Fig. 4. Accuracy and attack time of Algorithm 1 with different weights and α in Enron.

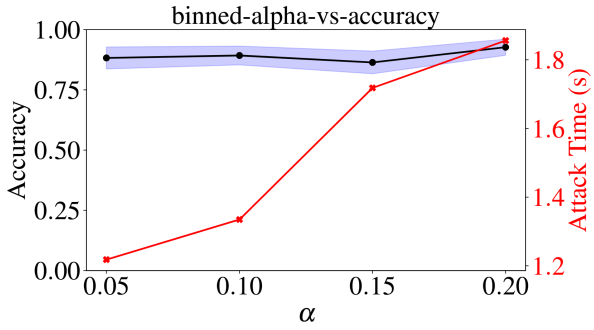

Fig. 5. Accuracy and attack time of Algorithm 1 with different weights and α in Lucene.

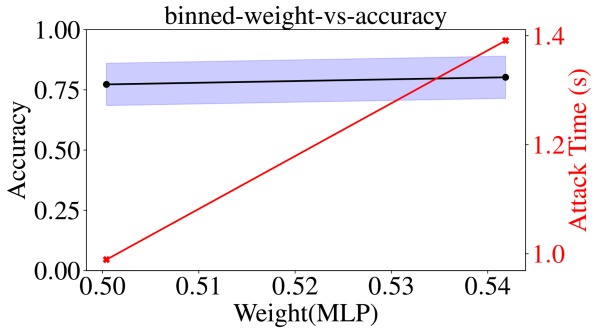

Fig. 6. Accuracy and attack time of Algorithm 1 with different weights and accuracy in Enron.

Fig. 4 shows the results in Enron, where α is 0.2, the accuracy rate is the highest, approximately 87%, and the attack time is about 1.5 seconds. α was selected as 0.05. Fig. 5 shows the results in Lucene, α is selected as 0.05, the attack time is approximately 1.3 seconds, and the accuracy rate is about 87%. In Fig. 6, when the weight keeps changing, the accuracy rate remains basically unchanged. The weight is selected as 0.5, with an accuracy rate of approximately 77% and an attack time of about 1.0 seconds. In Fig. 7, the weight is 0.55, with the highest accuracy rate of approximately 90%, and the

TABLE II. RESULT OF ALGORITHM 1 AND 2 IN ENRON AND LUCENE.

Dataset	BaseRec	part=1			part=0.5			part=0.2		
		ACC	QRR	CRECNUM	ACC	QRR	CRECNUM	ACC	QRR	CRECNUM
Enron	25	0.9942	0.2276	24.4	1.0000	0.0271	12.0	1.0000	0.0066	5.0
	50	0.9600	0.3149	43.3	0.9974	0.0721	24.8	1.0000	0.0178	10.0
	100	0.8986	0.4259	76.5	0.9645	0.1481	45.7	1.0000	0.0453	20.0
	200	0.7779	0.5964	121.6	0.9233	0.2767	82.4	0.9835	0.0840	38.5
	400	0.6728	0.7933	175.2	0.8686	0.4990	130.3	0.9680	0.1602	73.2
Lucene	25	0.9990	0.2835	24.9	1.0000	0.0405	12.0	1.0000	0.0204	5.0
	50	0.9888	0.3547	48.2	0.9997	0.0786	25.0	0.9859	0.0339	9.6
	100	0.9332	0.4747	86.1	0.9788	0.1912	48.3	1.0000	0.0499	20.0
	200	0.8522	0.6417	143.8	0.9897	0.3401	95.5	1.0000	0.1036	40.0
	400	0.7305	0.8301	204.8	0.9292	0.5282	150.5	0.9956	0.2093	78.6

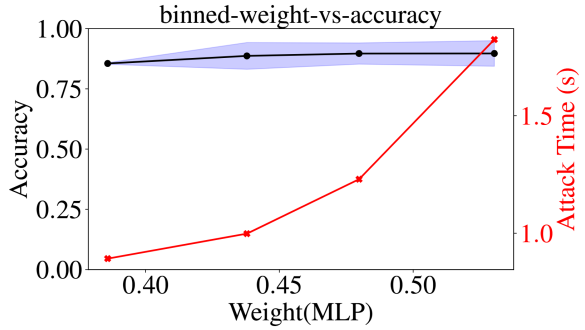

Fig. 7. Accuracy and attack time of Algorithm 1 with different weights and accuracy in Lucene.

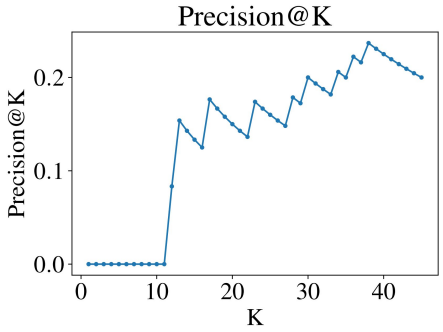

Fig. 8. Result of P_K for Algorithm 1 using grid search for weights in Enron.

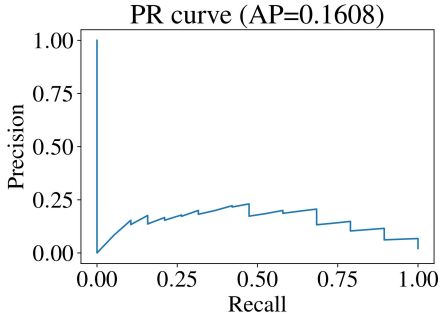

Fig. 9. Result of AP for Algorithm 1 using grid search for weights in Enron

attack time is about 1.8 seconds. The MLP can model the relationship between volume and frequency statistical features, thereby better adapting to the distribution discrepancy between the auxiliary dataset and the target dataset. The learned weights can dynamically adjust the contribution proportions of volume and frequency features, thus improving the generalization performance of the attack under different query distributions. In practical attack scenarios where the attacker only possesses incomplete auxiliary knowledge — for example, when the scale of the auxiliary document set is only half that of the target set — the MLP can mine latent correlations from limited data and maintain high query recovery accuracy.

Algorithm 1 conducts a comparative experiment of grid search and MLP learning volume and frequency weight to identify and recover unique queries. *BaseRec* is 45. Collect the differential distance. Manually divided HVHF queries are the actual queries to be recovered, as well as the prediction results of Algorithm 1 and F_r, V_r. Test the influence of grid search weight and MLP learning weight on Algorithm 1.

Fig. 8 and Fig. 11 are the Precision@K result graphs of the grid search weights of Algorithm 1 in Enron and Lucene. Algorithm 1, which only utilizes the grid search weight to recover the results of unique queries in Top-K, is indicated to be not ideal. Algorithm 1 focuses on how many of the top K of its prediction results are correct recoveries for unique queries. Because Algorithm 1 has a poor ability to recover unique queries, it will affect the performance of the entire attack algorithm. Algorithms with high classification accuracy but low Top-K accuracy are of no use to attacks. Fig. 9 and Fig. 12 show the PR curves in Enron and Lucene respectively. The average precision AP in Enron is 0.1608, and that in Lucene is 0.4058. Fig. 10 and Fig. 13 show the ROC curves in Enron and Lucene. The AUC value in Enron is 0.9204, and that in Lucene is 0.9712. In [22], the authors indicate that the keyword universe size is 1000, and the number of unique queries is approximately 18. Positive samples (unique queries) are extremely unbalanced, while PR and ROC curves are more sensitive and meaningful to the problem of sample imbalance.

