Information Retrieval Journal (2023) 26:12
https://doi.org/10.1007/s10791-023-09428-z

Privacy-aware document retrieval with two-level inverted indexing

Yifan Qiao[1] · Shiyu Ji[1] · Changhai Wang[1] · Jinjin Shao[1] · Tao Yang[1]

Received: 27 March 2023 / Accepted: 20 October 2023 / Published online: 17 November 2023

Abstract
Previous work on privacy-aware ranking has addressed the minimization of information leakage when scoring top k documents, and has not studied on how to retrieve these top documents and their features for ranking. This paper proposes a privacy-aware document retrieval scheme with a two-level inverted index structure. In this scheme, posting records are grouped with bucket tags and runtime query processing produces query-specific tags in order to gather encoded features of matched documents with a privacy protection during index traversal. To thwart leakage-abuse attacks, our design minimizes the chance that a server processes unauthorized queries or identifies document sharing across posting lists through index inspection or across-query association. This paper presents the evaluation and analytic results of the proposed scheme to demonstrate the tradeoffs in its design considerations for privacy, efficiency, and relevance.

Keywords Document search with ranking · Privacy protection · Retrieval efficiency

1 Introduction

As sensitive information is increasingly stored on the cloud, privacy concerns on unauthorized data access or inferences have been a critical factor for individuals or corporations to adopt cloud-based information retrieval services. A dilemma considered is that a user wants to take full advantage of the cloud resource to search a large hosted dataset quickly with an index stored together on the cloud, but this user may not fully trust this cloud on

✉ Yifan Qiao
yifanqiao@cs.ucsb.edu

✉ Shiyu Ji
shiyuji@google.com

✉ Tao Yang
tyang@cs.ucsb.edu

Changhai Wang
changhai_wang@cs.ucsb.edu

Jinjin Shao
jinjinshao@google.com

[1] Department of Computer Science, University of California, Santa Barbara, CA 93106, USA

data privacy. This is because a cloud server may be honest-but-curious and can observe the client-initiated query processing flow, and reason about client's data. Consequently, user data leakage may occur accidentally if not intentionally. The previous work in searchable encryption (Song et al., 2000; Curtmola et al., 2006; Kamara et al., 2012; Cash et al., 2013, 2014; Kamara & Moataz, 2017; Lai et al., 2018) has been proposed to identify documents that match a user query from the encrypted index with minimized information leakage. None of these papers has considered ranking of the matched results. Privacy issues have also attracted research interests in the information retrieval community and SIGIR held several privacy workshops on this subject (Yang & Soboroff, 2015; Yang et al., 2016).

This paper studies privacy-enhanced document matching and feature access to facilitate top-*k* search with ranking over an encrypted index. Privacy-aware ranking solutions (Ji et al., 2018; Shao et al., 2019) with tree ensembles and neural networks have studied how to avoid the leakage of privacy-sensitive feature information during ranking computation with ranking feature encoding and obfuscation, but they do not address how to retrieve top matched documents and gather their features safely for ranking. Thus this paper studies this open problem. The work in Agun et al. (2018) studies additive ranking with retrieval, but a server needs to send a large amount of data back to a client to complete ranking, and the work in Xu et al. (2021) furthers pushes for client-side search by securely fetching index data from a server. These studies do not support full server-side ranking.

Document retrieval for search often uses an inverted index in which each searchable term is associated with a list of posting records containing document ID and its term-based features. For privacy protection, document IDs and their ranking features of an index are encoded or encrypted when hosted on a cloud server. The challenge in designing such a search index is that a server adversary can inspect the index, observe the query processing flow, and learn some sensitive patterns, which may recover document plaintext even partially. Thus a document retrieval solution needs to minimize the chance that a server processes unauthorized queries or learn sensitive information from query processing. The previous studies on privacy-abuse attacks (Islam et al., 2012; Cash et al., 2015; Wang et al., 2018; Pouliot & Wright, 2016) show the leakage of the information about term co-occurrence in a document can yield a plaintext attack. For example, if an adversary knows word "new" appears in a document and can estimate its co-occurrence probability with another word x in the same index. Then this adversary may be able to recover x as "york" appeared in this document by comparing against some known knowledge on the co-occurrence probabilities of pair "new" and "york", and other word pairs. Notice that the co-occurrence probability of two terms is computed as being proportional to the number of documents shared between the posting lists of these two terms. Thus avoiding leakage of document sharing among posting lists is our key consideration in privacy protection.

When designing an index for privacy-aware search that retrieves and ranks documents with their features, one challenge faced is how to associate term-specific features distributed in different online data structures with a document that owns these features, without leaking document sharing patterns among posting lists. If such a document ID is encrypted with a fixed static value, this value does not change from one query to another. Then a server adversary can observe the document sharing pattern among two posting lists based on static document IDs, and can estimate the co-occurrence probability of these two terms appearing in the same document in this index. Following the discussion in the previous paragraph, the risk of a privacy attack or leakage that successfully recovers some plaintext exists. On the other hand, if the encrypted value of the same document ID is non-deterministic in posting lists of different terms, the server is unable to recognize term-based features belonging to the same matched document and conduct effective ranking. The existing searchable encryption studies use non-deterministic document IDs and cannot be easily extended for a search system that gathers ranking features and computes preliminary scores during document retrieval. Another challenge faced is that cryptographic computing operators used in searchable encryption techniques involve very long bits of integers, which is expensive and becomes a performance bottleneck for a reasonable query response time even for a modestly large dataset.

The contribution of this paper is to address this open problem of privacy-aware document retrieval with ranking by introducing a two-level inverted index and query-specific tagging called QDT. QDT gathers encoded ranking features of matched documents with privacy-protection to facilitate document ranking. During index traversal, QDT generates query-specific tags to correctly recognize the term-specific ranking features of matched documents and uses a two-level index structure to strike a trade-off of privacy and efficiency for ensuring a reasonable query response time. The design of QDT minimizes the chance that a server conducts unauthorized search with ranking or learns document sharing patterns of posting lists when processing a sequence of client-authorized queries. Our evaluation indicates that adding privacy protection for document retrieval does incur significant cost compared to the well-known retrieval baselines without privacy constraints, and the result demonstrates the importance of our two-level design in optimizing the use of cryptographic computation under privacy requirement. Another contribution of this paper is to show that a leakage-abuse attack is still possible in a naive version of the above design, and the guided fake document padding in QDT thwarts such an attack. Our evaluation validates its compatibility with ranking and its effectiveness in attack prevention.

The rest of this paper is organized as follows. Section 2 provides the background information of the problem focused in this paper and the related work. Section 3 discusses the design considerations and the proposed two-level index data structure, document-specific tagging, and guided fake document padding. Section 4 provides an evaluation of the proposed QDT scheme in terms of time efficiency, and its compatibility with ranking and its effectiveness in attack prevention. Appendix 1 discusses a design consideration of QDT in preventing an attack based on document-pair ID mapping. Appendix 2 provides several privacy properties of QDT. Appendix 3 lists the leakage profile of QDT following an analytic framework of the previous work.

2 Background and related work

Search systems for text documents often employ multi-stage ranking in practice (e.g. (Matveeva et al., 2006; Wang et al., 2011)). The first retrieval stage extracts top candidate documents matching a query from a large search index with a fast and simple ranking method. The second stage or a later stage uses a more complex machine learning algorithm to re-rank top results thoroughly. Following the disjunctive query semantics widely used in the previous work (e.g. (Broder et al., 2003; Ding & Suel, 2011)), this paper is focused on the problem of document retrieval that identifies documents matching at least one search term, and aggregates their term features for ranking in privacy-aware cloud search. This extra privacy protection constraint requires the minimization of the information leakage to a cloud server and the threat model is discussed below. This paper assumes that document search returns a list of document IDs ranked on the top for a client to make further data fetching.

Traditional retrieval algorithms in IR often use an inverted index (Baeza-Yates & Ribeiro-Neto, 2011) which contains a set of terms and each term points to a list of document IDs that possess a feature for such a term. This list is called a *posting* list. Each record in this list containing a document ID and its term-specific feature is called a *posting* record.

Privacy and threat model There are three entities in a cloud system: data set owner, a search user, and a cloud server. We assume a client owns a collection of private documents and outsources the encrypted data to a cloud server. The client builds an encrypted searchable index, lets a server host such index, and *only this client or the other users authorized by this client can search the hosted data*. We assume that a client can periodically overwrite the index on the cloud to include new content. Dynamic index update (Kamara et al., 2012) is not considered in this paper. The server is allowed to access this hosted index for client query processing only, and it is honest-but-curious, i.e., the server will honestly follow the client's protocol, but will also try to learn some information based on what can be observed from the index and during query processing. Multi-round communication between the server and client (e.g. (Naveed et al., 2014; Hu et al., 2011; Lai et al., 2018)), client–server collaborative ranking (Agun et al., 2018), and securely fetching index data from a server to a client for client-side search (Xu et al., 2021) are not considered in this paper because they incur high client–server communication cost for large datasets or low-bandwidth platforms, and they represent an orthogonal approach.

For privacy protection, document IDs and index information in a search system are encrypted or encoded. The biggest threat to search privacy is the leakage of document/term-related statistical information induced from an index and access patterns within a query or across queries. From the above information, a server can launch a privacy attack and reveal document content partially (Islam et al., 2012; Cash et al., 2015; Wang et al., 2018). Specifically, Islam et al. (2012) proposed a query recovery attack to identify the plaintext of some words, assuming that a server adversary knows about a subset of plaintext of some words, and the co-occurrence probability of two words in a document. Through intensive guessing and validation computation, an adversary attack may estimate the co-occurrence probability of words in a hosted index, derive a mapping from word IDs to English words that closely matches the known co-occurrence probability, which reveals the plaintext of some words as a privacy breach. Cash et al. (2015) improved the above work by exploiting extra information such as the length of the posting list of each searchable term. There are also attacks exploiting leaked document similarities (Wang et al., 2018).

The above attacks only work if the adversary knows co-occurrence probability of targeted keywords in the document set, which is proportional to the count of shared documents among the posting lists of these keywords. By preventing the leakage of the document sharing information of documents among posting lists in a hosted index, threats from these attacks can be greatly alleviated.

Blackstone et al. (2019) introduces attacks based on volume information (document length). This paper assumes the server only needs to return the encrypted document IDs of top results but not the text of each document, and a client may find document text from another place later with these IDs. Thus, these volume attacks would not be applicable in our case.

Document retrieval with privacy-aware ranking The previous work on document retrieval (e.g. (Broder et al., 2003; Ding & Suel, 2011)) often selects top k results based on a simple additive formula as the first stage of search and it computes the rank score of each document d as: $RankScore(d) = \sum_{t \in Q} w_t$ where Q is the set of all search terms and w_t is a term-specific weight of this document. An example of such additive formula is BM25 (Jones et al., 2000) which is widely used. This paper assumes the disjunctive query semantics, motivated by the above formula which implies any document that contains a query term accumulates some score and is a candidate for the top results. Notice that relevant text documents may not contain any query term in many applications, and recent advancement in document term expansion through a deep neural model addresses document-query vocabulary mismatch (Formal et al., 2021; Mallia et al., 2021; Lin & Ma, 2021). Thus the use of disjunctive semantics can still be applied widely by coupling with such techniques.

To conduct additive scoring like BM25, a server needs to decrypt the encrypted features before computation, that results in the leakage of sensitive ranking features. Homomorphic encryption (Gentry, 2009; Paillier, 1999) can let the server perform arithmetic calculations without decrypting the underlying data. But such a scheme does not have the ability of comparing two results using scores computed with homomorphic encryption at the server side, and it is also very time consuming and computationally non-scalable when many numbers are involved in ranking computation. Dense matrix multiplication is used in Cao et al. (2014), Sun et al. (2014), Xia et al. (2016) for additive ranking, and its complexity is proportional to the number of documents multiplied by the number of distinct words, which is not scalable for a large data collection. The evaluation of this paper adopts a simple obfuscation-based approach to improve privacy protection by mapping each BM25 term feature into a set of non-uniform partitions to hide privacy-sensitive true values. This follows the value partitioning technique (Hacigümüş et al., 2002) as extreme downsampling (Ryoo et al., 2017) to mask privacy-sensitive feature details with obfuscation.

