Index Obfuscation for Oblivious Document Retrieval in a Trusted Execution Environment

Jinjin Shao, Shiyu Ji, Alvin Oliver Glova, Yifan Qiao, Tao Yang, Tim Sherwood
Department of Computer Science, University of California
Santa Barbara, California, USA

ABSTRACT

This paper studies privacy-aware inverted index design and document retrieval for multi-keyword document search in a trusted hardware execution environment such as Intel SGX. The previous work uses time-consuming oblivious computing techniques to avoid the leakage of memory access patterns for privacy preservations in such an environment. This paper proposes an efficiency-enhanced design that obfuscates the inverted index structure with posting bucketing and document ID masking, which aims to hide document-term association and avoid the access pattern leakage. This paper describes privacy-aware oblivious document retrieval during online query processing based on such an index. Both privacy and efficiency analyses are provided, followed by evaluation results comparing proposed designs with multiple baselines.

CCS CONCEPTS

• Information systems → Retrieval efficiency; **• Security and privacy → Privacy-preserving protocols**; **Management and querying of encrypted data**.

ACM Reference Format:
Jinjin Shao, Shiyu Ji, Alvin Oliver Glova, Yifan Qiao, Tao Yang, Tim Sherwood. 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), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3412035

1 INTRODUCTION

Privacy protection has become increasingly important for cloud-based information services including keyword search. Searchable encryption [8, 11, 22, 34] has been developed to support conjunctive or disjunctive search without considering ranking. Secure linear additive ranking based matrix transformation is studied in [5, 37, 41] and such techniques based on matrix transformation is only applicable for small datasets while the solution in [1] still requires a significant amount of post-ranking conducted at the client side with partial server-side ranking, which is not suitable for a large dataset. Nonlinear ranking privacy-aware solutions are studied in [20, 33], where the number of results to be ranked is small. None of the previous work has solved the problem of privacy-preserving document retrieval with additive ranking in dealing with a large dataset. This is a critical problem to address for document search because popular solutions for searching a large-scale dataset typically involves a multi-stage ranking scheme [28, 40].

The previous work on the above searchable encryption work still cannot reach a fully secured level, because it leaks certain statistical query patterns, which can yield leakage-abuse attacks [6, 19, 25]. Another challenge is that ranking involves query-dependent feature calculations. Hiding feature information through encryption prevents the server from performing effective scoring and result comparison. On the other hand, unencrypted feature values can lend themselves to privacy attacks [6, 19]. Homomorphic encryption techniques [15, 32] have been offered to secure data while letting the server perform arithmetic calculations without decrypting the underlying data, and have been optimized for some practical usage [27]. But such a scheme is still not computationally feasible for large applications where a large number of slow multiplication operations are involved, resulting in orders of magnitude slowdown.

All of the above solutions use traditional hardware platforms. Given the difficulty faced in the previous work, this paper studies the usage of trusted hardware technology. Recently, hardware-based Trusted Execution Environments (TEE), such as Intel Software Guard Extensions (SGX) and ARM TrustZone, have emerged as options for secure computation [14, 17, 31] even if the hosting operating system may still curiously seek for information. A TEE provides a secure and isolated space where an application can run protected operations that deal with sensitive data. The OS of a server is blocked to inspect computations and data inside this TEE. Even though a TEE can provide a reasonably-trustable computing environment, a server can still observe the access traffic patterns of its TEE to the main memory and to the TEE's internal buffer during server-side query processing. Various memory side-channel attacks with such hardware have been found (e.g. [3]). Data-dependent memory access pattern leakage can facilitate statistical privacy attacks for document search [6, 19]. To protect the leakage of data-dependent memory access patterns for keyword matching of documents, ORAM-based search solutions are proposed in [17, 29], but it is not computationally practical if the number of returned results is large. For example, single-keyword search on a posting list with 1000 documents can take around 100 seconds [17]. Another solution called REARGUARD [38] for single-word queries is proposed with oblivious memory accessing, in which a TEE intentionally scans a large number of posting lists to obfuscate access patterns, even if accessing only one of lists is needed. Oblivious computing in this solution also incurs time-consuming query processing cost.

The contribution of this paper is to propose inverted index obfuscation and a relatively efficient oblivious document retrieval solution with additive ranking by leveraging trusted hardware. Our

CIKM '20, October 19–23, 2020, Virtual Event, Ireland
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412035

evaluation results with analyses show that our scheme is much faster than REARGUARD, though it is slower than the traditional top K document retrieval based on the WAND and BMW optimization without privacy constraint [4, 13]. But these algorithms are vulnerable to privacy attacks, which we present in Section 3. To preserve privacy, our algorithm pursues the oblivious but exhaustive strategy, which represents a trade-off of efficiency for privacy.

2 BACKGROUND AND RELATED WORK

2.1 Inverted index and document retrieval

An *inverted index* contains a set of terms, and a *document posting list* of each term represents documents that contain such a term. Each term can be a single word used in a document or can represent other ranking features needed for a search system. Given a set of query terms that correspond to a query, the goal of the document retrieval problem is to find a set of documents that contain all query terms following the conjunctive query semantics, or at least one of these terms following the disjunctive query semantics. This paper is focused on the disjunctive query semantic while the proposed techniques are extensible for conjunctive queries.