Fig. 14 and Fig. 17 are the Precision@K result graphs

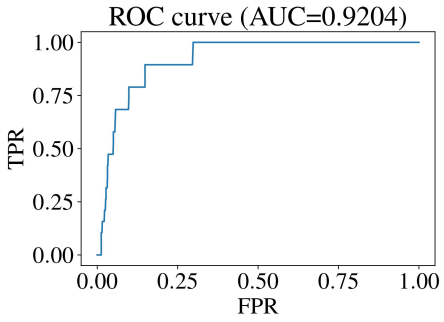

Fig. 10. Result of AUC for Algorithm 1 using grid search for weights in Enron

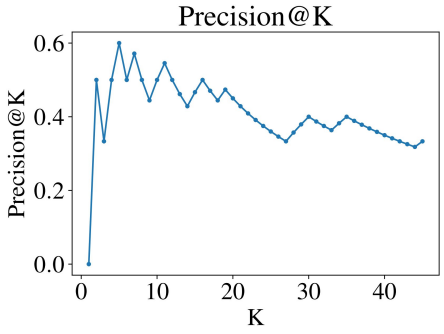

Fig. 11. Result of P_K for Algorithm 1 using grid search for weights in Lucene

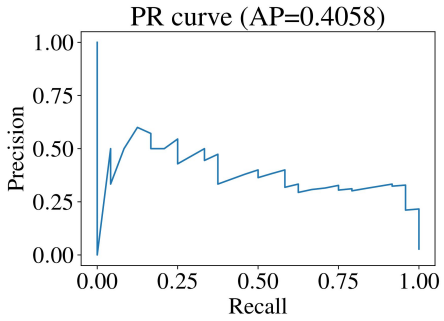

Fig. 12. Result of AP for Algorithm 1 using grid search for weights in Lucene

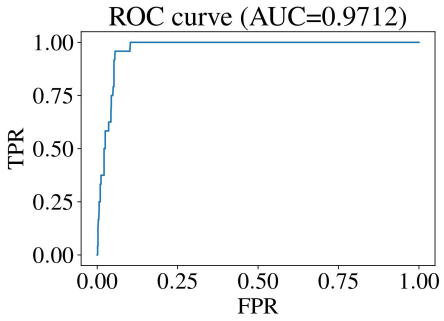

Fig. 13. Result of AUC for Algorithm 1 using grid search for weights in Lucene

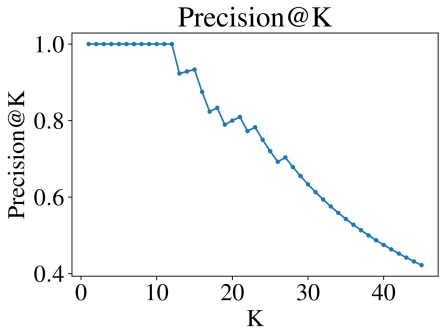

Fig. 14. Result of P_K AUC for Algorithm 1 using MLP to learn weight in Enron

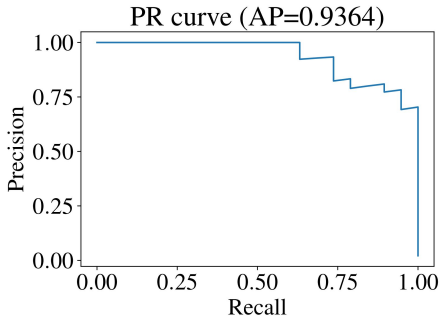

Fig. 15. Result of AP for Algorithm 1 using MLP to learn weight in Enron

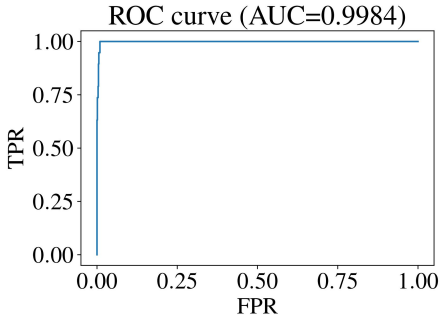

Fig. 16. Result of AUC for Algorithm 1 using MLP to learn weight in Enron

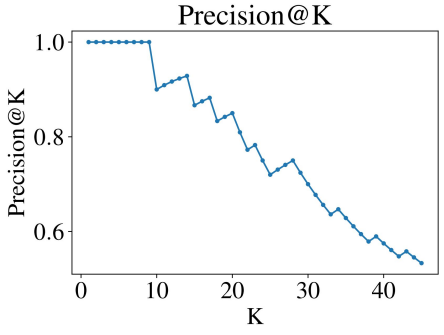

Fig. 17. Result of P_K for Algorithm 1 using MLP to learn weight in Lucene

of Algorithm 1 using MLP to learn the weights of volume and frequency in Enron and Lucene. In Enron and Lucene,

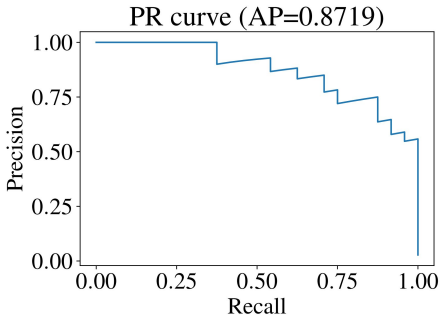

Fig. 18. Result of AP for Algorithm 1 using MLP to learn weight in Lucene

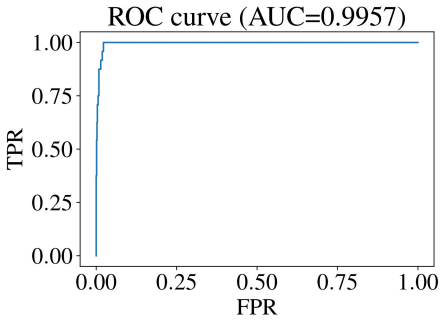

Fig. 19. Result of AUC for Algorithm 1 using MLP to learn weight in Lucene

the accuracy of K at 10 is nearly 1, Under Lucene, when K is between 0 and 10, the highest accuracy of the grid search method is only 0.6 (Fig. 11), while the accuracy of the MLP method is nearly 1 (Fig. 17). By comparing, it can be concluded that, regardless of which dataset, when K is any value within the interval, the MLP method is much more accurate than the grid search method. The accuracy of the MLP method in the Top-10 is nearly 1, significantly revealing that the Top 10 recovery results of Algorithm 1 are all recoveries of unique queries. The AP values in Enron and Lucene are 0.9364 (Fig. 15) and 0.8719 (Fig. 18), respectively. The AUC values are 0.9957 (Fig. 19) and 0.9984 (Fig. 16). In terms of AP values, the MLP method outperformed the grid search method by 83% and 53%, respectively, AUC values were 8% and 2% higher.

Algorithm 1 uses a small number of known queries to obtain a coefficient α for recovering the target query, and finally recovers the target query using volume and frequency. Test the impact of the known number of queries on the accuracy and attack time of Algorithm 1. Collect the accuracy rate and attack time of Algorithm 1 for each experiment under different known query numbers.