Our scheme can also be used for more complex ranking. For example, one may opt to directly adopt a tree ensemble or neural ranking method without going through the first-stage BM25 for a relatively small or medium-sized document collection for which such a method can be efficient enough. Privacy-aware tree ensemble ranking with feature encoding is studied by Ji et al. (2018) and neural ranking with feature obfuscation is proposed by Shao et al. (2019) based on KNRM and ConvKNRM ranking (Xiong et al., 2017; Dai et al., 2018). None of the above work addresses how to safely gather ranking features during document retrieval. The techniques developed in this paper to gather ranking features in a privacy-aware manner can be applicable when such a ranking method is used. The transformer-based ranking models with contextual embeddings outperform KNRM and ConvKNRM on relevance (e.g. (Lin et al., 2020; MacAvaney et al., 2020; Khattab & Zaharia, 2020; Yang et al., 2022; Li et al., 2023)). This paper does not use transformer-based models in the evaluation because there are no studies on how to address privacy issues for such complex ranking models. Since the work in Shao et al. (2019) provides a privacy-aware neural re-ranking even it is based on static document embeddings, this paper leverages Shao et al. (2019) to demonstrate safe top result retrieval and feature aggregation, and conducts an end-to-end neural ranking under privacy constraints. Recent optimization studies on learned sparse representation of documents using BERT have shown strong relevance and efficiency results (Formal et al., 2022; Qiao et al., 2023; Thakur et al., 2023), and these techniques are orthogonal optimization that our work can leverage in the future.

Searchable encryption, query authorization, and document identification The previous work on searchable encryption has not addressed ranking issues including feature gathering. However the searchable encryption techniques can be leveraged to help solving our problem with the disjunctive query search semantics.

Although OXT (Cash et al., 2013, 2014) does not support disjunctive query semantics, it provides a searchable encryption mechanism that a server can extract relevant information from an index and process a query only with an authorization from a client. A client sends a start-up term token and additional intersection tokens acting as a mean of authorizing conjunctive search on the encoded index hosted on a server. Lai et al. developed the HXT scheme (Lai et al., 2018) that improves the security of OXT using hidden vector encryption and a Bloom filter with extra communication overhead in multi-round client–server communication. The work by Agun et al. (2018) extends OXT for additive ranking but it only supports partial server-side ranking and large client–server communication is needed, and thus it does not scale well for a large dataset. All of the above works require one of terms to lead the flow of conjunction handling. and cannot be extended easily and efficiently to support disjunctive semantics.

In a traditional inverted index, it is natural and straightforward to store a feature in a posting record of a term, and use a document ID to show the ownership of this feature. The above natural way of associating a term feature with a deterministic and static document ID would not work for the privacy-sensitive index. That is because a server can easily count that the number of common static document IDs shared among the posting lists of two terms, to compute the co-occurrence probability of targeted words, defined as the number of documents that appear in two corresponding posting lists divided by the number of documents. The above action can lead to the plaintext attack as mentioned above.

The IEX scheme introduced by Kamara and Moataz (2017) supports disjunctive queries, where the same document ID that appears in different posting records has a different encrypted value with non-deterministic encryption. Thus document sharing patterns among posting lists are not leaked explicitly. Because of the above non-deterministic encryption, the server cannot figure out features of the same document appearing in different posting records and thus it is difficult to extend IEX to support ranking feature gathering.

Static index vs. dynamic update Recent studies on searchable encryption for conjunctive queries have proposed ODXT (Patranabis & Mukhopadhyay, 2021) and ESP-CKS (Xu et al., 2023) for the dynamic update of a database and its index, which is the main advancement compared to OXT and HXT, but server side ranking is not addressed. Our work on the other hand, does not support dynamic document addition and deletion to an existing search index, but focuses on ranking with a static index. We assume the search index in a hosted server can be refreshed periodically with a new or updated index. Thus the above dynamic index update work is orthogonal to this paper.

Fake document padding, obfuscation, and differential privacy The leakage of the posting list length can further aid text-revealing privacy attacks together with term co-occurrence leakage as shown in Islam et al. (2012), Cash et al. (2015), Pouliot and Wright (2016). To avoid this, padding fake documents in a posting list is proposed in Islam et al. (2012), Patel et al. (2019), Kamara and Moataz (2019) for searchable encryption. A bucket-based padding technique is proposed in Kamara and Moataz (2019), but it is not extensible for ranking because it cannot identify the same documents referenced in different posting records. Also to store ranking features in this scheme, excessive feature duplication is needed and the space cost is very expensive.

Padding with fake documents in the index can be viewed as an obfuscation strategy. Obfuscating search queries has been studied to achieve user anonymity (Ahmad et al., 2018, 2016). There is a line of work with data perturbation or obfuscation for differential privacy in classification (e.g. (Jagannathan et al., 2009; Liu et al., 2017)) and in neural model or representation learning (Chase et al., 2017; Habernal, 2022). Chen et al. (2018) introduced a differentially private obfuscation framework to mitigate access pattern leakage in searchable encryption and it intentionally includes false positives and false negatives in the index while adding redundancy by encoding a document into multiple shards based on erasure coding. Shang et al. (2021) proposes obfuscated searchable encryption to improves Chen et al.'s scheme by a fresh obfuscation per query with a tradeoff at a larger search cost. Its query processing has a time complexity of $O(\frac{n \log n}{\log \log n})$ where n is the number of documents in the index and that is not scalable for a large dataset. Both of these two papers do not address document ranking, and they deal with single-word queries only while this paper addresses document retrieval in a more complex disjunctive multi-keyword query setting.

ORAM and hardware enclaves Garg et al. (2015) proposed search encryption based on oblivious RAM (ORAM) to avoid the access-pattern leakage at a cost of high search overhead for a large dataset. Document retrieval leveraging hardware technologies such as Intel SGX and/or ORAM is studied in Sun et al. (2018), Mishra et al. (2018), Hoang et al. (2019), Shao et al. (2020), Vo et al. (2021). ORAM techniques are extremely expensive without such hardware (Mishra et al., 2018). On the other hand, the risk of privacy-sensitive attacks exists on such a platform (Costan & Devadas, 2016; Brasser et al., 2017; Xu et al., 2015) and a client may not fully trust such hardware owned by servers. This paper does not use such a hardware.

3 Indexing and query processing

To allow a server to gather meaningful feature information from the hosted index in a privacy-aware manner with client authorization, we propose a retrieval scheme with *query-specific document tagging* and will call it *QDT*. QDT leverages a cryptographic technique for information blinding used in OXT (Cash et al., 2013, 2014), but revises it for disjunctive query processing. The long-bit arithmetic is required for the involved cryptographic operations, which is expensive and significantly slows down query processing time. With this in mind, QDT adopts a two-level index structure using a two-dimensional representation of document IDs and derives a query-specific document tag to identify a document that appears in multiple accessed posting lists. This two-level structure helps QDT to control the time complexity of runtime tag computation. We also further develop guided padding of fake documents to enhance hiding of statistical term co-occurrence information in the index.

Like OXT, the formula of QDT employs pseudo-random functions (PRF), denoted as $H_i(x)$, which are either AES-256 (Dworkin, 2001) or a cryptographic hash function such as SHA-3 (Dworkin, 2015). They use the secret key called k_i, only known by the client. A PRF function is applied to search terms, document IDs and other index information for privacy protection. The formula in the rest of this section involves modular arithmetic of integers and will use a Diffie-Hellman Group (Boneh & Shoup, 2015) of size q defined as the integer set $\{1, g, g^2 \mod Q, \cdots, g^{q-1} \mod Q\}$, where g is called a generator, and both q and Q are large primes satisfying $g^q \mod Q \equiv 1$ and $q < Q$. The security assumption of the Diffie-Hellman group is based on the fact that given $g^x \mod Q$, it is computationally hard to infer any useful information of x, even g, q, and Q are public. More details on this subject and typical parameter values can be found in Section 10.4.2 of an online book (Boneh & Shoup, 2023) and in Appendix 4 of an NIST standard (Barker et al., 2018) and related RFCs (Kojo & Kivinen, 2003; Gillmor, 2016). Note that while q and Q can be typically in 2024 bits or more, a Diffie-Hellman group can also be built and represented using Elliptic Curve arithmetic with a pair of two numbers in a relatively smaller number of bits such as 224 bits (Boneh & Shparlinski, 2001) to speed up calculation.

Table 1 lists frequently used notations through this paper, and it does not include some symbols that are used locally.

Table 1 Frequently used notations

Symbols	Explanations
m	Number of search terms in a query
w	A query term
k	Number of top ranked results needed for document retrieval
$H_i()$	Pseudo random function (PRF) using the i-th secret key
k_0	Secret key used to encode terms as posting lists keys
k_1	Secret key used to encode group IDs
k_2	Secret key used to encode positions for the buckets in the index
k_3	Secret key used to encode document member IDs
gid	Group ID of a document with a 2D ID representation
mid	Member ID of a document with a 2D ID representation
$\lvert G \rvert$	The number of document groups with a 2D ID representation
p	The position of a bucket in a posting list
$B(\cdot)$	The tag of a bucket in a posting list
P	Modulus of modulo positioning
$\mathsf{dtoken}(\cdot)$	The deblinding token from a client as an authorization
gtag	A query-specific group tag for a matched document
$(\mathsf{gtag}, H_3(mid))$	A query-specific document tag for a matched document
R	Random integer sampled for each query independently
$\vert\vert$	Concatenation in binary presentation
Q	Diffie-Hellman group modulus
q	Diffie-Hellman group size
g	Generator of a Diffie-Hellman group
$E(\cdot)$	Symmetric encryption with a random seed, e.g., AES256
U	The average of the maximum padding ratio in posting lists

3.1 Bucketed posting lists

Our scheme divides all private documents into groups and a document ID d is represented as $d = (gid, mid)$ where gid is its group ID and mid is a member ID in such a group. By using group IDs, a posting list is decomposed into a set of buckets, and each bucket only contains documents of the same group ID. Thus the inverted index is structured in two levels: a term points to buckets of documents, and each bucket captures a subset of a document group. The inverted index of QDT does not reveal group IDs directly. Instead, the p-th bucket with group ID gid in a posting list of term w is marked by a bucket tag defined as

$$B(gid, w, p) = H_1(gid) \cdot (H_2(w||(p \bmod P)))^{-1} \mod q \qquad (1)$$

where bucket position p is an integer counted from 1. Pseudo-random functions H_1 or H_2 use secret keys k_1 and k_2 respectively. Expression $(H_2(x))^{-1}$ means the modular multiplicative inverse of integer $H_2(x)$, namely, $H_2(x) \cdot (H_2(x))^{-1} \equiv 1 \mod q$. Symbol $\cdot$ denotes modular multiplication. We adopt this inverse operator, inspired by OXT (Cash et al., 2013, 2014) for blinding, so that a server cannot learn blinded information from the index without client authorization. That is because the above expression for each bucket tag is computed during index generation before the index is outsourced to a server. Such a server only knows the final value of a bucket tag and does not know its integer factors including $H_1(gid)$. As discussed in Sect. 3.2, a deblinding token is sent from a client as a piece of query-specific authorization information to remove this blinding factor during query processing.