The document retrieval problem is often associated with *top K search* where only the top ranked results are selected. Computing ranking scores during document retrieval provides an opportunity to skip some low-score documents and speedup search. *Ranking* for document retrieval is typically simple with a term-document based additive formula (e.g. BM25 [21]) which is $\sum_{t \in Q} S(t, d)$, where Q is the set of all search terms, and $S(t, d)$ is a weighted term impact score for this document d. To support such ranking, the posting record of each term contains not only an document ID but also its impact score associated with such a term. The document retrieval scheme typically returns matched results to another search system component where more advanced schemes can be applied to re-rank after the top K results are selected in this stage.

The previous top K search studies have advocated earlier termination strategies for document retrieval to skip low-score documents, which cannot be on top K, during the traversal of posting lists [4, 36]. The latest well-known scheme, BMW [13] and its variation [26], use the block-based document skipping technique based on the WAND algorithm [4]. In such schemes, the runtime index visitation order follows document-at-a-time (DAAT) and postings are pre-ordered by their document IDs. Ordering postings by document IDs also facilitates effective data compression. Another index organization strategy is impact-layered index where postings are divided by layers. Impact scores of documents in a lower layer are higher than that of lower layers, and postings within each layer is sorted by document IDs. For simplicity, this paper assumes the document-sorted index.

2.2 Threat model

There are three entities in a cloud system: data owner, search user (client) and cloud server. The data owner has a collection of documents to be outsourced by the cloud server. To enable the searching and ranking functionality, the data owner needs to encrypt the documents, the inverted index, and the ranking model for the cloud server outsourcing. This proposal will initially assume that the data owner and the search user are the same entity. The client builds an encrypted but searchable index and lets a server host such index. The server is honest-but-curious, i.e., the server will honestly follow the client's protocol, but will also try to learn any private information from the client data based on what the server can observe from the hosted data and runtime information. To conduct a search query, the client sends several encrypted keywords and related information to the server. Our research targets at inexpensive client-server communication since extensive multi-round communication between the server and client (e.g. [18, 30]) incurs excessive high communication cost and response latency.

2.3 Known privacy attacks in document search

As the queries and documents are encrypted, the privacy abuse attacks can occur by acquiring statistical information to reveal search queries or documents partially. A server can always learn something by observing query processing. The question is if the learned information is sufficient for a server to recover original text words from encrypted index and content. The statistical patterns leaked during query processing can lead to the known attacks [6, 19, 25, 39] and we summarize them as follows.

- *Co-occurrence probability of search terms in a document* . Islam et al. [19] proposed the IKK plaintext attack to recover the plaintext of query words by computing the search word occurrence probability when knowing the result overlapping patterns of two single-word queries. The paper assumes that the adversary also knows the co-occurrence probability of dictionary words used in the dataset (e.g. approximated from a public dataset) and a plaintext mapping from a small set of words to their IDs.
- *Document frequency of search terms.* This information can be derived after knowing the length of a posting list for a term. The work in [6] shows that after knowing co-occurrence of search words in a document and their document frequency, an adversary can recover the plaintext of search terms without knowing the mapping from a small subset of search words to their word IDs.
- *Query equality pattern.* When a server knows about the repetition ratio of search queries, it can compare with some known background knowledge on query popularity, and deduce plaintext for some of queries as shown in [25].
- *Document similarities.* There are also attacks exploiting leaked similarities among documents [39]. Their attacks only work if the adversary knows occurrence frequency and co-occurrence frequency of selected terms in the entire document set.

All of these attacks require the leakage of term co-occurrence and term frequency, and knowing document sharing patterns among posting lists, the posting list length, and the list access frequency can lead to the above statistical information leakage. We will consider countermeasures in our design. We will also study how a traditional document retrieval algorithm without a privacy protection can reveal the above statistical patterns.

Searchable encryption (e.g. [7, 8, 12, 22, 23]) has been proposed to identify documents that match a user query from the encrypted index. These schemes do not support ranking and they still suffer from some degree of information leakage, which could cause a leakage-abuse attack discussed above. For example, OXT [7, 8] and IEX scheme [22] are the most comprehensive schemes to support multi-keyword queries. IEX for conjunctive queries leaks the overlapping of search terms from one query to another, thus can leak the co-occurrence probability of search terms and posting list access patterns. OXT for conjunctive queries leaks the overlapping of some search terms (e.g. leading terms) in a query history.

As discussed in Section 1, the previous work in [5, 37, 41] for secure additive ranking based matrix transformation is only applicable for small datasets, while the solution in [1] does not support full server-side additive ranking. None of the previous work including nonlinear ranking solutions in [20, 33] has solved the problem of privacy protection in document retrieval with additive ranking in dealing with a large dataset. Also, these schemes use deterministic IDs for search terms, thus leak the query equality patterns.