Fig. 20, when it is known that the number of queries is 1 and 2, the accuracy of Algorithm 1 is not stable. The lower quartile differs significantly from the upper quartile, and the data is highly dispersed and fluctuates greatly. Excluding outliers, the highest accuracy rate is 92%. In Fig. 21, when the number of known queries is small, the main reason for the unstable accuracy is that the selection of the Algorithm 1 coefficient α depends on the known queries. When the number

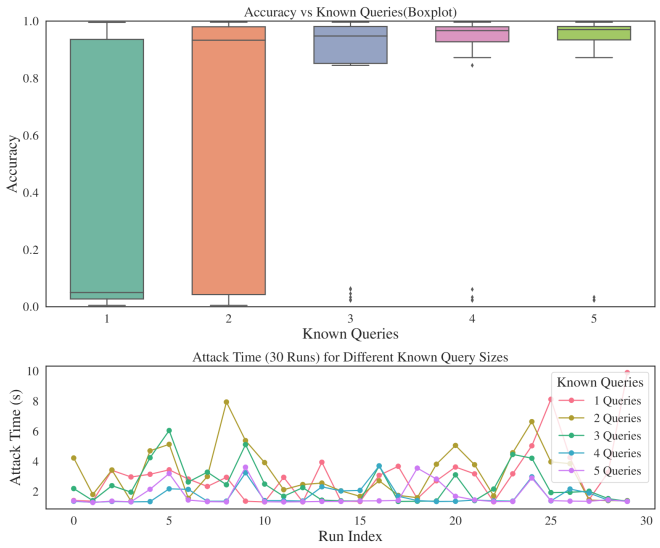

Fig. 20. Accuracy and per-attack time of Algorithm 1 with different known queries in Enron.

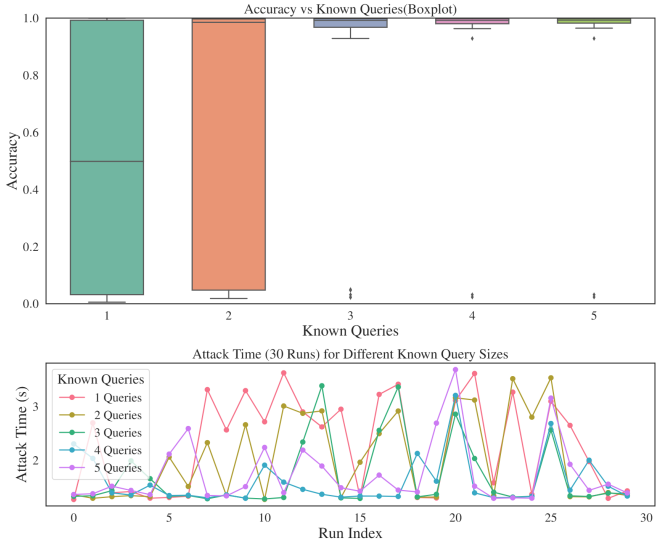

Fig. 21. Accuracy and per-attack time of Algorithm 1 with different known queries in Lucene.

is small, it is insufficient to support the selection of coefficients, resulting in unstable accuracy of Algorithm 1. The specific coefficients are shown in Fig. 4 and Fig. 5.

G. Attack Results and Durability

The following are the attack results of the entire algorithm and the impact of β on accuracy. *BaseRec* is 100 and *ConfRec* is 50. The size of the keyword space is 1000, the number of queries observed each week is 2000, and the total observation period is 50 weeks.

When $\beta = 0$, query recovery only relies on the embedding sub-matrix. Although it has a relatively high accuracy rate, in order to utilize the knowledge of weighted co-occurrence

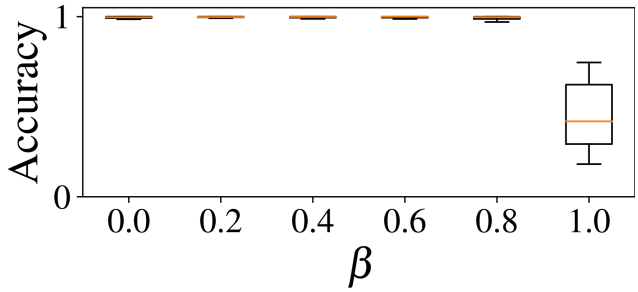

Fig. 22. Accuracy of the refine_atk under different β with embedding matrices in Enron.

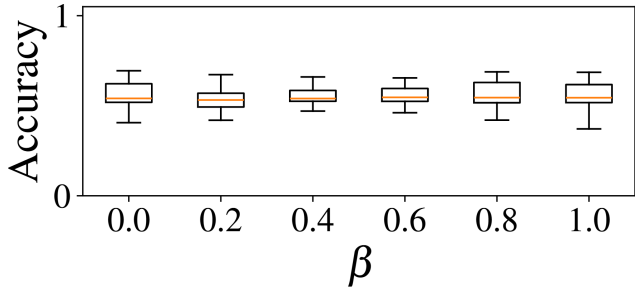

Fig. 23. Accuracy of the refine_atk under different β without embedding matrices in Enron.

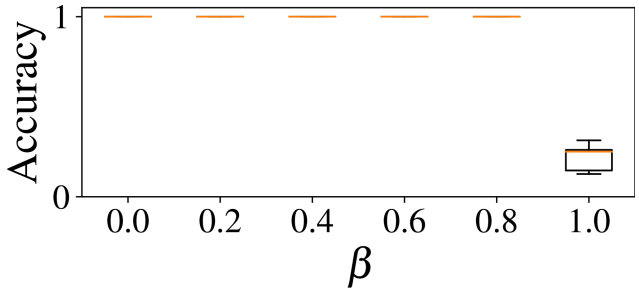

Fig. 24. Accuracy of the refine_atk under different β with embedding matrices in Lucene.

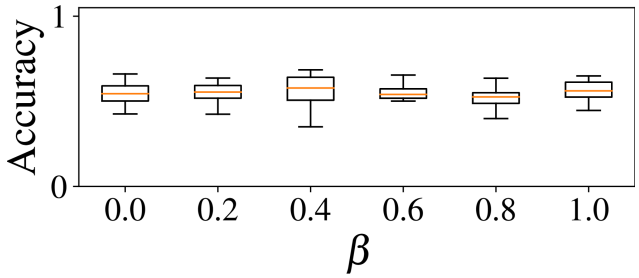

Fig. 25. Accuracy of the refine_atk under different β without embedding matrices in Lucene.

matrices, β is not set to 0. This indirectly reflects the significance of embedding sub-matrices for query recovery. Fig. 22 and Fig. 24 show the experimental results of the β value in Algorithm 3. When β = 1, query recovery only relies on the weighted co-occurrence matrix, with accuracy rates of 40% and 30% in Enron and Lucene, respectively. As β is reduced, by introducing an embedding sub-matrix, the accuracy rate is improved. The accuracy rate in Enron and Lucene is as high as 98%. This indicates that the embedding sub-matrix is crucial to Algorithm 3 and significantly enhances the attack performance. Without the embedding sub-matrix, the attack accuracy rate drops to around 50% [Fig. 23 and Fig. 25].

The embedding sub-matrix was introduced in Algorithm 3. The absence of an embedding sub-matrix will not affect the performance of Algorithm 1 and Algorithm 2, and unique queries can still be identified and recovered with high accuracy.

TABLE III. RESULT OF ATTACK ACCURACY FOR OUTDATED FREQUENCY UNDER DIFFERENT τ VALUES (IN WEEKS).

Dataset	τ (in weeks)			
	$\tau = 10$w	$\tau = 50$w	$\tau = 100$w	$\tau = 150$w
Enron	0.9879	0.9581	0.9272	0.9273
Lucene	0.9995	0.9774	0.9991	0.9890

The impact of "outdated" frequency on attack accuracy. "Outdated" refers to the time difference between the auxiliary frequency and the actual frequency. If the frequency information of the two overlaps less, the attack may fail. Durability refers to whether an attack can recover future queries by leveraging "outdated" frequency. Table III presents the experimental results for measuring the durability of attacks. The experiment used the frequency of Google Trends [37] from 1 to 50 weeks as the auxiliary information for attackers, and the frequency of 1 + τ to 50 + τ weeks was used to simulate the generation of user queries.