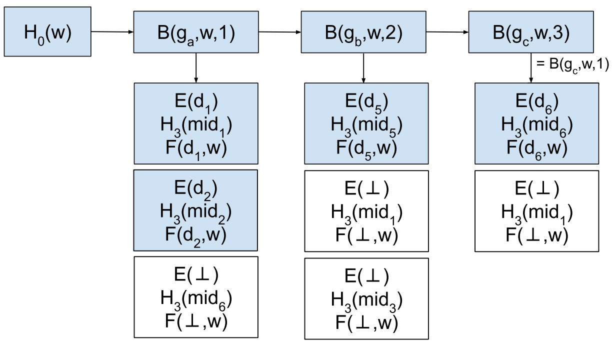
Notice that in the above expression, the modulo operator is applied to p as we adopt a modulo positioning technique (Lipmaa et al., 2000; Cash et al., 2013) which allows the cyclic re-use of the deblinding tokens from a client during query processing. Its impact on complexity control will be further discussed in Sect. 3.3. Symbol || denotes concatenation between two integers in binary representation, e.g., $11_b||01_b = 1101_b$.

Each bucket contains a number of posting records and each of them represents a document containing the targeted term w with the same group ID. The key to each posting list is encoded by a PRF called $H_0(w)$ using secret k_0. Each posting record for each document $d = (gid, mid)$ is a tuple $(E(d), H_3(mid), F(d, w))$ where

- $E(d)$ is a symmetric encryption result of document ID d, e.g., AES-256 with CBC mode (Dworkin, 2001). A document d may appear in different posting records and we use different seeds to compute the corresponding $E(d)$ values. Notice that this field $E(d)$ is needed only for a unigram term. The posting records that refer to the same document will be recognized with a query-specific document tag, and only one of these records needs to host the encrypted document ID, and this will be discussed in Sect. 3.2.
- $H_3(mid)$ is the hashed member ID using a PRF function with secret key k_3. Note that tuple $(B(gid, w, p), H_3(mid))$ uniquely identifies a document within the posting list of a term, but this bucket tag paired with mid is not sufficient to identify a document that appears in different posting lists, because the same document can have a different bucket tag in different posting lists.
- $F(d, w)$ is the encrypted or encoded element-based or vector-based feature of document d under feature key w. Ranking feature encoding and obfuscation are addressed in Ji et al. (2018), Shao et al. (2019) for tree based and neural ranking. Section 4 discusses feature value partitioning based obfuscation (Hacigümüş et al., 2002) used in our evaluation.

Figure 1 illustrates an example of the inverted index where the original posting list of term w has four real documents d_1, d_2, d_5, and d_6. Under QDT, the corresponding two-level index is organized with 3 group buckets. These buckets contain the above 4 real documents and also have padded extra 4 fake documents marked with symbol ⊥. Fake document

Fig. 1 An example of two-level inverted index with 3 group buckets



padding will be explained in Sect. 3.4. The first bucket contains 3 posting records. There are two real documents d_1 (with member ID mid_1) and d_2 (with member ID mid_2) containing term w in Group g_a. Similarly, the second bucket contains 3 posting records while the third bucket contains 2 posting records. Modulo positioning uses $P = 2$ and thus the third bucket has a bucket tag $B(g_c, w, 3) = B(g_c, w, 1)$.

3.2 Online query processing with QDT

Figure 2 shows the flow of query processing with QDT. Given a query with m terms as $H_0(w_1), \cdots, H_0(w_m)$, a client samples a random integer R, and computes a deblinding token at p-th bucket for each of search terms w as:

$$\mathsf{dtoken}(w,p) = g^{R \cdot H_2(w||(p \bmod P))} \mod Q. \tag{2}$$

Number P in the above expression limits the maximum number of tokens needed and this client sends all m terms $H_0(w_1), \cdots, H_0(w_m)$ and all $m \cdot P$ authorization deblinding tokens $\mathsf{dtoken}(w_i, p)$ for $1 \le i \le m$ and $0 \le p < P$ to the server.

After the client sends the above encoded terms and tokens to a server, this server linearly visits the bucketed posting lists of the searched terms one by one in the hosted inverted index. It computes a query-specific group tag at bucket position p for term w as

$$\mathsf{gtag} = \mathsf{dtoken}(w,p)^{B(gid,w,p)} \mod Q, \tag{3}$$

which is considered as a group tag for all documents in this posting bucket. Expanding the value of $\mathsf{dtoken}(w,p)^{B(gid,w,p)}$ using Expression 1, we have:

$$\begin{aligned} \mathsf{gtag} &= g^{R \cdot H_2(w||(p \bmod P)) \cdot H_1(gid) \cdot (H_2(w||(p \bmod P)))^{-1} \mod q} \mod Q \\ &= g^{R \cdot H_1(gid)} \mod Q, \end{aligned} \tag{4}$$

which is unique based on the group ID gid for the given query since value R is the same for different search terms of this query. This group tag is query specific, since R is chosen randomly for each query. Appendix 1 discusses this design further on preventing an attack.

Within a bucket, for each posting record, the server fetches its member ID information. The pair $(\mathsf{gtag}, H_3(mid))$ acts as a query-specific document tag and uniquely identifies a matched document that owns the feature stored in this record. Using this query-specific document tag, the runtime system builds a query-specific in-memory key-value hash table with this document tag as a key. The value of this key represents the corresponding set of features.

As the above index traversal linearly visits all posting records of multiple search terms, the features for the same document that appears in multiple posting records are

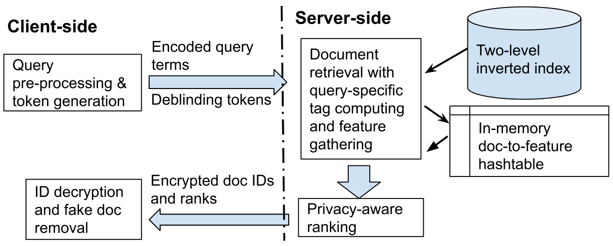

Client-side
Query pre-processing & token generation
Encoded query terms
Deblinding tokens
Server-side
Document retrieval with query-specific tag computing and feature gathering
Two-level inverted index
In-memory doc-to-feature hashtable
Privacy-aware ranking
Encrypted doc IDs and ranks
ID decryption and fake doc removal

Fig. 2 The flow of privacy-aware search with QDT

added to the hashtable gradually with the same query-specific document tag. Then the ranking features of this document can be recognized and aggregated.

The outcome of the above traversal is a set of matched documents and the features for each document accumulated in the above hashtable. This outcome can be incrementally injected as an input to a privacy-aware ranking scheme. As discussed in Sect. 2, our evaluation paper uses a simple obfuscation-based additive ranking which hides true BM25 values. Alternatively, one can opt to apply privacy-aware tree ensemble or neural ranking (Ji et al., 2018; Shao et al., 2019). Finally the server sends the ranked and encrypted document IDs to the client as shown in Fig. 2. The client will decrypt and may filter out some fake IDs to be discussed in Sect. 3.4.

To optimize time efficiency, the previous retrieval work with additive ranking has developed index skipping methods called WAND (Broder et al., 2003), BMW (Ding & Suel, 2011) and its variants (e.g. (Mallia et al., 2017; Shao et al., 2021)). These optimization methods avoid the processing of low-scoring documents below a top-k threshold to reduce the retrieval latency. We do not use such an optimization method because it requires that each posting list is presorted by static and deterministic document IDs to guide skipping. In our problem context with privacy protection requirement, document IDs in a search index are encrypted non-deterministically and the same document has different encrypted ID values in the different lists. Similarly, other optimization on list intersection (Culpepper & Moffat, 2010) that requires document pre-sorting cannot be leveraged. This represents an efficiency tradeoff for privacy.

3.3 Time cost of query processing with QDT

Having the two-level index design does not affect the effectiveness of document retrieval in terms of ranking quality. The main motivation of having this design is to control the time complexity. Assume that client–server communication cost is less significant to transmit encoded query terms and tokens, and top k results with relatively small k value. We assume to use the aforementioned obfuscated BM25 ranking, which is relatively fast and its cost is less significant compared to the other items listed below. Then the time complexity of query processing with QDT is dominated by the following expression:

$$m \cdot P \cdot T_{token} + m \cdot [T_{kvstore} + C \cdot T_{gtag} + L \cdot T_{hashtb}],$$

where

- m is the number of search terms derived from a given query;
- P is the modulus used in modulo positioning;
- T_{token} is the client-side time to generate one deblinding token;
- $T_{kvstore}$ is the average server-side time to access a posting list from a key-value store hosted on a disk drive;
- T_{gtag} is the server-side time to compute the group tag of a posting bucket;
- C is the average number of posting buckets in a posting list;
- L is the average total number of documents in the posting list per term;
- T_{hashtb} is the server-side time to perform a hash table lookup and/or insertion using a query specific document tag, and then combine the features of the corresponding document in this hash table.

Notice that cost parameter T_{token} is much larger than $T_{kvstore}$, and L is fixed. As shown below, controlling of parameters C and P is critical for a reasonable response time.

Cost parameter T_{gtag} is for computing a group tag with Expression (3) and involves integer exponentiation with a Diffie-Hellman group. Integers involved need to be in 2048 bits or more to be secure as a common cryptographic practice (Boneh & Shoup, 2015), and such exponentiation is extremely expensive. We use an elliptic curve based optimization to re-formulate exponentiation computation, represent each long-bit number with a pair of two numbers, and reduce the number of bits needed as 224 (Boneh & Shparlinski, 2001) instead of 2048 to speed up calculation, and it is still time-consuming. In our evaluation, $T_{gtag} \approx 0.13ms$ while $T_{hashtb} = 0.001ms$ using 256-bit elliptic curve-based modular exponentiation. With a standard one-level approach where $C = L$, the cost of tag computation would dominate. For 10,000 buckets and 5 search terms, group tag computation in processing a query would take around 6.5 s, and it is too slow for an interactive response. Our two-level design can allow $C << L$, which makes the tag computing time more affordable.

It is expensive for a client to compute deblinding tokens because token computation involves modular exponentiation, and in our evaluation, $T_{token} \approx 0.13ms$. The use of modulo positioning (Lipmaa et al., 2000; Cash et al., 2013) reduces the client token generation time because $\mathsf{dtoken}(w, p) = \mathsf{dtoken}(w, p \mod P)$, and parameter P limits the number of deblinding tokens computed at a client. Without limiting P, $P = L$ and client-side token computation would become a bottleneck.

With the above design consideration with $C << L$ and limited P value, the dominating part of query processing time cost of *QDT* is proportional to $m \cdot L$. Under a typical term distribution in an inverted index, $L \ll n$ where n is the total number of documents in a dataset. QDT can take advantages of an inverted index data structure for time efficiency like a traditional retrieval algorithm while there is still a performance gap due to the cryptographic operations used, which will be evaluated in Sect. 4.

A comparison with the previous work in design considerations While QDT adopts inverse-based blinding, following OXT (Cash et al., 2013), there is a major design difference of QDT compared to OXT for supporting the disjunctive query semantics while collecting ranking features correctly and efficiently. OXT combines a blinded static document ID d with a term ID w as a lookup key in a pre-computed static hash table, to check if document d contains term w in an index. This is designed for conjunctive query handling and may leak the cross-query two-term co-occurrence information. In comparison, QDT uses Expression (4) as the part of a query-specific hash table key which is term-independent and contains a query-dependent random number, and this is helpful in reducing inter-query information leakage of document sharing among posting lists involved in different queries.

For time efficiency, there is a significant difference between OXT and QDT due to the above design. OXT's list intersection scans the posting list of the first search term. For each document d in the first list, OXT uses key (d, w) to check if d contains w in a hash table for another term w. OXT's time complexity is dominated by expression $\Theta(m * L * (T_{keycomp} + T_{lookup}))$ where m is the number of search terms, L is the average posting list length, $T_{keycomp}$ is time to compose a key involving the exponentiation of long-bit integers, and T_{lookup} is the time to perform a hash table lookup. Parameter $T_{keycomp}$ is as expensive as T_{gtag} in QDT, and thus there is a large cost difference between the values of $m * L * T_{keycomp}$ in OXT and $m * C * T_{gtag}$ in QDT because $L >> C$ due to the two-level design in QDT. The hashtable tag lookup cost in QDT is less significant, because this query-specific hash table is small, while the hashtable involved in OXT is huge, and typically does not fit in memory.