2.4 Trusted hardware platform

We assume there is a server platform that incorporates a processor where applications can create and make use of protected memory regions in their trusted execution environment. For example, data center providers typically use SGX-equipped Intel processors. Microsoft currently provides Intel SGX extensions in its cloud platform. A TEE has its own memory space and a host processor can monitor all accesses to the TEE's memory space, knowing which addresses are visited by the code running inside the TEE, but it cannot view the content accessed by the TEE's code.

To thwart attacks based on the leakage of posting list access patterns, [38] designs REARGUARD, an oblivious matching algorithm for one-word queries as follows. First it pads all posting lists with encrypted useless records so that all lists have the same lengths. Second, with one query word, the server scans all posting lists one by one from the whole inverted index. When the TEE has scanned all posting lists, only matched one is retained and the server cannot differentiate which one is matched, corresponding to the search terms. To alleviate the huge time cost to scale all posting lists, [38] suggests to partition indexed terms into groups and a server only scans a group of posting lists instead of the entire index. This trade-off brings a leakage on keyword group access patterns. To be effective for privacy protection, the group size still needs to be large, thus the time cost is still expensive.

3 LEAKAGE-ABUSE ATTACKS AND DESIGN CONSIDERATIONS

We discuss possible leakage-abuse attacks during document retrieval with or without WAND-based document skipping [4], and present our design considerations in leveraging trusted hardware platforms. The main design challenge is that naively running document retrieval inside a TEE, even the index is encrypted, can leak unique query-dependent memory access traces, and an adversary server can take advantages and launch a leakage-abuse attack. We describe two attacks below: the first one reveals co-occurrence information and the second one reveals posting list access patterns. For the worst case, both attacks can recover plaintext queries from encrypted queries ([6] and [25] resp.).

Assumptions on adversary's knowledge. We assume the binary code of a program running inside TEE is known by a server adversary, and the server can observe all the memory accesses (namely, memory addresses that are read or written). Since some server can guess which instruction in TEE's code segment is read and executed, for the *worst case scenario*, we assume there exists an adversary who can trace program running step by step given a previously-issued client query. Hence an adversary can learn at least the number of search terms, and the number of the corresponding posting lists loaded to the TEE. With special obfuscation, the server can also learn the length of each posting list (the number of documents) approximately after decryption and decompression within the TEE.

We assume the run-time behavior of a document retrieval program is deterministic, only dependent on the input query terms. Also assume in a worst case, the binary code can be loaded into the TEE with the same starting memory address.

Attack revealing co-occurrence of search terms. We consider a standard document-at-a-time algorithm without using WAND, and it visits documents one by one in all posting lists in an increasing order of document IDs. The TEE program should have a code segment that adds the relevance score of matched postings from each posting list. The adversary can trace code execution and memory accesses within such a segment, and infer the intersection pattern of any two of the posting lists. Then the adversary can compute the co-occurrence probability of two search terms in a document as the number of common documents in two term lists divided by the total number of documents.

By now, the adversary knows the co-occurrence probability of encrypted search terms, and the term frequency based on the posting list length. The adversary can assume the dataset uses words from a dictionary (e.g. English), obtain the term co-occurrence probability based on the public knowledge, and recover the original plaintext of these encrypted search terms, following the work of [6, 19].

Attack revealing query equality patterns. For any document retrieval algorithm, there are two ways for a server to find the repeat probability of a query. One is to recognize the same set of search terms that has been used from one query to another. The other way is to observe and consider the memory address access sequence as a signature and detect if there is a repetition. With high probability, different queries will yield different memory access sequences. Then a server can use the derived repetition patterns to launch a query recovery attack following [25].

To minimize the chance of launching the above attacks, our countermeasure considerations are listed as follows.

- To hide document co-occurrence between posting lists, we can merge multiple terms into one large posting list. Hence the intersection size between two merged lists does not reflect any true co-occurrence information between two individual terms. Such merging essentially combines several terms together as a term bucket, which also hides the relationship of one-to-one mapping between index and query terms.
- To avoid leaking the signature of the memory address sequence, we pad the search results. For WAND/BMW based algorithms [4, 13, 26], it leaks a fixed memory address accessing sequence as a signature and one can infer the query repetition patterns. Initially, we considered using randomization of document skipping to yield different memory access patterns on the same input. However, given one query from the client, the server adversary can infer the statistical distribution of the memory access patterns by repeating such a query until the distribution can be inferred

with a high confidence. Namely, the server can still approximate query repetition patterns and detect the query repetition.
As a result, we decide not to use WAND-based document skipping as a trade-off of efficiency for privacy. That essentially follows an exhaustive search, which has a visible efficiency loss. We view this efficiency loss acceptable with today's CPU speed and will evaluate the response time in our experiments.

We will reflect the above countermeasures in our algorithm design.