When τ is 10 and 50, the attack accuracy rates in Enron and Lucene decrease by 3% and 2%, respectively. When τ is 50 and 100, Enron drops by 3% and Lucene rises by 2%. When τ is 100 and 150, the attack accuracy rate in Enron remains basically unchanged, while it decreases by 1% in Lucene. The results show that the attack proposed in this study can still successfully recover future queries by leveraging the auxiliary frequency from a long time ago.

H. User Query Distribution

The impact of different query distribution on attack accuracy. The distribution for simulating the generation of user queries is "real", "poisson", "uniform", and "zipfian". "real" refers to the simulation of user query generation using the frequency of 1 + τ to 50 + τ weeks in Google Trends. "poisson" refers to generating query records that follow a Poisson distribution, simulating the random characteristic of "the number of times a keyword is queried within a unit of time". "uniform" refers to a uniform probability distribution, meaning that the probability of all keywords being queried is exactly the same, simulating the ideal situation where "users pay the same attention to all keywords". "zipfian" refers to generating queries that follow the Zipf distribution, which conforms to the practical rule that "a few keywords are frequently queried while the majority are infrequently queried". The attack parameters *BaseRec* is 45, *ConfRec* is 35, and β is 0.5. The size of the experimental parameter keyword space is 1000, the number of queries observed each week is 500, the observation period is 50, τ is 50, and the number of known queries is 15. The distribution of user queries generated

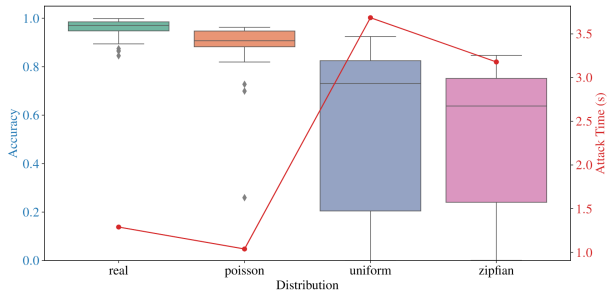

Fig. 26. Accuracy and average attack time of refine_atk under different distributions in Enron.

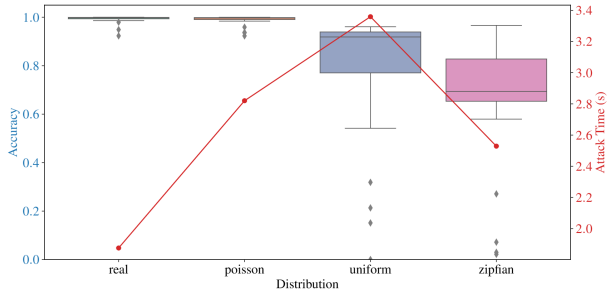

Fig. 27. Accuracy and average attack time of refine_atk under different distributions in Lucene.

is "real", "poisson", "uniform", and "zipfian". The following shows the results of the entire algorithm in Enron and Lucene.

In Fig. 26, the accuracy rate of the "real" distribution is as high as 97%. The average attack time of the "poisson" distribution is the shortest, at 1.1 seconds. The accuracy of "uniform" and "zipfian" is unstable because the queries generated by these two distributions contain a large number of difficult-to-recover LVLF queries. The average attack time of the "uniform" distribution is the longest, at 3.7 seconds. Fig. 27, the accuracy rates of the "real" and "poisson" distributions are 99%. The accuracy rates of "uniform" and "zipfian" are unstable, but perform better than Enron, with the lowest accuracy rate exceeding 60%. This is because the contents of the Lucene and Enron datasets are not related. Additionally, Lucene has a relatively large proportion of HVHF queries. The average attack time of the "uniform" distribution is the longest. The distribution that generates user queries has a certain impact on accuracy.

Table IV presents the results of paired t-tests, which compare the attack accuracy of the proposed refine_atk and the baseline Jigsaw Attack across two datasets (Enron and Lucene) under two keyword distributions (uniform and Zipfian). The core metrics include mean value, standard deviation, mean difference, p-value, Cohen's d effect size, and 95% confidence interval (CI).

Under the uniform distribution on the Enron dataset, refine_atk achieves an average accuracy of 0.9587, which is significantly higher than 0.8521 of Jigsaw Attack, with a mean improvement of 0.1067. Statistical tests confirm that the difference is statistically significant (p = 0.0012 $<$ 0.05), with a Cohen's d of 0.6548, indicating a medium-to-large effect size. The 95% CI [0.0458, 0.1675] does not contain zero, which further verifies the significance of the difference. The standard deviation of refine_atk is only 0.0074, much lower than 0.1620 of Jigsaw Attack, demonstrating that the attack results of refine_atk are extremely stable with negligible fluctuation across repeated experiments in this scenario.

The refine_atk achieves an average accuracy of 0.9678, substantially outperforming Jigsaw Attack (0.7261) with a mean improvement of 0.2417 under the uniform distribution on Lucene. The difference is highly statistically significant (p $<$ 0.0001), with a Cohen's d of 1.0842, which corresponds to a large effect size and indicates a pronounced performance gap between the two attacks. The 95% CI [0.1584,0.3249] does not contain zero, validating the statistical significance. Meanwhile, the standard deviation of refine_atk is only 0.0112, while that of Jigsaw Attack reaches 0.2261, highlighting the remarkable robustness advantage of the proposed method.

Jigsaw Attack achieves an average accuracy of 0.7894, which is significantly higher than 0.6370 of refine atk under the Zipfian distribution on Enron, with a mean difference of -0.1524. The difference is highly statistically significant (p $<$ 0.0001), with a Cohen's d of -0.8897, representing a large effect size. The 95% CI [-0.2163, -0.0884] does not contain zero, confirming the significance of the difference. The standard deviation of refine atk is 0.1706, much higher than 0.0593 of Jigsaw Attack, indicating that the attack results of refine atk fluctuate greatly in this scenario and its stability is inferior to the baseline method.

The accuracy of the two methods is nearly comparable under Zipfian distribution on Lucene: with a mean difference of only -0.0049. Statistical tests show that $p = 0.8438$, which is far greater than 0.05, indicating no statistically significant difference between the two methods. The Cohen's d is -0.0363, representing a negligible effect size, which means the performance of the two attacks is essentially equivalent in this scenario. With the exception of the Lucene-Zipfian combination, the remaining groups of comparisons reach statistical significance with effect sizes of medium level or above. This result demonstrates that the observed performance differences are not caused by random errors, but are genuine gaps derived from the inherent characteristics of the algorithms.

IV. Attack Experiment and Performance Analysis

This study will compare the performance of the refine_atk (Ours) attack with the Jigsaw attack [22], RSA attack [14], and IHOP attack [24].

A. Experimental Settings and Attack Parameters

The keyword space is the top $|W|$ keywords in terms of volume ranking. Evaluate the above attacks for different $|W|$ values (500, 1000, 2000). Suppose the attacker knows the frequency of each keyword in the keyword space in the first 50 weeks. By leveraging the frequency of 50 to 100 weeks in Google Trends, generate η queries per week, with τ = 50. η is 100, 500 and 2500.