Partial server additive ranking in Agun et al. (2018) uses the OXT's technique of inverse-based blinding, and thus can leak the cross-query two-term co-occurrence information. It follows the OXT's hashtable structure and lookup method driven by the first search term, requiring a large server memory to host its hash table. In addition, it does not allow multi-stage search with complex neural re-ranking conducted at the server side, and the server has to send a large amount of un-ranked results to the client-side with a significant communication overhead, which is not scalable for a large dataset with many results matching a query.

The HXT work in Lai et al. (2018) improves the security of OXT for conjunctive queries using hidden vector encryption and a Bloom filter with substantial overhead in multi-round client–server communication. Adopting HXT's encryption design with client–server collaborative communication at least would increase communication overhead by a factor proportional to the number of search terms. Multi-round communication between the server and client used in HXT and the previous work (e.g. (Naveed et al., 2014; Hu et al., 2011)) or extensive communication to rely more on clients (Agun et al., 2018; Xu et al., 2021) is not considered in this paper because they incur high communication cost and do not scale well for large datasets or a low-bandwidth platform. Like HXT, the work in Cash et al. (2013) also discusses the use of a Bloom filter to improve efficiency. Our work does not adopt the idea of a Bloom filter because with that, documents that do not satisfy query semantic may be matched due to false positives and wrong feature aggregation for some documents is possible, which yields incorrect ranking results.

3.4 Guided padding with fake documents

We adopt the idea of Kamara and Moataz (2019), Patel et al. (2019) to pad fake documents, which hides the length of real posting lists. We add a flag to a fake document ID before its encryption, thus a server is unable to detect fake documents added in an index and only a client can filter out fake document IDs mixed in the returned query results after ID decryption. A naive padding method based the idea of the previous work is to randomly add fake documents to each posting list and select the group ID and member ID of these fake documents from a large integer value space. This naive method can still leak document sharing patterns in an index and the reason is that an adversary server can identify a document approximately using its encoded member ID value when such values are relatively unique among documents, sampled from a large integer space, which is of size 2^{64} bits in our implementation.

There are three additional considerations. (1) The ranking feature values of fake documents cannot look dissimilar to real documents so that the server would not be able to detect fake documents easily. (2) The existence of fake documents should not affect the effectiveness of top candidate selection in ranking. (3) We want to add fake documents to the existing groups as much as possible because online computing of group tags is expensive and the addition of new groups needs to be avoided.

With the above considerations, we need a padding strategy different from that of Kamara and Moataz (2019), Patel et al. (2019). Our goal is to not only hide the distribution of the posting list lengths but also make padded documents indistinguishable from real documents while limiting the addition of new group IDs. Our idea is to use ID information and the feature value distribution of real documents to guide fake document generation, and also obfuscate these hashed member IDs with k-anonymity (Sweeney, 2002; Di Castro et al., 2016). Namely, each real member ID corresponds multiple documents appearing in a posting list. That is accomplished by mixing this member ID with $k-1$ randomly chosen fake member IDs and any attacker cannot distinguish. The example in Fig. 1 illustrates the inclusion of 4 fake documents where value mid_1 appears 3 times, and Document d_1 cannot be distinguished well by only using value mid_1.

Let U be the maximum padding factor. Given a posting list of term w with length r, we select a random number u in interval $[1, U \cdot r]$ and u represents the total number of fake documents to be added. Let G' be the set of all possible group IDs available to choose for a fake document, and our guided padding algorithm to add u fake documents to this posting list repeats the following steps:

1. Let G memorize all group IDs used so far. Initially G contains all group IDs used in this given posting list. The algorithm gradually expands G towards G' to cover more valid group IDs when needed. This allows the distribution of fake member IDs to be about the same as that of real member IDs.
2. Randomly sample mid' from D_{mid} and $feat'$ from D_{feat}. Here D_{mid} is the distribution of the member IDs of real documents in the entire index and D_{feat} is the distribution of the ranking features of the real documents in the entire index.
3. Randomly sample gid' from G such that (gid', mid') has not been appeared in this posting list. Add $(gid', mid', feat')$ as a new fake document with feature $feat'$ to this posting list.

Since ranking features of fake documents follow the real feature distributions in our padding method, fake documents have an equal chance to be selected on the top in ranking. With an average of maximum padding ratio being U, the number of fake documents added is about $\frac{U}{2}$ in the top results, an earlier stage ranking algorithm should enlarge search scope proportionally and submit about $K'(1+\frac{U}{2})$ top results to the next-stage ranking.

4 Evaluation

Datasets and setting We use the following TREC test collections for evaluations because they are widely adopted for ad-hoc keyword search studies (e.g. (Guo et al., 2016; Xiong et al., 2017; Dai et al., 2018; Ji et al., 2018; Shao et al., 2019)): (1) **Robust04** uses TREC Disks 4 & 5 (excluding Congressional Records), which has about 0.5M news articles. (2) **ClueWeb**09-Cat-B uses ClueWeb09 Category B of 50 M web pages (Jamie Callan's research group, 2020). Spam filtering is applied on ClueWeb09-Cat-B using Waterloo spam score with threshold 60. For ClueWeb, we have further divided the dataset into 60 partitions for parallel query processing thus the report time is the parallel time with 60 cores. For Robust04, the average number of documents for each posting list is 9789, while the average number of buckets is 355. For ClueWeb, the average number of documents for each posting list is 15,462, while the average number of buckets is 394.

The relevance features include BM25 term scores for single words and word pairs in the title and body sections. To control the number of word-pair terms, we limit word pairs under distance 3 in indexing and query term generation. BM25 features are obfuscated by 50 non-uniform value partitioning with extreme down-sampling (Hacigümüş et al., 2002; Ryoo et al., 2017) to hide privacy-sensitive details. We leveraged Indri (Strohman et al., 2005) to generate the posting lists and produce a two-level index. We have used 256-bit elliptic curve optimization instead of using 2048 bit long integers (Boneh & Shparlinski, 2001) as discussed in Sect. 3.3. The majority of indexing time comes from encryption of document IDs and bucket tag computations with 256-bit elliptic curves, which is still slow. For example, it takes about 960 CPU hours to generate the encrypted index for ClueWeb, and this is about 64x slower than traditional indexing with Indri (Strohman et al., 2005) without encryption.

Our evaluation uses 250 queries with average query length 2.64 from TREC Robust 2004 and 2005, and 200 queries with average query length 2.48 from TREC Million Query 2009 to 2012 for ClueWeb. Experiments are conducted on Linux servers and each has Intel i5-8259U 2.3GHz, 32GB DDR4 memory and NVMe solid-state drives (SSD). We have implemented QDT in C++ and the code is compiled with flag -O3. We report the average response time in query processing and the relevance score which includes the normalized discounted cumulative gain (NDCG) (Järvelin & Kekäläinen, 2002) and the precision of the top results (Baeza-Yates & Ribeiro-Neto, 2011). NDCG@p with a value between 0 and 1 measures the quality of a ranking result against ideal ranking for top p positions. All reported query time numbers x are within 95% confidence interval $x \pm 0.01$ by gathering data through multiple runs.

4.1 Time efficiency and ranking relevance

Search relevance in the presence of fake documents Table 2 lists the relevance score of QDT in NDCG and precision (Järvelin & Kekäläinen, 2002) for Robust04 and ClueWeb after filtering out padded fake documents. It also lists the relevance scores of several up-to-date baselines for ranking with and without privacy constraints in searching the tested TREC datasets. Row 3 is for QDT with BM25-based ordering without using fake documents and feature obfuscation. Row 4 is for QDT with BM25 ranking with obfuscated features and fake documents. Row 5 is for re-ranking with a privacy-aware neural model called ConvKNRM/TOC (Shao et al., 2019; Dai et al., 2018) after QDT retrieval and obfuscated BM25 first-stage ranking. Since final results received by a client contain fake documents, the server needs to send more top results than needed, depending on the maximum padding ratio. Ranking selects top 1000 documents when no fake documents are padded, and top 1500 documents when the maximum padding ratio is 1. The result shows that our padding of fake documents has no visible impact on relevance score for Robust04. We have performed a pairwise t-test on the retrieval with and without fake documents,

Table 2 Client-side view of relevance for ranking with and without privacy constraints

	Robust04		ClueWeb	
	P@20	NDCG@10	P@20	NDCG@10
Privacy-aware ranking				
QDT/BM25 w/o fake docs	0.346	0.428	0.280	0.210
QDT/BM25 w. fake docs	0.346	0.428	0.274	0.204
QDT + ConvKNRM/TOC w. fake	0.395	0.449	0.399	0.310
PAR: Partial server-side additive ranking (Agun et al., 2018)	–	0.412	–	0.255
ConvKNRM/TOC re-ranking (Shao et al., 2019)	–	0.450	–	0.310
BERT-based neural ranking with no privacy constraints				
SPLADE sparse retrieval (Formal et al., 2022; Thakur et al., 2023)	–	0.468	–	–
BECR: Composite re-ranking (Yang et al., 2022)	0.401	0.491	0.399	0.342

which shows that there is no statistically significant difference at the 95% confidence level. For ClueWeb, although there is a small relevance degradation during retrieval, almost all desired documents still appear in top retrieved results.

Table 2 compares QDT with two baselines for privacy-aware ranking. Row 7 is for partial server-side additive ranking with a linear combination (Agun et al., 2018) (we call it PAR). PAR underperforms QDT with neural ranking due to two reasons: (1) It does not use semantic matching with neural embeddings. (2) It uses conjunctive query semantic, reducing the chance of finding semantically-relevant documents that do not contain all query keywords. Row 8 is the relevance result of ConvKNRM/TOC published in Shao et al. (2019), which assumes the top results and their ranking features are fetched safely with a privacy protection. Its result is comparable on average to Row 5. Thus QDT provides a privacy-aware proper retrieval support for neural re-ranking with ConvKNRM/TOC.

As a reference, Table 2 lists the relevance score of two recent BERT-based ranking methods without privacy constraints. Row 10 is for SPLADE v2 (Formal et al., 2022, 2021) with a learned sparse index and knowledge distillation. Its Robust04 performance is collected from Thakur et al. (2023). Row 11 is for BECR composite re-ranking (Yang et al., 2022) which uses transformer-based contextual embeddings for document and query representation. These latest methods without privacy constraints outperform QDT in Row 5 that uses static neural embedding while the gap is modest with 4% and 9.4% for Robust04, and 9.4% for ClueWeb. They represent complementary and orthogonal optimizations and leveraging such optimization in QDT will be a future work to exploit. Addressing privacy in transformer-based models is still an open problem in general, and their query processing is much more expensive due to their complexity. Thus our current work represents a trade-off in advancing the state of the art on privacy-aware ranking.