4 INDEX BUCKETING AND MASKING

We propose *masked inverted index (MII)* with term bucketing to support efficient query processing, while avoiding the leakage of query equality pattern, co-occurrence probability pattern, and term frequency pattern. Our key ideas are: 1) To obfuscate the index, we group terms into buckets. Each bucket is linked to the union of posting lists of terms grouped in this bucket. The TEE of a server can conduct the term-bucket based posting list retrieval, and identify matched documents in an oblivious way to prevent the above-mentioned leakage; for example, the server cannot differentiate the identities of query terms. 2) To increase the degree of obfuscation, we duplicate each term and its posting list by k times, and randomly map duplicated copies to different term buckets. As a result, each term bucket contains mixed popular or unpopular terms, and the access pattern of each term bucket would be drastically different from that of original individual terms. Then, it is unlikely that a server can derive reasonably accurate query equality patterns and document frequency patterns. We let a TEE decrypt a posting record inside its trusted memory buffers to conduct protected query processing. Since the server is unable to observe the buffer content, it cannot detect co-occurrence probability patterns among term posting lists.

Table 1 lists frequently used symbols through this paper.

Table 1: Frequently used notations

Symbols	Explanations
K	Number of top ranked results needed for document retrieval
V	Vocabulary size, namely, number of searchable terms
q	Number of search terms in a query
k	Number of duplicate copies for each term
b	Number of term copies hosted in each bucket
B	Number of term buckets
m	Mask code for indicating term association in a bucket
u_i	Term bucket ID for the i-th query term
$m(u_i)$	Selector for the i-th term hosted in Bucket u_i
Enc	Symmetric encryption with a random seed, e.g., AES256
reg_i	The i-th register

4.1 Inverted index with term buckets

We organize the index as follows.

- The index contains a set of term buckets. These buckets along with their posting lists will be compressed first, and then encrypted. Each a term bucket can represent b terms: $t_1, t_2, \cdots, t_b$. A term bucket data structure is accessed by a bucket ID and its value is a sequence of posting records. Each posting record is denoted as (d, m, f) where d is the document ID, m is a masking bit code with h bits indicates which of the corresponding h terms own this posting, and f is a sequence of features for these terms in d. Document d belongs to the posting list of term t_i if and only if the rightmost i-bit of masking code m is 1. The f component only stores the document features for terms t_i whose term posting list has d. Postings in this term bucket are sorted in an increasing order of document IDs.
- Given V terms in an inverted index, we intend to duplicate each term k times and randomly map these kV terms to a set of buckets so that each bucket has b terms. Our goal in forming these buckets is to have a random distribution of these term copies among these buckets. As a result, such randomness enhances privacy protection. After a random mapping, there exist some buckets where multiple copies of the same term appear in the same bucket, and we call this mapping collisions. A large number of collisions reduces the effectiveness of randomness for privacy. In the next subsection, we describe a collision tolerance condition, and present a bucket building algorithm that reduces collisions until the tolerance condition is satisfied.
- The index building process will derive a mapping from each searchable term t to the bucket location of its k copies: $u_1, u_2, \cdots, u_k$. $map(t) = \{(u_1, m(u_1)), (u_2, m(u_2)), \cdots, (u_k, m(u_k))\}$ where for i from 1 to k, the $m(u_i)$-th term of bucket u_i represents term t. A client-side machine maintains such a map.

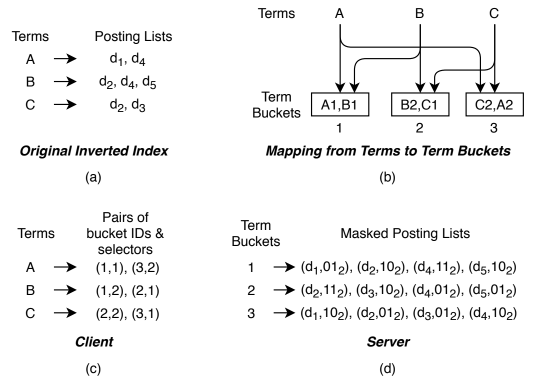

Figure 1: Masked inverted index with term buckets

Figure 1 illustrates a masked inverted index in Part (d) derived from an original inverted index in Part (a). In Fig. 1(a), the posting list of term A contains documents d_1 and d_4. In Fig. 1(b), each term is duplicated twice. For example, A has two copies $A1$ and $A2$. The copies of these terms are mapped to 3 buckets. Bucket 1 contains $A1$ and $B1$. Fig. 1(c) is the client-side map based on the bucket layout from Fig. 1(b). "$A \longrightarrow (1, 1)(3, 2)$" means that term A has one copy at Bucket 1 as the first term with a term selector 1, and another copy at Bucket 3 as the second term with a selector 2. Fig. 1(d) is the server-side collection of term buckets. Term bucket 1 has a posting list of four documents, where each document is associated with a mask code. For example, d_4 has binary mask 11_2, and is retrievable with a selector 1 or 2, while d_5 is only retrievable with a selector 2. For simplicity, Figure 1 does not include the feature sequence in each posting record, which is discussed later in this section.