Briefly introduce the attack parameters required for the above-mentioned attack and the reasons for their values. The

TABLE IV. STATISTICAL RESULTS OF ATTACK ACCURACY COMPARISON[1]

Dataset	Distribution	Refine_atk		Jigsaw Attack		Mean Diff.	p-value	Cohen's d	95% CI	
		Mean	Std	Mean	Std				Lower	Upper
Lucene	zipfian	0.6904	0.1242	0.6953	0.0534	-0.0049	0.8438	-0.0363	-0.0553	0.0455
Lucene	uniform	0.9678	0.0112	0.7261	0.2261	0.2417	< 0.0001	1.0842	0.1584	0.3249
Enron	zipfian	0.6370	0.1706	0.7894	0.0593	-0.1524	< 0.0001	-0.8897	-0.2163	-0.0884
Enron	uniform	0.9587	0.0074	0.8521	0.1620	0.1067	0.0012	0.6548	0.0458	0.1675

[1] All experiments are conducted with a keyword universe size of 1000 and 30 iterations.

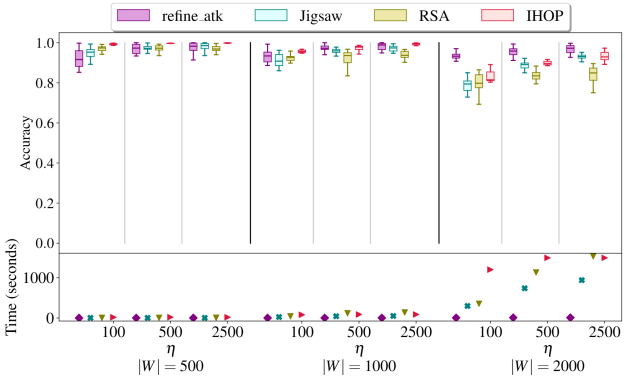

Fig. 28. Comparison of accuracy and time under different attacks in Enron.

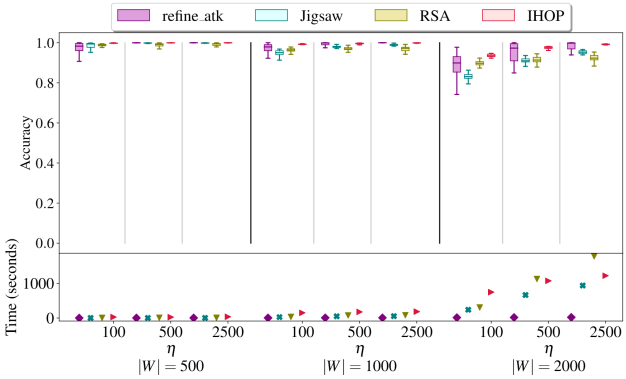

Fig. 29. Comparison of accuracy and time under different attacks in Lucene.

selection of β, *BaseRec* and *ConfRec* significantly affects the accuracy of refine_atk attacks. For β, select a value between 0.2 and 0.8 (Fig. 22). Fig. 6 has shown that without the embedding sub-matrix, the attack accuracy rate drops to 50%. Consider embedding sub-matrices and choose a relatively β, such as 0.5. The minimum *ConfRec* of Jigsaw [22] is 5. If *ConfRec* < 5, Algorithm 2 only correctly recovers < 5 queries, so that Algorithm 3 almost has to recover all user queries. Algorithm 1 and Algorithm 2 do not offer any favorable conditions for the query recovery task, which goes against the original design intention of the entire algorithm. *BaseRec* and *ConfRec* are 45 and 35, respectively, consistent with Jigsaw [22]. The refine_atk is known to have 15 queries. However, during the experiment, these queries are still regarded as "unknown".

For other attack parameters, for Jigsaw attack [22], α is 0.3, β is 0.9. *RefSpeed* controls the number of queries recovered in each iteration. If there are defense measures, increase the *RefSpeed* to reduce the attack time and improve the accuracy. RSA is known to have 10 queries, that is, it randomly selects 10 queries and reveals the underlying plaintext keywords. $|W| \leq 2000$, *RefSpeed* is 10; $|W| > 2000$, *RefSpeed* is 50 [22]. The *RefSpeed* parameter setting strategy for RSA is consistent with that of Jigsaw. The p_{free} of IHOP is 0.25 and that of n_{iters} is 500.

In Fig. 28, the attack accuracy of refine_atk(Ours) is comparable to that of Jigsaw and IHOP. when $|W|$ increases, the attack accuracy rates of Jigsaw, RSA and IHOP decline, but the refine_atk can still remain $> 90\%$. This is because a larger $|W|$ will introduce more low-volume queries that are difficult to recover. The attack time of refine_atk does not fluctuate too much with the changes of $|W|$ and η, and always remains stable. $|W| = 2000$, the attack time of Jigsaw is greater than 500 seconds, and RSA and IHOP are greater than 1000 seconds. This is disadvantageous in some cases because when the attack lasts too long, some databases or systems will record such attack behaviors and issue alerts.

In Fig. 29, when $|W| = 2000$ and $\eta = 100$, the attack accuracy of IHOP is even higher than the refine_atk. The fundamental reason is the extreme sparsity of statistical information. In this case, IHOP's global iterative Hungarian algorithm recovers the global structure from sparse co-occurrence more effectively than refine_atk stagewise greedy strategy, and refine_atk suffers from a severe shortage of HVHF seeds. However, in practical application scenarios, refine_atk outperforms IHOP and other existing attacks by leveraging semantic embeddings, efficient stagewise recovery, and strong robustness. The experimental results in the study confirm this (in Fig. 29, under the vast majority of parameter combinations, refine_atk achieves an accuracy of $\geq 90\%$, and its attack time is far lower than that of IHOP).

B. Refine_atk Ablation Study

1) Known queries ablation experiment: Algorithm 1 recovers unique queries with the aid of known queries. This study now investigates the impact of known queries on the attack performance of Algorithm 1, as shown in Fig. 30. We conducted 30 independent attack simulations on the Enron and Lucene datasets. The keyword space size is $|W|$=1000, *BaseRec*=1000, *ConfRec*= None, the number of observed queries per week $\eta = 5000$, the observation period is 30 weeks, $\tau = 0$, and the user query distribution is "real". In Fig. 30, "enron-kq" denotes the experiment on the Enron dataset with the aid of known queries, while "enron" denotes the experiment without known queries.

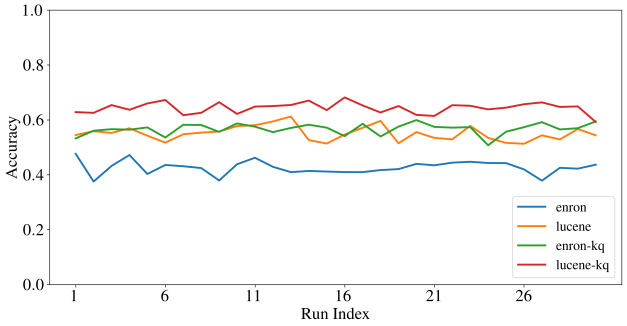

Fig. 30. Ablation study of the known queries component in Algorithm 1.