Time cost breakdown in query processing with QDT Table 3 lists the average query processing time and its cost breakdown when the number of query words varies. Row 1 is the query length and the number of query words used for each length. For both datasets, the maximum padding ratio is 1 and on average 33% of documents are fake. The query response time breakdown is listed as follows. Rows marked "Client" are client-side preprocessing time mainly dominated by token generation; Rows marked "Comm." means

Table 3 Query time in seconds with max padding ratio 1

#Query words		1	2	3	4–5	Average
Robust04	Client	–	0.04	0.07	0.11	0.06
	Comm	0.020	0.02	0.02	0.02	0.02
	KV-store	0.008	0.02	0.04	0.06	0.04
	Tag comp	–	0.16	0.30	0.50	0.25
	Hash table	–	0.04	0.18	0.30	0.11
	Decrypt	0.020	0.02	0.02	0.02	0.02
	Total (s)	0.048	0.30	0.63	1.01	0.50
ClueWeb	Client	–	0.03	0.06	0.10	0.05
	Comm	0.020	0.02	0.02	0.02	0.02
	KV-store	0.012	0.03	0.06	0.09	0.06
	Tag comp	–	0.19	0.38	0.60	0.29
	Hash table	–	0.11	0.30	0.44	0.19
	Decrypt	0.020	0.02	0.02	0.02	0.02
	Total (s)	0.052	0.40	0.84	1.27	0.63

the client–server communication of sending tokens and encoded terms, and receiving the top ranked results, and we have assumed networking speed of 10ms latency and 5Mbits/s bandwidth; Rows marked “Tag comp.” are the server-side time for group tag computation; Rows marked “Hash table” are the server-side time to combine features by document tags through a hash table; Rows marked “Decrypt” are the client-side post-processing time to decrypt ranked document IDs and filter out fake documents.

Table 3 shows that the client-side cost of computing deblinding tokens is not significant with the use of modulo positioning. The group tag computation takes a significant portion on the server side, and this server-side cost is proportional to the number of buckets. Thus two-level indexing is effective to control this portion of the cost. As the query length increases, the number of deblinding tokens and involved posting lists increases, and thus there is an increasing amount of client-side token computation and server-side group tag computation. The cost of hash table operations for feature aggregation is relatively small. The time on posting list access from the SSD-based key-value store is small as the number of store lookups is equal to the number of search terms.

Impact of two-level indexing on retrieval time Table 4 shows the query time in three settings when maximum padding ratio 1: (1) one-level index with no group buckets (Column 3); (2) two-level index with 1024 groups but without using modulo positioning (Column 4); (3) two-level index with 1024 groups and modulo positioning with $P = 128$ (Last column). Only the cost of client-side token computation, client–server communication, and server-side tag computation is listed as grouping only affects these three items significantly. Without two-level indexing, there is much more cost in client-side token computation and server-side tag computation. For ClueWeb, two-level indexing without modulo positioning reduces the average number of deblinding tokens needed per term from about 15,462 to 1,024 in this case. Modulo positioning furthers reduces the number of tokens to 128, but the number of bucket group tags to compute does not change. The communication cost is also reduced proportionally with fewer deblinding tokens. The overall speedup is up to 130x after the use of the two-level design from the ungrouped setting. Modulo positioning brings an additional 1.8x speedup. The speedup for Robust04 is 131x with the two-level design.

Impact of padding on query time and storage cost Table 5 lists the query time and storage space cost without padding, and with maximum padding ratio of 0, 1, 2, and 3. For ClueWeb, the storage cost listed is the total gigabytes for all partitions. Impact of adding fake documents to buckets on the query time is mainly on hash table operations since this mainly enlarges the number of posting records per list randomly. The storage space

Table 4 Query time with or without two-level indexing

Grouping method		Ungrouped	1K groups w/o modulo pos.	1K groups $P = 128$
Robust04	Client	6.38	0.47	0.06
	Comm	0.38	0.10	0.02
	Tag comp	123	0.25	0.25
	Total (s)	129.93	0.99	0.50
ClueWeb	Client	9.27	0.47	0.05
	Comm	0.53	0.11	0.02
	Tag comp	138	0.29	0.29
	Total (s)	148.07	1.14	0.63

Table 5 Impact of padding on query time and space cost

Max padding ratio		U=0	U=1	U=2	U=3
Robust04	Hash table	0.07	0.11	0.16	0.20
	Total query time (s)	0.46	0.50	0.55	0.59
	Index storage (GB)	5.8	8.6	11.4	14.2
ClueWeb	Hash table	0.11	0.19	0.26	0.34
	Total query time (s)	0.55	0.63	0.70	0.78
	Index storage (GB)	510	762	1008	1260

proportionally increases when the maximum padding ratio increases since the posting lists consume a majority portion of space.

Storage cost 32 bytes are used for the ID of a posting bucket. Each posting record for a unigram takes 38 bytes, which includes 32 bytes encryption of document ID (initialization vector and one AES block, each of which needs 16 bytes), 2-byte hashed member ID, and 4-bytes for basic features. For a word pair, the document ID field is not needed as explained in Sect. 3.1. Since values of document ID encryptions and bucket tags generated are uniformly distributed long-bit random integers following the standard cryptographic requirement, the index of QDT cannot be significantly compressed. In comparison, the traditional index produced by Indri (Strohman et al., 2005) takes about 107GB for ClueWeb and hence the QDT index without no padding ($U = 0$) has about 4.8x more space cost. This is mainly caused by the non-compressibility of random integers with long bits in our index and the use of word-pair-based terms.

Comparisons of retrieval time with and without privacy constraints Table 6 lists a baseline comparison of document retrieval time without or with privacy constraints. Indri (Strohman et al., 2005) is an open-source search engine. Each of VBMW/p and DBMW/p (Shao et al., 2021) is a recent variant of BMW (Ding & Suel, 2011; Mallia et al., 2017), which optimizes index navigation by retrieving documents in blocks to skip low-score documents. The above retrieval methods do not consider privacy. In comparison, QDT response time is 0.50s and 0.63s respectively where the hash-table based feature gathering during list traversal costs around 0.11s and 0.19s as shown before in Table 3. The higher latency in QDT compared to VBMW/p and DBMW/p represents a tradeoff for privacy. As discussed in Sect. 3.2, we do not use BMW-based optimization in Shao et al. (2021), Ding and Suel (2011), Mallia et al. (2017) because it requires each posting list to be presorted by static and deterministic document IDs, which is not possible in our privacy-aware setting where the same document has different encrypted ID values in the different posting lists.

Excluding tag computing cost, the list traversal of QDT is dominated by the conversion from a 512-bit document tag to a 32-bit hash-table key (~ 70%), and without index

Table 6 Retrieval time in seconds with and without privacy constraints

Dataset	Indri (Strohman et al., 2005)	DBMW/p (Shao et al., 2021)	VBMW/p (Shao et al., 2021)	PAR (Agun et al., 2018)	PAR/D (Agun et al., 2018)	QDT
Robust04	0.032	0.014	0.011	0.60	1.1	0.50
ClueWeb	0.071	0.025	0.019	0.87	4.3	0.63

skipping, it is slower than VBMW/p and DBMW/p while gaining privacy. Excluding these two factors, our list traversal time is on a par with the existing work. This comparison reconfirms that the price paid for supporting privacy is significant because of expensive cryptographic computation with very long bits. That also means optimization studied in this paper is necessary to bring down the cost. Our scheme can deliver a sub-second response time after using the proposed optimization, otherwise, the cost would be about 130x bigger as shown in Table 4.

Table 6 also lists the retrieval time of PAR ranking (Agun et al., 2018) under the same setting with privacy-aware partial server ranking. PAR is an extension and improvement over OXT (Cash et al., 2013) for ranking with conjunctive queries. QDT is faster than PAR, even excluding the client-side communication cost incurred in PAR. Notice that PAR has a lower relevance as discussed in Table 2, partially because of its conjunctive constraint. The extension of PAR (called PAR/D) to follow disjunctive query semantics is listed in Table 6 as well. The result shows that QDT is 2.20x faster than PAR/D for Robust04 and is 6.83x faster for ClueWeb. PAR/D converts a disjunctive query as several conjunctive subqueries with different s-terms. For each subquery, PAR/D calculates the union of documents that match such a subquery and aggregates ranking features for the same document IDs using OXT-based hashtable lookups. Since PAR with partial server-side ranking requires a client to conduct final ranking, there is a large amount of data sent from the server to the client. The test setup in this evaluation has 100 megabits per second communication bandwidth between the server and a client, which is reasonable on average with a home cable or 5 G cellular Internet connection in USA. The client–server communication in PAR/D takes 0.03 s for Robust04 and increases to 2.7 s for ClueWeb. This large communication cost increase is caused by the fact that ClueWeb dataset is 100x larger than Robust04. Excluding this client–server communication overhead, QDT is still 60% faster for Robust04 and 2.54x faster for ClueWeb compared to PAR/D. In general, QDT provides more flexibility to reach a higher relevance score with a lower cost because PAR or PAR/D does not conduct full server ranking and it pays a significant communication overhead to let a client collect a large amount of data for further ranking, in addition to the hashtable cost issue explained in the end of Sect. 3.3.

4.2 Guided padding for attack prevention

We demonstrate a plaintext-recovery attack exploiting the co-occurrence leakage (Islam et al., 2012) of two or three words to recover the plaintext of encoded word IDs appeared in a hosted index and examine how guided padding in QDT discussed in Sect. 3.4 thwarts such an attack. This 2-word attack makes the following assumptions:

1. A server hosts the index for dataset D (Robust04 in this case) and knows the plaintext of a set of English words included in D, called E. We assume 10% of documents in D are public or injected by the server adversary by some means and the server can use them to approximate the co-occurrence probability of two words in E among documents of D. The corresponding co-occurrence matrix is called M. Recall that the co-occurrence probability of two words w_1 and w_2 is the number of documents in D that contain both w_1 and w_2, divided by $|D|$. The server does not know the IDs of English words in E and it wants to recover plaintext-to-ID mappings for some of these words.
2. The server estimates the co-occurrence probability of two word IDs using one of the two methods below and the corresponding probability matrix is called M':

- *A*) Observe processing of some 2-word queries without knowing their plaintext, and these query word IDs form Set E'. We assume $E' \subseteq E$. By observing the execution of QDT, the server can guess two matched documents for a query in different postings are the same if their query-specific document tags are the same, even such a document could be fake. Let Y be the fraction of all possible word pairs formed from E' whose co-occurrence can be estimated using Method A.

  Initially, $Y = 1$, namely all pairs of two words in E' appear in these observed queries. We will vary Y value to assess if our finding is similar when not all pairs of the two words in E' appear in these queries.
- *B*) Inspect the inverted index, and guess two documents appeared in the postings are the same if their group member IDs are equal.

3. The server has already obtained a mapping from word IDs to the plaintext for a small number (X) of English words in E'. The goal of server attack is to recover the ID mapping of plaintext for the English word IDs in subset E'. In our evaluation, $|E'| = 150$ and we vary the value of parameter X from 20 to 0 and parameter Y from 1 to 0.1.

Attack method The server approximates the co-occurrence probability matrix M' for set E' using the above Method A or B. Next, the attack algorithm follows the simulated annealing technique (Kirkpatrick et al., 1983) used in Islam et al. (2012) to find a mapping from the word IDs in matrix M' to the plaintext words of a sub-matrix of matrix M with a minimized co-occurrence matching error.

Table 7 shows the number of recovered English words in Set E' in Columns from 2 to 4 marked "Eq. member ID guess" by using Method B based on equal member ID values to guess the term co-occurrence probability. X varies from 20 to 0 in these 3 cases. The rest of the columns uses Method A by collecting which documents have the equal query-specific document tags during query processing under various values of X and Y. The attack computing time for each configuration setting takes a few hours. It should be noted that the server does not know if the recovered word mapping from an ID to a plaintext is correct or not, and thus entry value 14 means that the server's plaintext recovering accuracy is 14 out of 130 unknown words given $|E'| - X = 130$.