4.2 Term bucketing with limited collisions

Ideally speaking, a term t is duplicated to k copies and they are mapped to k distinct term buckets, and each term bucket has b

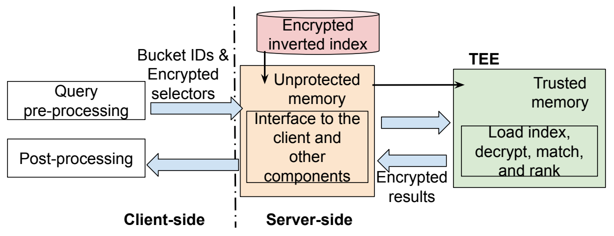

Figure 2: Online document retrieval with TEE

distinct terms. As a result, search term t can be obfuscated by the other $(k \cdot b - 1)$ unique terms. That was the goal of our design. However, a randomized mapping process most likely fails to achieve the above optimal situation due to mapping collisions, thus as a compromise, we design a near-optimal solution.

Our term bucket building algorithm performs as follows. Let S be a sequence of combining the k duplicates of all terms $S = \{(i, j) : 1 \le i \le V, 1 \le j \le k\}$ where tuple (i, j) represents the j-th copy of the i-th term. We add fake terms if necessary to make the total number of term copies divisible by b and pad these fake documents at the end of S. We randomly shuffle the tuples in S, then group b consecutive tuples.

We define a collision pair as two copies of the same term are mapped to the same bucket. If the collision tolerance condition, defined below, is not satisfied in the current mapping, we will restart another random shuffle of tuples in S, and regroup them again. We repeat this random mapping until the collision condition is satisfied.

Collision tolerance condition: After executing a randomized mapping from term copies to buckets, let each term t be duplicated to the following k buckets: $u_1, u_2, \cdots$, and u_k. These buckets satisfy both of the following constraints: 1) The set of $\{u_1, u_2, \cdots, u_k\}$ has at least $(k - 1)$ unique bucket IDs. 2) For each term bucket in $\{u_1, u_2, \cdots, u_k\}$, it has at least $(b - 1)$ unique term copies from different terms, among all b term copies.

Let z be the number of times required to conduct remapping until the above collision tolerance condition is met. From Theorem A.3 in the appendix, the probability of having z iterations is $(\frac{kb^2}{V}(\frac{1}{3} + \frac{k+b}{8}))^z$. For the datasets we tested, V is over 0.5 million, and kb^2 is small, so the probability of $z \ge 3$ is very small. Therefore, the random re-mapping process stops after a few iterations.

4.3 Oblivious online document retrieval

Figure 2 illustrates the client-server interaction with the presence of a TEE in a host server. Algorithm 1 is a description of our oblivious document retrieval scheme with additive rank score calculations for a disjunctive query. Given a query which corresponds to a number of search terms, a client first performs a map lookup for each term t as follows. Given $map(t) = \{(u_1, m(u_1)), (u_2, m(u_2)), \cdots, (u_k, m(u_k))\}$, the client randomly selects one of these k tuples, say, $(u_i, m(u_i))$, and then sends its encrypted form to the server.

Once the server receives all term bucket IDs and encrypted term selectors, its TEE loads encrypted posting lists for those term buckets to the trusted buffer space of the TEE, decrypts and decompresses them. For example, given $(u_i, Enc(m(u_i)))$, the TEE loads encrypted list for u_i, and decompresses the list after decryption. Then the TEE accesses each posting record of bucket u_i, say, (d, m, f). Assume $m(u_i)$ is the current term selector, the TEE considers d as a candidate when $m \& (1 << (m(u_i) - 1)) \ne 0$, where "&" is with a bit-wise *AND*. Then Algorithm 1 extracts a feature score from f obliviously using Algorithm 2.

For the example in Fig. 1, with a query including a keyword "A", a client can randomly choose and send term bucket ID 3 and a term selector 2 to the server. On the server side, a TEE loads, decrypts, and decompresses the posting list of bucket 3. For document d_1 with binary mask 10_2, it matches the term selector 2, and thus d_1 could be a candidate for the final result list.

Algorithm 1: Oblivious top K retrieval for MII in a TEE

```
Input: An encrypted query that contains $q$ terms. The $i$-th query
       term is represented by a bucket ID and an encrypted term
       selector: $(u_i, Enc(m_i))$
For each $i$-th term where $1 \le i \le q$, use $u_i$ to locate and load the
  encrypted and compressed bucket posting list and decrypt term
  selector to get $m_i$;
Let $\mathbf{L}$ be the list of posting lists of term buckets after decryption and
  decompression such that $\mathbf{L}[i]$ is the posting list of the term bucket
  corresponding to the $i$-th query term;
while there are documents left in $\mathbf{L}$ do
    Let $d$ be the next document with minimal id in $\mathbf{L}$, and advance the
      pointer of $d$ in each list of $\mathbf{L}$ where $d$ appears;
    Let $D$ be the set of indices $i$'s such that for each $i$, $d$ appears in
      the posting list $\mathbf{L}[i]$;
    The relevance score for $d$ in $\mathbf{L}[i]$ where $i \in D$, i.e., $s_i$, can be
      obliviously obtained through Algorithm 2;
    The ranking score for $d$ is $\sum_{i \in D} s_i$;
end
Use oblivious sorting [43] to find the top $K$ ranked documents;
Return the encrypted top $K$ document IDs and rank information;
```

Oblivious feature extraction. As discussed in Section 4.1, the third component, f, of each posting record (d, m, f) contains a sequence of rank score feature for document d corresponding to different terms specified by bit mask m of size b. To prove the oblivious property of Algorithm 1 in Lemma 5.1, the feature score fetch procedure is required to be oblivious with respect to different term selectors. Given the number of bits in mask m is b, a naive oblivious method is to store all b features for each bucket posting record for which a special value is used for an invalid feature since some of bits in m can be zero. However, this approach requires higher storage cost given there are many 0 bits in term masks.