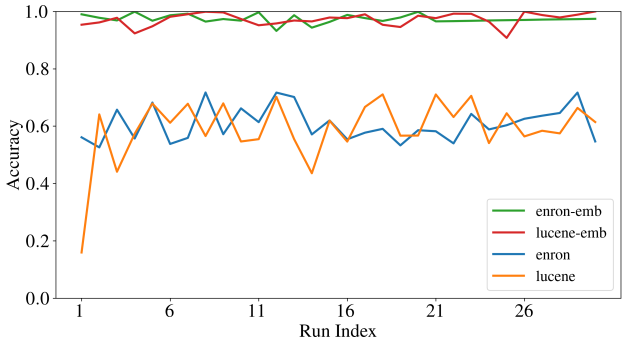

Fig. 31. Ablation study of the semantic embedding component in Algorithm 3.

The accuracy of enron-kq improved by approximately 0.1–0.15 over enron, increasing from 0.4–0.5 to 0.5–0.6, while lucene-kq improved by about 0.1 over lucene, from 0.5–0.6 to 0.6–0.7. It can be observed that the accuracy achieved with known queries is not very high. This is because when *BaseRec*=1000, Algorithm 1 recovers 1000 queries as unique queries, whereas in practice, for a keyword space size of 1000, there are only about 18 unique queries [22].*BaseRec*=1000 far exceeds the actual number of unique queries, leading to the low accuracy of Algorithm 1. Therefore, a lower *BaseRec* value can be set to improve accuracy; please refer to Section III-F for the value of *BaseRec*.

2) Semantic ablation experiment: As shown in Fig. 31, Algorithm 3 in Section III-D recovers the remaining queries at one time with the help of semantic embedding. This section investigates the impact of semantic embedding on the overall attack algorithm. We conducted 30 independent attack experiments on the Enron and Lucene datasets, respectively. The experimental settings are as follows: the keyword space size $|W| = 1000$, $\beta = 0.5$, *BaseRec*=100, *ConfRec*=25, the number of observed queries per week $\eta = 5000$, the number of observed weeks is 30, $\tau = 0$, the number of known queries is 15, and the user query distribution is "real". In Fig. 31, "enron-emb" denotes the experiment with semantic embedding on the Enron dataset, and "enron" denotes the experiment without semantic embedding.

After introducing semantic embedding, the accuracy of "enron-emb" and "lucene-emb" is almost stable between 0.95 and 1.0, while the attack accuracy without semantic embedding is only between 0.4 and 0.7, which is much lower than that

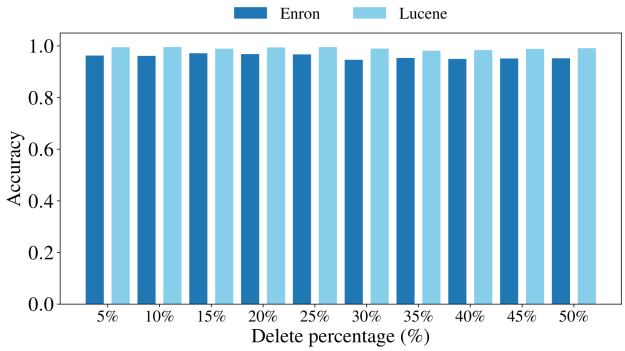

Fig. 32. Comparison of accuracy and delete percentage in Enron and Lucene.

of the attack with semantic embedding. Semantic embedding is a key factor to improve attack accuracy, as it provides additional semantic information for query recovery. The attack performances of the Enron and Lucene datasets show almost no difference, indicating that the attack method with semantic embedding has good generalization to different datasets. The initial fluctuation of Lucene is obvious, which is related to the keyword distribution of the dataset.

C. Delete Similar Data

An experiment on the impact of deleting some similar data on attack accuracy and the recovery of the top 10 high-frequency queries under the partial deletion ratio of Algorithm 1. Deleting partially similar data refers to the assumption that the size of similar data documents is 50 and the deletion ratio is 50%, that is, 50% of the similar data documents are retained, which is 25 documents. The attacker only used these 25 documents to generate the keyword-document matrix, while the real data documents always maintained the size after division. Under the partial deletion ratio, record the number of occurrences of each query in all pellations in the recovery results of Algorithm 1. Each query is sorted in descending order of its occurrence frequency as a list of high-frequency queries. Filter out the pairings that "the query is not on the high-frequency list" or "the predicted keyword is not on the high-frequency list", and retain the top 10 valid pairings where both parties belong to the high-frequency list. The deletion ratio is 5%, ..., 50%. Thirty experiments were independently simulated under each dataset and deletion ratio, and the accuracy of each experiment was collected. Each round of the simulated attack is conducted independently, and there is no mutual interference between different test runs.

When the deletion ratio is 5%, 10%, 15%, 20% and 25%, the attack accuracy rate is slightly higher than that of other ratios. This rule is reflected in both datasets [Fig. 32]. The accuracy rate of the deletion ratio of 5%, 10%, 15%, 20% and 25% in Enron was 96%. The accuracy rate of deletion ratios of 30%, 35%, 40%, 45% and 50% in Lucene is 97%. Regardless of the deletion ratio, the attack accuracy rate is all $>$ 95%. The number of auxiliary documents used by the attacker to recover the query is only half of the user's real documents. Under such unfavorable circumstances, the refine_atk attack can still achieve an accuracy rate of $>$ 90%. In practical scenarios, attackers typically have only limited

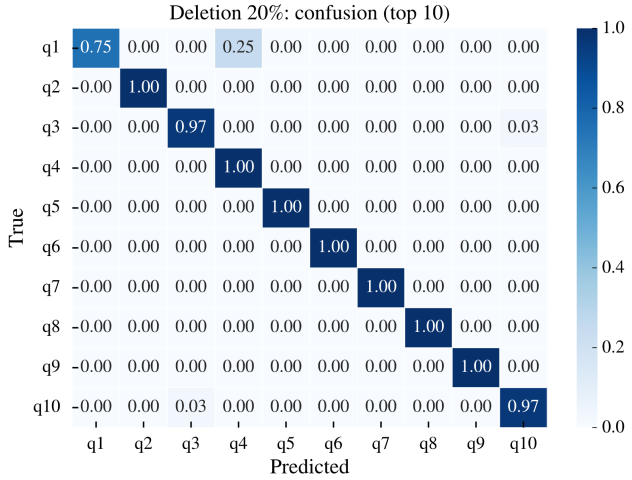

Fig. 33. Recovery results of the top 10 high-frequency queries by Algorithm 1 under 20% ratio. The vertical axis represents the labels of real queries.The horizontal axis represents the labels of the corresponding predicted keywords. (q1, q3) indicates that the real label q3 is predicted as q1.

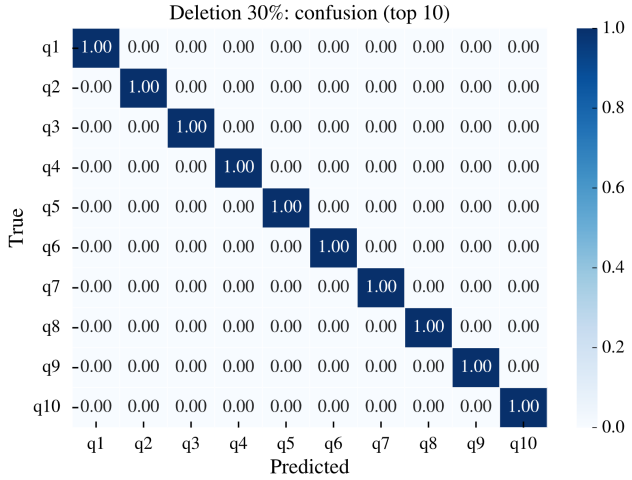

Fig. 34. Recovery results of the top 10 high-frequency queries by Algorithm 1 under ratio 30%.