The first column from Row 2 in Table 7 lists the different padding methods used as discussed in Sect. 3.4. We explain the second column with $X = 20$ as follows. If there is no padding of fake documents (Row 2), Though matrix M is approximated, the recovery accuracy of this attack is 14 out of 130, given 20 known mappings. When the naive padding method is used, the selection of random IDs for fake documents is unbiased and these IDs are unlikely to collide. Therefore, this strategy does not obfuscate the co-occurrence

Table 7 Recover 150 words under various padding strategies

Method	Eq. member ID guess			Eq. document tags; X=20			X = 10	X = 1	X = 0
	X = 20	X = 1	X = 0	Y = 1	Y = 0.5	Y = 0.1	Y = 1		
No padding	14	2	0	15	2	1	7	2	0
Naive pad	13	2	0	9	2	1	7	2	0
Guided, U=1	2	0	0	1	0	0	1	0	0
Guided, U=2	0	0	0	0	0	0	0	0	0
Guided, U=3	0	0	0	0	0	0	0	0	0

information, and the server can still recover 13 English words in E'. By applying the guided padding with padding ratio $U = 1$, the recovery accuracy decreases to 2 out of 130. As U goes higher, M' is totally different from M with much more noise. As a result, no words are recovered. Table 7 also varies X and Y values. As X becomes smaller, less known ID-to-word mappings are available to guide the attack to perform successfully. For each setting, guided padding decreases the server's recover accuracy significantly and it becomes zero when $U \geq 1$ or $U \geq 2$. As Y becomes smaller, less information is available to guide simulated annealing in finding a good word matching, and less words are recovered.

Our attack using 3-word co-occurrence leakage is similar as above with a 3D co-occurrence matrix. Following a similar setting as Table 7, we find that a server adversary can recover up-to 3 words by exploiting the 3-word co-occurrence with naive padding or without padding, and our guided padding strategy effectively decreases the number of recovered words to zero with $U \geq 1$.

We have also evaluated QDT under a count attack (Cash et al., 2015) which further exploits the use of a posting list length under a slightly different co-occurrence assumption. The results show that guided padding is still effective and decreases the number of recovered words to zero with $U \geq 1$ or 2.

5 Concluding remarks

The contribution of this paper is a privacy-aware document retrieval scheme with query-specific tagging and two-level indexing to minimize the leakage of sharing patterns when matching documents and gathering their features for ranking. Our evaluation shows that two-level indexing in QDT can control query processing cost with up-to 131x time speedups in the tested cases compared to a single-level design, and guided padding can effectively thwart leakage-abuse plaintext attacks that exploit word co-occurrence information. Compared to a traditional retrieval scheme with no privacy constraints, QDT does pay a significant space and time cost for enhancing privacy protection with the use of cryptographic operations on long-bit numbers, and this also demonstrates the importance of efficiency optimization proposed in QDT.

Appendix 1: Document-pair ID mapping attacks

Our group tag design in Sect. 3 intends to prevent the following attacks based on mapping of exposed document-pair ID information including group tags. As defined in Cash et al. (2015), given an inverted index with m terms, let C be the $m \times m$ co-occurrence count matrix where each entry $C_{i,j}$ is the number of documents in which terms w_i and w_j both occur. This matrix can be normalized as a co-occurrence probability matrix, assisting a plaintext recovery attack (Cash et al., 2015).

Proposition on document-pair ID mapping attacks. Assume that any entry in a co-occurrence count matrix C is no less than 2. In an inverted index without fake document padding, let $z(d)$ be the observed information regarding document d during search. If there exists mapping function f where $f(z(d_1), z(d_2))$ is unique and query-independent for any pair of documents d_1 and d_2, this server could derive matrix C for plaintext recovering in some cases.

Attack sketch Notice that each entry $C_{i,j}$ of co-occurrence count matrix C is the intersection count of two posting lists for w_i and w_j. Consider a case that $C_{i,j} \geq 2$. Let y be the number of pairs of documents which appear in both posting lists of w_i and w_j, then

$$y = C_{i,j}(C_{i,j} - 1)/2.$$

If value y can be estimated by the server adversary, $C_{i,j}$ can be solved from the above equation. Now we describe how the server can estimate this y value. Since the server can compute and observe the value of $f(z(d_1), z(d_2))$ when visiting term w_i during search. As such a value is unique, the server can observe if this mapped value appears when visiting the posting list of another term w_j, and it can estimate y value by looking at the f value of all document pairs in the posting lists of w_i and w_j. To derive all entries in matrix C, all possible terms need to appear in multiple queries so that a server can observe their processing, and such a case may exist. ■

Discussion If the uniqueness of f value is approximately true, the server can still estimate y with a reasonable accuracy. In QDT, a server can observe a group tag valued as $g^{R \cdot H_1(gid)} \mod Q$ and the member ID for each matched document. Since it is computationally hard to infer $R \cdot H_1(gid)$ from such a group tag and R is a query-specific random number, the chance that the condition specified in the above proposition is true is negligible. If the group tag formula were designed as $g^{H_1(gid)} \mod Q$ or just $R \cdot H_1(gid) \mod Q$, the group tag ratio of two documents could satisfy the condition of the above proposition, and thus we do not accept such a design.

Appendix 2: Properties of QDT

We discuss the properties of QDT on its correctness in handling authorized or unauthorized queries for document retrieval with ranking, and on information protection and leakage related to document sharing patterns among posting lists.

Property 1 *For a query authorized by a client, a server with QDT can correctly follow the disjunctive query semantic, and gather the term features for each matched document. For an un-authorized query and document d containing two query search terms* w_1 *and* w_2*, if one of* w_1 *and* w_2 *is not seen in the query history or both of them only appear in two past queries with no overlapping terms, then this server cannot recognize the term features of* w_1 *and* w_2 *belonging to document d for any document d matches this query.*

Proof For a query authorized by a client, it is straightforward to show that QDT gathers term features correctly based on group tag Expression (3) in Sect. 3.2. Given document $d = (gid, mid)$ that appears in some or all posting lists of the multiple search terms, the features of posting records in different lists belonging to d are recognized based on the same group tag and member ID pair as all of them have the same group tag value $g^{R \cdot H_1(gid)} \mod Q$ with the same number R.

When a query is un-authorized, a server may try to use the past queries to gather term features for a document. Given document d containing two search terms w_1 and w_2, there are two cases to analyze. Case 1: assume w_1 is not seen in the past queries. Given a bucket stored in the hosted index and its bucket tag computed based on Expression 1, it is computationally hard for a server to remove integer factor $H_2(w||(p \mod P)))^{-1}$ because such an integer uses a very large number of bits, and its randomness is guaranteed by definition of

PRF H_2. By inspecting an index without authorization tokens, a server is unable to compute the group tag for document d in the posting list of term w_1, and cannot recognize the two features under w_1 and w_2 belonging to the same document d.

Case 2. When w_1 appears in past query Q_1, w_2 appears in past query Q_2, and Q_1 and Q_2 do not overlap, this means the server accesses completely different posting lists when processing Q_1 and Q_2 and there is no common memory address that can be used to associate the posting records accessed when processing Q_1 and Q_2. Since these group tags produced for d in above process involve a query-dependent random number, the server is unable to infer two posting records of w_1 and w_2 have features belonging to the same document.

□

Note that when an un-authorized query Q contains two past sub-queries with overlapping terms such as "w_1w_2" and "w_2w_3", and document d contains w_1, w_2 and w_3, a server could infer that their corresponding posting records belong to d through common term w_2, and thus gather d's features together correctly for Q. However the server does not know if d is fake or not.

Is it possible that a server adversary inspects the inverted index and uses member IDs to guess the equality of two documents in two posting lists to derive the two-term co-occurrence? The following property shows that the chance is small. Considering a server needs to derive the co-occurrence probability for many term pairs in order to launch a successful attack, it is unlikely that this server can succeed. Section 4.2 gives an evaluation.

Property 2 *Given a two-level inverted index with* $|G|$ *groups, for any two documents with the same member ID that appear in two different posting lists, the probability that they refer to the same real document is at most* $\frac{1}{|G|}$.

Proof Suppose we have two different non-empty posting list: one is for term w_1 with r_1 real documents, and the other is for term w_2 with r_2 real documents. For the posting list of term w_i where $i = 1$ or 2, u_i fake documents are padded where u_i is uniformly sampled from the interval $[1, U_i \cdot r_i]$. Following the proof idea in Cash et al. (2013) which is based on security assumptions of Diffie-Hellman group in Sect. 3, any server cannot efficiently find overlapping documents between two posting lists offline, and thus the probability that two documents with identical member ID from different posting lists refer to the same document (real or fake) is at most $1/|G|$. Note that since any server cannot efficiently distinguish symmetric encryption of document IDs, and all fake member IDs and ranking features are sampled from the real distributions, any server cannot efficiently distinguish fake documents and real documents in the encrypted index. Hence given u_i fake documents padded to a posting list with r_i real documents, the probability that any document selected by the server is real is $r_i/(r_i + u_i)$.

Let $\mathbf{Pad}(x, w)$ denote the event that the posting list of term w is padded with x fake documents. Following the padding method described in Sect. 3.4, the probability that any documents d_1 from posting list of w_1 and d_2 from posting list of w_2 with the same group member ID refer to the same real document is

$$
\begin{aligned}
&\Pr[(d_1 \text{ is real}) \wedge (d_2 \text{ is real}) \wedge (d_1 = d_2)]\\
&= \sum_{x=1}^{U_1 \cdot r_1} \sum_{y=1}^{U_2 \cdot r_2} \Pr[\mathbf{Pad}(x, w_1)] \Pr[d_1 \text{ is real} | \mathbf{Pad}(x, w_1)] \cdot \\
&\quad \Pr[\mathbf{Pad}(y, w_2)] \Pr[d_2 \text{ is real} | \mathbf{Pad}(y, w_2)] \cdot \\
&\quad \Pr[d_1 = d_2 | d_1, d_2 \text{ are real}]\\
&\le \sum_{x=1}^{U_1 \cdot r_1} \sum_{y=1}^{U_2 \cdot r_2} \frac{1}{U_1 r_1} \frac{r_1}{r_1 + x} \cdot \frac{1}{U_2 r_2} \frac{r_2}{r_2 + y} \cdot \frac{1}{|G|}\\
&\le \frac{1}{U_1 r_1 U_2 r_2 \cdot |G|} \int_1^{U_1 \cdot r_1 + 1} \int_1^{U_2 \cdot r_2 + 1} \frac{r_1}{r_1 + x - 1} \frac{r_2}{r_2 + y - 1} dxdy\\
&= \frac{r_1 \ln(1 + U_1) r_2 \ln(1 + U_2)}{U_1 r_1 U_2 r_2 \cdot |G|} \le \frac{1}{|G|}.
\end{aligned}
$$

The last step uses the fact that $\ln(1 + x) \le x$ for any $x > 0$. □

Can a server learn some document sharing patterns of posting lists across queries? That is a hard problem, and even without ranking support, under the single-round client–server communication protocol, IEX and OXT designs leaks document sharing patterns in certain cases when executing a sequence of queries, which can lead to an attack concern. The following property examines what is protected and what is leaked in QDT with ranking support in terms of document sharing patterns among posting lists during the execution of a sequence of queries. While the design in QDT with query-specific document tagging tries to minimize the chance of learning from tags across queries, QDT is not perfect because there exists some special cases that some document sharing pattern among posting lists can leak. We discuss below the likelihood a server can take advantages to launch an attack and how to mitigate.

Definition Given a query history, two terms w_0 and w_{k+1} have transitively co-appeared in the query history if there exists a term sequence k terms $w_1, w_2, \cdots, w_k$ such that w_i and w_{i+1} both appear in a query in the history for all integers i from 0 to k.

Property 3

1. *For any two terms w_0 and w_k that have transitively co-appeared in a query history through a term sequence $w_1, w_2, \cdots, w_k$, a QDT-based server can detect if a document has appeared in all posting lists of these $k + 2$ terms from w_0 to w_{k+1} while the server does not know if this document is real or fake.*
2. *For any two terms w and w' that have not transitively co-appeared in the given history, a QDT-based server is unable to gain additional information through online query processing compared to offline index inspection to correctly detect document sharing of the posting lists of w and w'.*

Proof For Statement (1) in this property, there are two cases to analyze.