We devise a space-conscious oblivious method, without storing invalid features. We let f store a sequence of valid features following the non-zero pattern in the bit mask m, and design a bit manipulation procedure in Algorithm 2 which locates the position of the corresponding feature in f given a term selector. The complexity of Algorithm 2 is linear to the number of bits in m which is b. Given any document in any bucket, Algorithm 2 is oblivious to any term selector for this term bucket. Namely, Algorithm 2 gives an identical access pattern for different term selectors from 1 to b.

Example of Algorithm 2. Assume a posting record is (d, m, f) where mask $m = 0101_2$. Assume the input term selector is 3, which means to select the third bit starting the rightmost. The feature list f is $[f_1, f_3]$. Algorithm 2 first finds the feature offset $ind = 2$, since there are 2 ones among the first 3 bits from the right side. Then the extracted feature is $s = 1 \cdot f_3 \cdot 1 + 0 \cdot f_1 \cdot 1 = f_3$. If the term selector is 2, then $ind = 1$, and $s = 0 \cdot f_1 \cdot 0 + 1 \cdot f_3 \cdot 0 = 0$ implying no feature is selected.

Algorithm 2: Oblivious Feature Extraction for MII

```
Input: A term selector $m(u)$, ranging from 1 to $b$ (both inclusively).
       A mask code $m$ with $b$ bits, and a list of all valid features $f$ for
       the current bucket posting record.
$ind \leftarrow 0$;      ▷ Count # bit 1 before the $m(u)$-th bit in $m$
for $c$ from 1 to $b$ do
    Let $maskBit$ be the $c$-th bit of $m$;
    $reg_1 \leftarrow c - m(u) - 1$;
    Let $signBit$ be the sign bit of $reg_1$, namely, 1 iff $reg_1 < 0$;
    $ind \leftarrow ind + (maskBit \& signBit)$, where & is bit-wise AND;
end
Let $test$ be the $m(u)$-th bit of $m$;
The selected feature score can be obliviously computed as
  $s \leftarrow \sum_{j=1}^{f.size}$ Equal$(ind, j) \cdot f[j] \cdot test$;
Return extracted feature score $s$;

Function Equal($a$, $b$):
    Store $a$ to a register $reg_1$, and $b$ to a register $reg_2$;
    $reg_3 \leftarrow reg_1 \oplus reg_2$, where $\oplus$ is bit-wise logical XOR;
    Store logical NOR of the bits in $reg_3$ to $reg_4$;
    return $reg_4$;
```

5 PRIVACY ANALYSIS

5.1 Obliviousness of MII

Definition 5.1. Memory access pattern and obliviousness [16, 35]. Memory access pattern is the sequence of memory accesses during the lifetime of algorithm execution. If the accessed memory is within TEE, each memory access contains two pieces of information: 1) access type (namely, read or write), and 2) memory address. TEE hides its memory content from the server. A document retrieval scheme is oblivious over an input query set if for any two queries from this query set, the memory access patterns are identical (for a deterministic algorithm) or identically distributed (for a randomized algorithm).

Note that our main algorithm (Algorithm 1) is deterministic. Hence we need to show that our algorithm gives identical memory access patterns over some queries. If one works with a randomized algorithm, then to be oblivious, this algorithm needs to preserve the distribution of memory access patterns.

Lemma 5.1. *Replacing any queried term with any term in the same bucket cannot change the memory access pattern of Algorithm 1.*

Proof. Replacing any queried term with any term in the same bucket does not change **L** in Algorithm 1. The proof follows the observations as below:

(1) Iterating over sorted bucket lists **L** is deterministic. Since we do not change **L**, the memory access pattern of the iterating is not changed either. Computing feature aggregation $\sum_{i \in D} s_i$ is deterministic for each document given the same bucket lists **L**.
(2) The for loop in Algorithm 2 to compute the feature offset is oblivious to any input, since the memory access sequence is always a linear scan on the b bits of the bitmap.
(3) Following the for loop, the feature aggregation in Algorithm 2 is oblivious if only selector is changed in the input data. Note that in Algorithm 1, as long as **L** is not changed, the sequence of documents that are passed to Algorithm 2 is not changed either. For each call of Algorithm 2, only selectors can be changed for different queries, while masks and feature lists keep the same. Hence the memory access pattern of feature aggregation in Algorithm 2 is not changed.
(4) `Equal` has the same memory access pattern for any inputs a and b, namely, the sequence $(r1, \downarrow, r2, \downarrow, \downarrow, \downarrow, ret)$, where $r1$, $r2$ denote reading the memory locations of two arguments for `Equal` respectively, $\downarrow$ denotes reading the next instruction of the algorithm, and ret denotes returning to the caller of `Equal`.
(5) Decryption, decompression, oblivious sorting and encryption give the same memory access pattern for any input lists with identical length.