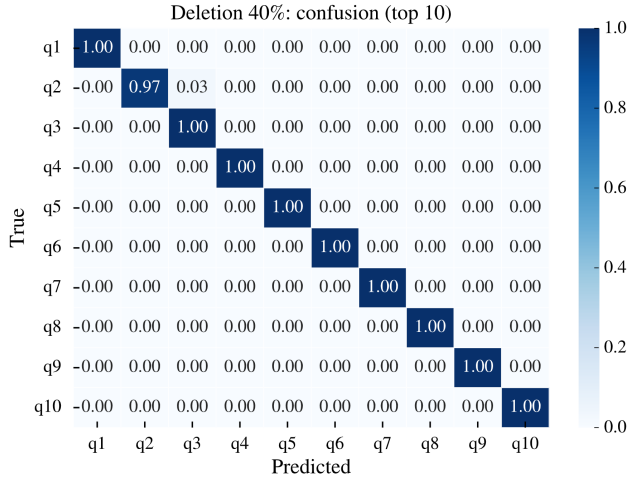

Fig. 35. Recovery results of the top 10 high-frequency queries by Algorithm 1 under ratio 40%.

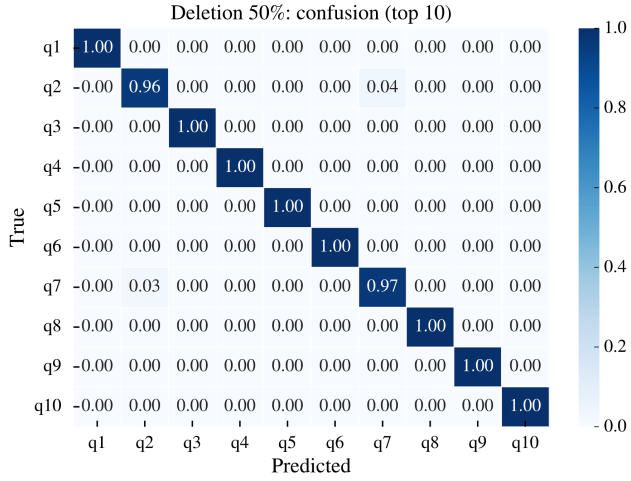

Fig. 36. Recovery results of the top 10 high-frequency queries by Algorithm 1 under ratio 50%.

auxiliary knowledge. Even under this restrictive condition, the proposed attack can still successfully recover users' underlying keywords, which alerts users that their query behaviors entail potential privacy leakage risks.

This shows the recovery of the top 10 high-frequency queries under the partial deletion ratio of Algorithm 1. Fig. 33 shows that the probability of correct recovery for query q1 is relatively low, at 0.75. Recover to q4 with a probability of 0.25. q3 and q10 were correctly recovered with a probability of 0.97. Fig. 34 shows that the probability of correct recovery for query q1-q10 is relatively high, nearly 1. Fig. 35 shows that the probability of correct recovery for query q2 is 0.97. Fig. 36 shows that queries q2 and q7 were correctly recovered with probabilities of 0.96 and 0.97 respectively. This proves the experimental conclusion in Fig. 32, that is, when the attacker auxiliary knowledge is relatively limited, the attack proposed in this study can still correctly recover high-frequency queries with a probability of nearly 1. This suffices to demonstrate the robustness of Algorithm 1 and its sensitivity to HVHF queries. Algorithm 1 exhibits sound generalization performance on both datasets.

V. Conclusion

The proposed attack, refine_atk, performs excellently in most cases. The Algorithm 3 encodes unrecovered queries and unpaired keywords with the help of a pre-trained language model. The refine_atk is tested on different datasets, and the attack accuracy is around 95%. The attack time is much lower than other attacks. Compared with Jigsaw [22], under different datasets and parameters of *BaseRec* and *ConfRec*, the refine_atk achieves higher attack accuracy, more correctly recovered queries, and a recovery rate comparable to that in [22].Algorithm 1, in this study can significantly improve the attack accuracy in Top-10, and can even utilize the weights of query learning capacity and frequency such as LVLF (HVLF, LVHF). The attack performance remains robust when auxiliary

knowledge is limited. In defense evaluation, more advanced defense mechanisms such as semantic obfuscation or synonym expansion should be considered.

The query recovery attack can be extended to dynamic searchable symmetric encryption (DSSE) scenarios. Most existing query recovery attacks are modeled on static document collections, whereas dynamic update operations, such as document addition, deletion and modification, are ubiquitous in practical cloud storage environments and introduce additional novel leakage sources, including update patterns and temporal access correlations. Future work can leverage the leakage characteristics inherent in the dynamic update process to evaluate the query privacy leakage risks of state-of-the-art DSSE schemes in dynamic scenarios. Therefore, it is crucial to develop a secure and practical SSE scheme that can resist semantic-level attacks to protect user data security.

ACKNOWLEDGMENT

This study was supported by the Key Project of the University-Level Scientific Research Innovation Project of Gansu University of Political Science and Law (Grant No. GZF2025XZD27). The authors would like to express their sincere gratitude to the funding organization for its support of this research.

REFERENCES

[1] Y. Zhang, J. Katz, C. Papamanthou (2016) All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. Paper presented at the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, Aug 2016, 707–720. https://doi.org/10.1109/EuroSP48549.2020.00030.

[2] R. Poddar, S. Wang, J. Lu, R. A. Popa (2020) Practical Volume-Based Attacks on Encrypted Databases. Paper presented at the IEEE Symp Security Privacy (S&P), 2020, 354–369. https://doi.org/10.1109/EuroSP48549.2020.00030.

[3] X. Zhang, W. Wang, P. Xu, L. T. Yang, K. Liang (2023) High Recovery with Fewer Injections: Practical Binary Volumetric Injection Attacks against Dynamic Searchable Encryption. Paper presented at the 32nd USENIX Security Symposium (USENIX Security 23), 2023, 5953–5970.

[4] L. Zhang, J. Wang, J. Wu, Y. Wang, S. Sun (2025) Violin: Powerful Volumetric Injection Attack Against Searchable Encryption with Optimal Injection Size. IEEE Trans Industr Inf 21 (9):4103–4115. https://doi.org/10.1109/TDSC.2025.3543248.

[5] D. Cash, P. Grubbs, J. Perry, T. Ristenpart (2015) Leakage-Abuse Attacks Against Searchable Encryption. Paper presented at the ACM SIGSAC Conf Comput Commun Security (CCS), 2015, 668–679. https://doi.org/10.1145/2810103.2813700.

[6] M. S. Islam, M. Kuzu, M. Kantarcioglu (2012) Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. Paper presented at the Netw Distrib Syst Security Symp (NDSS), 2012.

[7] L. Blackstone, S. Kamara, T. Moataz (2020) Revisiting Leakage Abuse Attacks. Paper presented at the Netw Distrib Syst Security Symp (NDSS), 2020.

[8] J. Ning, X. Huang, G. S. Poh, J. Yuan, Y. Li, J. Weng, R. H. Deng (2021) LEAP: Leakage-Abuse Attack on Efficiently Deployable, Efficiently Searchable Encryption with Partially Known Dataset. Paper presented at the ACM SIGSAC Conf Comput Commun Security (CCS), 2021, 2307–2320. https://doi.org/10.1145/3460120.3484540.