- Case 1.1. For a single query that contains $w_1$and w_2, a server that uses QDT to process this query can use the posting record positions to identify different documents, observe the hash-table based feature aggregation for the same documents that appear in posting lists of the two terms.
- Case 1.2. If a document has appeared the posting lists of all terms $w_0, \cdots, w_{k+1}$, and terms w_i and w_{i+1} have appeared pairwise in a query in the history for all integers i from 0 to k, then the server would know such a document by connecting the QDT-based processing history of these queries and using the posting position to identify such a document in the corresponding posting lists in a pairwise manner.

For Statement (2) of this property, there are two cases to analyze.

- Case 2.1. When these two terms w and w' have not appeared in the same query, or transitively co-appeared in a query sequence the result from Statement (1) of this property would not apply.
- Case 2.2. There are two sub-cases.
  - When any of these two terms, say w, has never appeared in any query, query processing does not visit the posting list of w, and thus no information is gained by a server in observing query processing.
  - When term w has appeared in query Q, and w' has appeared in another query Q', a document tag is generated for every document in the posting list of w during processing of Q. Such a tag is query specific due to the use of a random number in the tag generation formula. Notice transitively there exists no sequence of queries that allows the positions of posting listings can be associated from one query to another with respect to w and w' like Case 2.1. Any document tag produced in processing Q cannot be associated with another document tag produced in processing Q'. Thus the server cannot gain additional information such as ID equality during query processing on the document sharing of posting lists of w and w'.

□

The above property shows that there are some special cases that a server can learn the document sharing of the posting lists of two terms if these two terms appear in the same query or transitively co-appear in a sequence of queries. But even that occurs, QDT does not reveal such a document shared is real or fake, and thus the term co-occurrence probability estimated by such document sharing patterns is still incorrect. As demonstrated in Islam et al. (2012), Cash et al. (2015) and Sect. 4.2, to successfully form an attack, enough client-authorized queries that leak the document co-occurrence probability of many two-term pairs. This is less likely to happen based on the above analysis. In practice, we can install a mechanism at the client side to detect and warn if the query history becomes too big, and a client can periodically re-generate the QDT-based index with different fake document padding and PRF keys.

Appendix 3: Leakage profile

This appendix gives the leakage profile of document retrieval with QDT and this leakage profile is a summary of information leaked to a server during query processing. The previous work (Cash et al., 2013; Kamara & Moataz, 2017) studies such a leakage profile by considering a searchable encryption system is semantically secure if the information that any adversary can acquire during the execution of the real search protocol can be well simulated by an interactive experiment between this adversary and a simulation algorithm which only knows the leakage of this search system. To follow the above framework, we descibe the basic notation in details below.

Notion: Let Π be a searchable symmetric encryption scheme (Cash et al., 2013; Kamara & Moataz, 2017), which consists of two algorithms: $\mathsf{EDBSetup}$ distributes the keys to the client and server and encrypts inverted index; Search executes the search protocol between the client and server. Let $\mathcal{L}$ be a leakage profile function, which takes the inverted index (in plaintext) and the encrypted queries as input, and outputs the leakage information of the whole system. For a security parameter λ (roughly related to the number of bits in the key), for two efficient algorithms A (for Adversary) and S (for Simulator), Experiments $\mathbf{Real}^{\Pi}_{A}(\lambda)$ and $\mathbf{Ideal}^{\Pi}_{A,S}(\lambda)$ are defined as follows:

- $\mathbf{Real}^{\Pi}_{A}(\lambda)$: $A(1^{\lambda})$ chooses the plaintext inverted index DB and a list of queries $\mathbf{q}$. The experiment runs $(K, \mathsf{EDB}) \leftarrow \mathsf{EDBSetup}(\mathsf{DB})$, where K is the secret set of keys only known to the client, and EDB is the encrypted index. For each $i \in \{1, \cdots, |\mathbf{q}|\}$, it runs the algorithm Search with client input $(K, \mathbf{q}[i])$ and server input EDB and stores the transcript (i.e., all the communications between the client and server) in $\mathbf{t}[i]$. Then the experiment gives EDB and transcript $\mathbf{t}$ to A, which returns a Boolean value $b \in \{0, 1\}$ answering whether t is produced by real search protocol or simulator. Finally the experiment outputs b.
- $\mathbf{Ideal}^{\Pi}_{A,S}(\lambda)$: $A(1^{\lambda})$ chooses DB and a list of queries $\mathbf{q}$. The experiment runs $S(\mathcal{L}(\mathsf{DB}, \mathbf{q}))$ and gives its output to A, which returns a Boolean $b \in \{0, 1\}$ answering whether this output is produced by the real search protocol or simulator. Finally the experiment outputs b.

Definition Π is $\mathcal{L}$-semantically-secure against non-adaptive attacks if for any efficient adversary A there exists an algorithm S such that the probability difference $\Pr[\mathbf{Real}^{\Pi}_{A}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{\Pi}_{A,S}(\lambda) = 1]$ is negligible, i.e., asymptotically less than λ^{-c} for any integer c.

A key point in the above framework is that attacker A is unable to tell apart the communications of the real secure protocol and the communications between the client and the simulator S. This notion is only for non-adaptive attacks, since in each experiment the attacker can only choose the queries at the beginning, and can never change the chosen queries during the interaction with the server. This assumption fits most scenarios since often a third-party attacker has little control over the queries that the search users would like to ask. The only thing the attacker can know is the distribution of the queries. With the above definition and standard cryptographic assumptions, we can show the leakage profile of QDT as follows.

Leakage profile of QDT The QDT document retrieval scheme specified in Sect. 3 is semantically secure against non-adaptive attackers with the following leakage profile (i.e., the information leaked to the attacker):

- (1) The total number of searchable terms in an index and the length of each posting list after padding;

- (2) The number of terms for each query, and the sharing of terms among queries;
- (3) The patterns of accessing posting lists for searched terms and the number of documents that match each query;
- (4) The occurrence pattern of the produced group tags, and the encoded member IDs of documents in posting lists;
- (5) The encoded features for encrypted document IDs.

To validate the above leakage profile, we can follow the analysis in Cash et al. (2013), Kamara and Moataz (2017) that the data that any attacker (server) can acquire during the execution of the real search protocol cannot be distinguished from the data output by a simulation algorithm which only takes as input the above leakage profile of QDT.

The previous work in OXT/IEX or their extensions (Cash et al., 2013; Kamara & Moataz, 2017; Agun et al., 2018; Lai et al., 2018) also leaks items (1), (2) and (3). Item (5) is necessary as we have to support ranking. Item (4) is introduced by QDT, and Property 2 in Appendix 2 addresses the leakage of member IDs. The guided padding in Sect. 3.4 thwarts a related attack.

Acknowledgements We thank Stefano Tessaro for his critical advice and help, and the anonymous referees for their valuable comments. This work was supported in part by U.S. National Science Foundation (NSF) 2040146/2225942 and by a Google faculty research award. It has used the computing resource of the XSEDE and ACCESS programs supported by NSF. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these sponsors.

Author Contributions The authors have contributed to design, implementation, evaluation, and/or writing. They have read and approved the final manuscript.

Funding This work was supported in part by U.S. National Science Foundation (NSF) and by a Google faculty research award.

Availability of data and materials Related code will be made available from the project website.

Declarations

Conflict of interest Not applicable.

Ethical approval Not applicable.

References

Agun, D., Shao, J., Ji, S., Tessaro, S., & Yang, T. (2018). Privacy and efficiency tradeoffs for multiword top k search with linear additive rank scoring. In *Proceedings of the 2018 world wide web conference)* (pp. 1725–1734). International World Wide Web Conferences Steering Committee.

Ahmad, W.U., Chang, K.-W., & Wang, H. (2018). Intent-aware query obfuscation for privacy protection in personalized web search. In *The 41st international ACM SIGIR conference on research & development in information retrieval* (pp. 285–294). ACM.

Ahmad, W.U., Rahman, M.M., & Wang, H. (2016). Topic model based privacy protection in personalized web search. In *Proceedings of the 39th international ACM SIGIR conference on research and development in information retrieval* (pp. 1025–1028). ACM.

Baeza-Yates, R., & Ribeiro-Neto, B. (2011). *Modern information retrieval* (2nd ed.). New Jersey: Addison Wesley.

Barker, E., Roginsky, L.C.A., Vassilev, A., & Davis, R. (2018). Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography. NIST, US Department of Commerce. https://doi.org/10.6028/NIST.SP.800-56Ar3

Blackstone, L., Kamara, S., & Moataz, T. (2019). Revisiting leakage abuse attacks. *Cryptology ePrint Archive,*

Boneh, D., & Shoup, V. (2015). A graduate course in applied cryptography. Draft 0.2

Boneh, D., & Shoup, V. (2023). A graduate course in applied cryptography. https://toc.cryptobook.us. Online.

Boneh, D., & Shparlinski, I.E. (2001). On the unpredictability of bits of the elliptic curve diffie-hellman scheme. In *Annual international cryptology conference* (pp. 201–212). Springer.

Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S., & Sadeghi, A.-R. (2017). Software grand exposure: Sgx cache attacks are practical. In *11th USENIX workshop on offensive technologies (WOOT 17).*

Broder, A.Z., Carmel, D., Herscovici, M., & Soffer, A., Zien, J. (2003) Efficient query evaluation using a two-level retrieval process. In *Proceedings of the twelfth international conference on information and knowledge management* (pp. 426–434).

Cao, N., Wang, C., Li, M., Ren, K., & Lou, W. (2014). Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Transactions on Parallel and Distributed Systems, 25*(1), 222–233.

Cash, D., Grubbs, P., Perry, J., & Ristenpart, T. (2015). Leakage-abuse attacks against searchable encryption. In *CCS'15* (pp. 668–679). ACM.

Cash, D., Jaeger, J., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.-C., & Steiner, M. (2014) Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS* (Vol. 14, pp. 23–26). Citeseer.

Cash, D., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M., & Steiner, M. (2013) Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO 2013* (pp. 353–373).

Chase, M., Gilad-Bachrach, R., Laine1, K., Lauter, K., & Rinda, P. (2017) Private collaborative neural network learning. https://eprint.iacr.org/2017/762.pdf

Chen, G., Lai, T.-H., Reiter, M.K., & Zhang, Y. (2018). Differentially private access patterns for searchable symmetric encryption. In *IEEE INFOCOM 2018 - IEEE conference on computer communications* (pp. 810–818). https://doi.org/10.1109/INFOCOM.2018.8486381

Costan, V., & Devadas, S. (2016). Intel sgx explained. *IACR Cryptology ePrint Archive, 2016*, 86.

Culpepper, J. S., & Moffat, A. (2010). Efficient set intersection for inverted indexing. *ACM Transaction Information System, 29*(1), 1–1125.

Curtmola, R., Garay, J., Kamara, S., & Ostrovsky, R. (2006). Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM CCS* (pp. 79–88).

Dai, Z., Xiong, C., Callan, J., & Liu, Z. (2018). Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proceedings of the eleventh ACM international conference on web search and data mining* (pp. 126–134). ACM.

Di Castro, D., Lewin-Eytan, L., Maarek, Y., Wolff, R., & Zohar, E. (2016). Enforcing k-anonymity in web mail auditing. In *Proceedings of the ninth ACM international conference on web search and data mining* (pp. 327–336). ACM.

Ding, S., & Suel, T. (2011). Faster top-k document retrieval using block-max indexes. In *Proceedings of the 34th international ACM SIGIR conference on research and development in information retrieval* (pp. 993–1002).

Dworkin, M. (2001) Recommendation for block cipher modes of operation. methods and techniques. Technical report, National Inst of Standards and Technology Gaithersburg MD Computer security Div

Dworkin, M. J. (2015). Sha-3 standard: Permutation-based hash and extendable-output functions. *Technical Report*.