□

Theorem 5.2. *For any query with q terms, there exist at least $b^q - 1$ other queries, where b denotes the number of terms in each bucket, such that Algorithm 1 is oblivious over all these b^q queries.*

Proof. By Lemma 5.1, for any query with q terms, each of the q terms can be replaced to b terms in the same bucket while preserving memory access patterns. Hence the entire query can change to at least $b^q - 1$ other alternative queries preserving obliviousness of Algorithm 1. □

5.2 Obfuscations of terms and queries

Theorem 5.3. *Assume each of V terms is duplicated to k buckets, and each bucket merges b terms. Then on average each query with q different terms is obfuscated over at least $\left((b-1)(k-1)(1-\frac{bk(k-1)}{2kV-2})\right)^q$ different queries, which are possible to match the same buckets during document retrieval.*

Proof. We first derive the lower bound on the number of terms which any query term is obfuscated over. Following the mapping process of Section 4.2, let $w_{i,j}$ denote the j-th term merged by the i-th matched bucket ($1 \le i \le k$ and $1 \le j \le b$). Hence

$$\Pr[w_{i,j} = w_{i',j'} | i > i'] = \frac{V\binom{k}{2}}{\binom{kV}{2}} = \frac{k-1}{kV-1}.$$

By union bound,

$$\Pr[\bigvee_{(i',j'):i>i'} w_{i,j} = w_{i',j'}] \le \sum_{(i',j'):i>i'} \Pr[w_{i,j} = w_{i',j'}] = \frac{(i-1)b(k-1)}{kV-1}.$$

Let indicator $\mathbf{1}_{i,j}$ denote that there exists no (i',j') such that $i > i'$ and $w_{i,j} = w_{i',j'}$. Hence

$$\mathbf{E}[\mathbf{1}_{i,j}] = 1 - \Pr[\bigvee_{(i',j'):i>i'} w_{i,j} = w_{i',j'}] \ge 1 - \frac{(i-1)b(k-1)}{kV-1}.$$

Following the collision tolerance condition enforced in the bucket building process of Section 4.2, and by arranging the orders of bucket locations for each term, and the term copies in each bucket, we can satisfy that

(1) only the first two bucket locations for any term can be the same, and
(2) only the first two term copies in each bucket can be the same.

Hence the average number of terms which the matched buckets are obfuscated over is at least

$$\mathbf{E}[\sum_{i=2}^{k}\sum_{j=2}^{b}\mathbf{1}_{i,j}] = \sum_{i=2}^{k}\sum_{j=2}^{b}\mathbf{E}[\mathbf{1}_{i,j}] \geq \sum_{i=2}^{k}\sum_{j=2}^{b}\left(1 - \frac{(i-1)b(k-1)}{kV-1}\right)$$

$$= (b-1)(k-1) - \frac{b(b-1)k(k-1)^2}{2kV-2}.$$

Since each of the q terms in the query has the above lower bound of obfuscation, raising to the power of q gives the lower bound for the entire query obfuscation. □

From the above result, any term t with k duplicates in a given inverted index of V terms with b terms per bucket, on average term t is obfuscated by at least $(b-1)(k-1)(1-\frac{bk(k-1)}{2kV-2})$ different terms. Since a term in REARGUARD is obfuscated by $g-1$ terms in a group of size g, we can choose $(b-1)(k-1) \approx g$ so that privacy protection in MII is competitive to REARGUARD at least in terms of term and query obfuscation.

5.3 Leakage profile

We list all the following information that can be observed by a server when processing a query with MII. 1) *Static size information.* The number of term buckets in the index. The number of documents in each term bucket. 2) *Dynamic query size information.* The number of search terms in each query. The size of matched documents with padded results for a query. 3) *Term bucket access patterns.* The set of term buckets accessed for each query is exposed. The server can compute statistical information such as bucket access frequency. When two or more term buckets are queried, the set of documents appearing in these buckets may be leaked. Note that this only leaks positions of documents in these buckets, not real documents IDs. 4) *Feature size patterns.* The number of features of each document in each bucket posting list is exposed. This also tells the adversary how many of the b terms of the bucket are contained in each document.

Among all leakages specified above, there is no known privacy issue on learning the number of search terms and buckets used. For the length of each bucket, since the bucket has b randomly-mixed terms where $b > 1$, it is unlikely that the server can calculate the length of the posting list for a real term. For the result size information, since we pad unmatched or unselected documents, the server is not able to identify the real size. For the access patterns, since a bucket contains multiple terms that a server cannot differentiate, it is unlikely that the server can accurately compute the frequency of terms that appear in a history of queries, and calculate the document sharing pattern between postings of real terms.