[9] Y. Zhou, Y. Ren, M. Yi, Y. Xiao, Z. S. Tan, N. Moustafa, Z. Tian (2023) CDTier: A Chinese Dataset of Threat Intelligence Entity Relationships. IEEE Trans Sustain Comput 8 (4):627–638. https://doi.org/10.1109/TSUSC.2023.3240411.

[10] Y. Ren, Y. Xiao, Y. Zhou, Z. Zhang, Z. Tian (2023) CSKG4APT: A Cybersecurity Knowledge Graph for Advanced Persistent Threat Organization Attribution. IEEE Trans Knowl Data Eng 35 (6):5695–5709. https://doi.org/10.1109/TKDE.2022.3175719.

[11] C. Liu, L. Zhu, M. Wang, Y. Tan (2014) Search Pattern Leakage in Searchable Encryption: Attacks and New Construction. Inf Sci 265:176–188. https://doi.org/10.1016/j.ins.2013.11.021.

[12] S. Oya, F. Kerschbaum (2021) Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption. Paper presented at the 30th USENIX Security Symposium (USENIX Security 21), 2021, 127–142.

[13] M. L. Fredman, R. E. Tarjan (1987) Fibonacci heaps and their uses in improved network optimization algorithms. J ACM 34 (3):596–615. https://doi.org/10.1145/28869.28874.

[14] M. Damie, F. Hahn, A. Peter (2021) A Highly Accurate Query-Recovery Attack against Searchable Encryption using Non-Indexed Documents. Paper presented at the 30th USENIX Security Symposium (USENIX Security 21), 2021, 143–160.

[15] Z. Shang, S. Oya, A. Peter, F. Kerschbaum (2021) Obfuscated Access and Search Patterns in Searchable Encryption. Paper presented at the 28th Netw Distrib Syst Security Symp (NDSS), 2021. https://doi.org/10.14722/ndss.2021.23041.

[16] L. Xu, L. Zheng, C. Xu, X. Yuan, C. Wang (2023) Leakage-Abuse Attacks Against Forward and Backward Private Searchable Symmetric Encryption. Paper presented at the ACM SIGSAC Conf Comput Commun Security (CCS), 2023, 3003–3017. https://doi.org/10.1145/3576915.3623085.

[17] Z. Gui, K. G. Paterson, S. Patranabis (2023) Rethinking Searchable Symmetric Encryption. Paper presented at the IEEE Symp Security Privacy (S&P), 2023, 1401–1418. https://doi.org/10.1109/SP46215.2023.10179460.

[18] L. Wang, L. Xu, Y. Chen, Y. Zou, C. Wang (2025) ALERT: machine learning-enhanced risk estimation for databases supporting encrypted queries. Paper presented at the 34th USENIX Security Symposium (USENIX Security 25), 2025.

[19] Y. Chai, J. Qiu, L. Yin, L. Zhang, B. B. Gupta, Z. Tian (2022) From Data and Model Levels: Improve the Performance of Few-Shot Malware Classification. IEEE Trans Netw Service Manage 19 (4):4248–4261. https://doi.org/10.1109/TNSM.2022.3200866.

[20] Y. Chai, L. Du, J. Qiu, L. Yin, Z. Tian (2023) Dynamic Prototype Network Based on Sample Adaptation for Few-Shot Malware Detection. IEEE Trans Knowl Data Eng 35 (5):4754–4766. https://doi.org/10.1109/TKDE.2022.3142820.

[21] Z. Wang, Y. Zhou, H. Liu, J. Qiu, B. Fang, Z. Tian (2024) ThreatInsight: Innovating Early Threat Detection Through Threat-Intelligence-Driven Analysis and Attribution. IEEE Trans Knowl Data Eng 36 (12):9388–9402. https://doi.org/10.1109/TKDE.2024.3474792.

[22] H. Nie, W. Wang, P. Xu, X. Zhang, L. T. Yang, K. Liang (2024) Query Recovery from Easy to Hard: Jigsaw Attack against SSE. Paper presented at the 33rd USENIX Security Symposium (USENIX Security 24), 2024, 2599–2616.

[23] G. K. Zipf (1948) Human Behavior and the Principle of Least Effort.

[24] S. Oya, F. Kerschbaum (2022) IHOP: Improved Statistical Query Recovery against Searchable Symmetric Encryption through Quadratic Optimization. Paper presented at the 31st USENIX Security Symposium (USENIX Security 22), 2022, 2407–2424.

[25] R. Bost (2016) $\sum o\phi o\varsigma$: Forward Secure Searchable Encryption. Paper presented at the ACM SIGSAC Conf Comput Commun Security (CCS), 2016, 1143–1154. https://doi.org/10.1145/2976749.2978303.

[26] R. Bost, B. Minaud, O. Ohrimenko (2017) Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. Paper presented at the ACM SIGSAC Conf Comput Commun Security (CCS), 2017, 1465–1482. https://doi.org/10.1145/3133956.3133980.

[27] D. Cash, S. Tessaro (2014) The Locality of Searchable Symmetric Encryption. Paper presented at the Advances Cryptology (EUROCRYPT), 2014, 351–368. https://doi.org/10.1007/978-3-642-55220-5_20.

[28] J. G. Chamani, D. Papadopoulos, C. Papamanthou, R. Jalili (2018) New Constructions for Forward and Backward Private Symmetric Searchable Encryption. Paper presented at the ACM SIGSAC Conf Comput Commun Security (CCS), 2018, 1038–1055. https://doi.org/10.1145/3243734.3243833.

[29] S. Kamara, T. Moataz (2017) Boolean Searchable Symmetric Encryption with Worst-Case Sub-linear Complexity. Paper presented at the Advances Cryptology (EUROCRYPT), 2017, 94–124. https://doi.org/10.1007/978-3-319-56617-7_4.

[30] S. Patel, G. Persiano, K. Yeo (2018) Symmetric Searchable Encryption with Sharing and Unsharing. Comput Security 74:207–227. https://doi.org/10.1007/978-3-319-98989-1_11.

[31] S. Sun, X. Yuan, J. K. Liu, R. Steinfeld, A. Sakzad, V. Vo, S. Nepal (2018) Practical Backward-Secure Searchable Encryption from Symmetric Puncturable Encryption. Paper presented at the ACM SIGSAC Conf Comput Commun Security (CCS), 2018, 763–780. https://doi.org/10.1145/3243734.3243782.

[32] P. Xu, W. Susilo, W. Wang, T. Chen, Q. Wu, K. Liang, H. Jin (2022) ROSE: Robust Searchable Encryption With Forward and Backward Security. IEEE Trans Inf Forensics Security 17:1115–1130. https://doi.org/10.1109/TIFS.2022.3155977.

[33] W. W. Cohen, MLD (2015) Enron email datasets. https://www.cs.cmu.edu/ ./enron/.

[34] Apache Foundation (1999) Mail archives of lucene. https://mail-archives.apache.org/modmbox/#lucene.

[35] G. Chen, T. Lai, M. K. Reiter, Y. Zhang (2018) Differentially Private Access Patterns for Searchable Symmetric Encryption. Paper presented at the IEEE Conf Comput Commun (INFOCOM), 2018, 810–818. https://doi.org/10.1109/INFOCOM.2018.8486381.

[36] Y. Liu, J. Hu, Z. Chen, X. Wan, T.-H. Chang (2023) EASAL: Entity-Aware Subsequence-Based Active Learning for Named Entity Recognition. Paper presented at the AAAI Conf Artif Intell (AAAI), 2023, 1000–1008. https://doi.org/10.1609/aaai.v37i7.26069.

[37] Google (2004) Google trends.