Formal, T., Lassance, C., Piwowarski, B., & Clinchant, S. (2021) Splade v2: Sparse lexical and expansion model for information retrieval. SIGIR 2021. ArXiv:2109.10086

Formal, T., Lassance, C., Piwowarski, B., & Clinchant, S. (2022). From distillation to hard negative sampling: Making sparse neural IR models more effective. In *SIGIR*.

Garg, S., Mohassel, P., & Papamanthou, C. (2015). Tworam: Round-optimal oblivious ram with applications to searchable encryption. *IACR Cryptol. ePrint Arch., 2015*, 1010.

Gentry, C. (2009) Fully homomorphic encryption using ideal lattices. In *ACM STOC '09* (pp. 169–178).

Gillmor, D.K. (2016). Negotiated finite field Diffie-Hellman ephemeral parameters for transport layer security (TLS). RFC Editor, https://doi.org/10.17487/RFC7919.

Guo, J., Fan, Y., Ai, Q., & Croft, W.B. (2016). A deep relevance matching model for ad-hoc retrieval. In *Proceedings of CIKM'16* (pp. 55–64). ACM.

Habernal, I. (2022). How reparametrization trick broke differentially-private text representation learning. In *Proceedings of the 60th annual meeting of the association for computational linguistics* (Volume 2: Short Papers, pp. 771–777). Association for Computational Linguistics, Dublin, Ireland. https://doi.org/10.18653/v1/2022.acl-short.87

Hacigümüş, H., Iyer, B., Li, C., & Mehrotra, S. (2002) Executing sql over encrypted data in the database-service-provider model. In *SIGMOD '02* (pp. 216–227). ACM.

Hoang, T., Ozmen, M. O., Jang, Y., & Yavuz, A. A. (2019). Hardware-supported oram in effect: Practical oblivious search and update on very large dataset. *Proceedings on Privacy Enhancing Technologies, 2019*(1), 172–191.

Hu, H., Xu, J., Ren, C., & Choi, B. (2011). Processing private queries over untrusted data cloud through privacy homomorphism. In *ICDE* (pp. 601–612).

Islam, M.S., Kuzu, M., & Kantarcioglu, M. (2012) Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS 2012*.

Jagannathan, G., Pillaipakkamnatt, K., & Wright, R.N. (2009). A practical differentially private random decision tree classifier. In *2009 IEEE international conference on data mining workshops* (pp. 114–121). IEEE.

Jamie Callan's research group. (2020). The ClueWeb09 Dataset. http://boston.lti.cs.cmu.edu/Data/clueweb09. Carnegie Mellon University's Language Technologies Institute.

Järvelin, K., & Kekäläinen, J. (2002). Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS), 20*(4), 422–446.

Ji, S., Shao, J., Agun, D., & Yang, T. (2018). Privacy-aware ranking with tree ensembles on the cloud. In *The 41st international ACM SIGIR conference on research & development in information retrieval* (pp. 315–324). ACM.

Jones, K.S., Walker, S., & Robertson, S.E. (2000). A probabilistic model of information retrieval: Development and comparative experiments. In *Information processing and management* (pp. 779–840).

Kamara, S., & Moataz, T. (2017). Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Annual international conference on the theory and applications of cryptographic techniques* (pp. 94–124). Springer.

Kamara, S., & Moataz, T. (2019). Computationally volume-hiding structured encryption. In *Annual international conference on the theory and applications of cryptographic techniques* (pp. 183–213). Springer.

Kamara, S., Papamanthou, C., & Roeder, T. (2012). Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on computer and communications security* (pp. 965–976). ACM.

Khattab, O., & Zaharia, M. A. (2020). Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *SIGIR*.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

Kojo, M., & Kivinen, T. (2003). More Modular Exponential (MODP) Diffie-Hellman groups for internet key exchange (IKE). RFC Editor. https://doi.org/10.17487/RFC3526.

Lai, S., Patranabis, S., Sakzad, A., Liu, J.K., Mukhopadhyay, D., Steinfeld, R., Sun, S.-F., Liu, D., & Zuo, C. (2018) Result pattern hiding searchable encryption for conjunctive queries. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. CCS '18* (pp. 745–762). ACM.

Li, M., Lin, S.-C., Ma, X., & Lin, J. (2023). SLIM: Sparsified late interaction for multi-vector retrieval with inverted indexes. In *SIGIR*.

Lin, J.J., & Ma, X. (2021) A few brief notes on deepimpact, coil, and a conceptual framework for information retrieval techniques. ArXiv:2106.14807

Lin, J., Nogueira, R., & Yates, A. (2020) Pretrained transformers for text ranking: Bert and beyond. ArXiv: 2010.06467

Lipmaa, H., Rogaway, P., & Wagner, D. (2000) Ctr-mode encryption. In *First NIST workshop on modes of operation* (vol. 39).

Liu, X., Li, Q., Li, T., & Chen, D. (2017). Differentially private classification with decision tree ensemble. *Applied Soft Computing*.

MacAvaney, S., Nardini, F., Perego, R., Tonellotto, N., Goharian, N., & Frieder, O. (2020). Efficient document re-ranking for transformers by precomputing term representations. In *SIGIR*.

Mallia, A., Khattab, O., Suel, T., & Tonellotto, N. (2021). Learning passage impacts for inverted indexes. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval* (pp. 1723–1727).

Mallia, A., Ottaviano, G., Porciani, E., Tonellotto, N., & Venturini, R. (2017). Faster blockmax wand with variable-sized blocks. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval* (pp. 625–634).

Matveeva, I., Burges, C., Burkard, T., Laucius, A., & Wong, L. (2006). High accuracy retrieval with multiple nested ranker. In *Proceedings of SIGIR'06* (pp. 437–444). ACM.

Mishra, P., Poddar, R., Chen, J., Chiesa, A., & Popa, R.A. (2018) Oblix: An efficient oblivious search index. In *2018 IEEE symposium on security and privacy (SP)* (pp. 279–296).

Naveed, M., Prabhakaran, M., & Gunter, C.A. (2014). Dynamic searchable encryption via blind storage. In *2014 IEEE symposium on security and privacy* (pp. 639–654). IEEE.

Paillier, P. (1999) Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT '99* (pp. 223–238).

Patel, S., Persiano, G., Yeo, K., & Yung, M. (2019) Mitigating leakage in secure cloud-hosted data structures: volume-hiding for multi-maps via hashing. CCS 2019.

Patranabis, S., & Mukhopadhyay, D. (2021) Forward and backward private conjunctive searchable symmetric encryption. In *Proceedings of NDSS* 2021.

Pouliot, D., & Wright, C.V. (2016). The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In *CCS'16* (pp. 1341–1352). ACM.

Qiao, Y., Yang, Y., He, S., & Yang, T. (2023). Representation sparsification with hybrid thresholding for fast splade-based document retrieval. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval. SIGIR '23* (pp. 2329–2333). Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3539618.3592051

Ryoo, M.S., Rothrock, B., Fleming, C., & Yang, H.J. (2017) Privacy-preserving human activity recognition from extreme low resolution. In *Proceedings of AAAI'17* (pp. 4255–4262).

Shang, Z., Oya, S., Peter, A., & Kerschbaum, F. (2021). Obfuscated access and search patterns in searchable encryption. In *Proceedings of NDSS 2021*. https://doi.org/10.14722/ndss.2021.23041.

Shao, J., Ji, S., & Yang, T. (2019). Privacy-aware document ranking with neural signals. In *The 42st international ACM SIGIR conference on research & development in information retrieval*. ACM.

Shao, J., Ji, S., Glova, A.O., Qiao, Y., Yang, T., & Sherwood, T. (2020). Index obfuscation for oblivious document retrieval in a trusted execution environment. In *Proceedings of the 29th ACM international conference on information and knowledge management. CIKM '20* (pp. 1345–1354). Association for Computing Machinery, New York, NY, USA https://doi.org/10.1145/3340531.3412035

Shao, J., Qiao, Y., Ji, S., & Yang, T. (2021) Window navigation with adaptive probing for executing blockmax wand. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval* (pp. 2323–2327).

Song, D. X., Wagner, D., & Perrig, A. (2000). Practical techniques for searches on encrypted data. *IEEE SP,* '00.

Strohman, T., Metzler, D., Turtle, H., & Croft, W.B. (2005). Indri: A language model-based search engine for complex queries. In *Proceedings of the international conference on intelligent analysis* (Vol. 2, pp. 2–6). Amherst, MA, USA.

Sun, W., Zhang, R., Lou, W., & Hou, Y.T. (2018). Rearguard: Secure keyword search using trusted hardware. In *IEEE INFOCOM 2018-IEEE conference on computer communications* (pp. 801–809). IEEE.

Sun, W., Wang, B., Cao, N., Li, M., Lou, W., Hou, Y. T., & Li, H. (2014). Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. *IEEE Transactions on Parallel and Distributed Systems, 25*(11), 3025–3035.

Sweeney, L. (2002). k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 10*(05), 557–570.

Thakur, N., Wang, K., Gurevych, I., & Lin, J. (2023) Sprint: A unified toolkit for evaluating and demystifying zero-shot neural sparse retrieval. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval. SIGIR '23* (pp. 2964–2974). Association for Computing Machinery, New York, NY, USA.

Vo, V., Lai, S., Yuan, X., Nepal, S., & Liu, J.K. (2021) Towards efficient and strong backward private searchable encryption with secure enclaves. In Sako, K., & Tippenhauer, N.O. (eds.) *Applied cryptography and network security* (pp. 50–75).

Wang, L., Lin, J., & Metzler, D. (2011) A cascade ranking model for efficient ranked retrieval. In *ACM SIGIR* (pp. 105–114).

Wang, G., Liu, C., Dong, Y., Choo, K.-K.R., Han, P., Pan, H., & Fang, B. (2018). Leakage models and inference attacks on searchable encryption for cyber-physical social systems. *IEEE Access, 6*, 21828–21839.

Xia, Z., Wang, X., Sun, X., & Wang, Q. (2016). A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE Transactions on Parallel and Distributed Systems, 27*(2), 340–352.

Xiong, C., Dai, Z., Callan, J., Liu, Z., & Power, R. (2017). End-to-end neural ad-hoc ranking with kernel pooling. In *SIGIR* (pp. 55–64). ACM.

Xu, Y., Cui, W., & Peinado, M. (2015). Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE symposium on security and privacy* (pp. 640–656). IEEE.

Xu, M., Namavari, A., Cash, D., & Ristenpart, T. (2021). Searching encrypted data with size-locked indexes. In Bailey, M., & Greenstadt, R. (eds.) *30th USENIX security symposium, USENIX security 2021* (August 11-13, pp. 4025–4042). USENIX Association.

Xu, C., Wang, R., Zhu, L., Zhang, C., Lu, R., & Sharif, K. (2023). Efficient strong privacy-preserving conjunctive keyword search over encrypted cloud data. *IEEE Transactions on Big Data, 9*(03), 805–817. https://doi.org/10.1109/TBDATA.2022.3205668

Yang, H., & Soboroff, I. (2015). Privacy-preserving ir 2015: When information retrieval meets privacy and security. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval. SIGIR '15* (pp. 1157–1158). ACM, New York, NY, USA https://doi.org/10.1145/2766462.2767857

Yang, Y., Qiao, Y., Shao, J., Yan, X., & Yang, T. (2022) Lightweight composite re-ranking for efficient keyword search with BERT. In: *Proceedings of the fifteenth ACM international conference on web search and data mining. WSDM '22* (pp. 1234–1244). ACM, New York, NY, USA.

Yang, H., Soboroff, I., Xiong, L., Clarke, C.L.A., & Garfinkel, S.L. (2016). Privacy-preserving ir 2016: Differential privacy, search, and social media. In *Proceedings of the 39th international ACM SIGIR conference on research and development in information retrieval. SIGIR '16* (pp. 1247–1248). ACM, New York, NY, USA https://doi.org/10.1145/2911451.2917763