6 COMPLEXITY COMPARISON

Table 2 gives a comparison of the index storage space and document matching time of MII, with the extended REARGUARD scheme [38] for handling q terms when the group size is g. The space cost is represented by the number of integers used to store the index before compression. The original REARGUARD does not deal with multiple search keywords or ranking also, and our extension is based on the best option we choose. Thus this comparison is illustrative to explain a cost advantage of MII over REARGUARD under certain assumptions. The time cost listed includes server side disk I/O and in-memory data processing. Notice that decryption of posting records can be conducted in the entire list for each term block, thus it is relatively fast. The client-side query processing cost of REARGUARD and MII is comparable, and is relatively insignificant.

Assumptions. Let n be the number of documents in a dataset, and V be the number of unique terms. It is known that the number of documents in each term's posting list often follows a Zipf-like distribution, and Table 2 illustrates the complexity difference under a simple Zipf distribution: assuming that the length of posting list for the i-th popular term is $\frac{n}{i}$. The longest posting list length of a term is n, and the average posting list length is $\frac{1}{V}\sum_{i=1}^{V}\frac{n}{i} \approx \frac{n\ln V}{V}$.

The space cost of MII is the sum of all posting lists multiplied by k because of term duplication. The query processing time cost of MII is proportional to the average posting list length multiplied by b and q.

For REARGUARD, with each posting list group of size g and following the Zipf distribution, we can show that the average posting list length of each group is $\Theta(\frac{ng}{V}\ln\frac{V}{g})$, considering V is large and $V >> g$. The detailed analysis is omitted due to the page limit of this paper. Given the fact that each posting list in a group has to be processed for each query term, the time cost is this number multiplied by g and q.

Table 2: A comparison of space and time complexity

	REARGUARD	MII
# of integers in the index	$\Theta(V + ng\ln\frac{V}{g})$	$\Theta(\frac{kV}{b} + kn\ln V)$
Server-side time	$\Theta(\frac{qng^2}{V}\ln\frac{V}{g})$	$\Theta(\frac{qbn}{V}\ln V)$

In general, b and k are small constants while g needs to be reasonably large. Assuming $n >> V$ and $V >> g$, the ratio of space cost of REARGUARD over MII is approximately $\Theta(\frac{g}{kb})$, thus the index storage space of MII should be the same as that of REARGUARD, under this simple Zipf distribution when choosing $g = kb$. In addition, the time cost ratio of REARGUARD over MII is about $\Theta(\frac{g^2}{b})$. As $g > b$, MII is significantly faster.

7 EVALUATION

Experimental Settings. We evaluate disjunctive queries on MII and three other baselines, including block-max WAND, exhaustive OR, and REARGUARD. Given the fact that there is no standard IR toolkits supported by Intel SGX, we built our implementations for MII and all baselines with the C/C++ library under the SGX programming environment, which can make fair comparisons. The implementations of REARGUARD and MII access their corresponding encrypted indexes. All indexes are compressed using Simple-9 [2] before any encryption and this compression is simple with a reasonable effectiveness for our setting [42]. Our implementations of BMW and exhaustive OR directly access unencrypted indexes. We let $g = 85$ for REARGUARD following the setting in [38]. We also let $k = 18$ and $b = 6$ for MII. In this case, $g = (k-1)(b-1)$

and thus the privacy level of MII is approximately the same as that of REARGUARD. Experiments are conducted on a single machine with Intel Core i7-9700K CPU 3.60GHz, 32GB RAM, Samsung 970 NVMe SSD running Ubuntu 18.04 with Intel SGX Linux 2.7 SDK.

Datasets. Two TREC collections are evaluated: ClueWeb-09-Cat-B and TREC disks 4&5. TREC4&5 has about 0.5 million documents, and ClueWeb has nearly 30 millions documents after removing those with Waterloo spam score [9] below 40. We assume that ClueWeb can be hosted on multiple cores for parallel query processing. Thus we randomly partition this dataset and use one partition with 1 million documents for performance assessment. For TREC4&5, 530, 348 terms in total are indexed including Stop Words. For ClueWeb, with discarding terms whose frequency no more than 2, there are in total 1, 239, 769 terms indexed. For query processing, there are 837 queries for ClueWeb from WebTrack 2010-2012 and Millions Query Track. TREC4&5 uses 250 queries from Robust 2004. All test queries have 1-5 query keywords. Both documents and query words are stemmed using the Krovetz stemmer [24].

A comparison of query processing times. In Table 3, the query time for document retrieval with MII and three baselines are listed. An average time is reported in each different query length, and the rightmost column is the average time for all test queries. The time cost in Table 3 measures the duration starting from the time when the server receives (encrypted) queries, to the time when all top 1000 matched documents are derived. We make sure the index is not cached before each single query.

MII vs. REARGUARD. We observe that compared with REARGUARD, MII is 6.73x faster for TREC4&5, and 10.71x faster for ClueWeb. Notice that the time spent on the disk I/O to fetch posting lists occupies about 80% of time in REARGUARD, and about 56% in MII. The reduction ratio is not as high as the predicted number in Section 6, because the estimation does not include the startup cost of each I/O operation to access the SSD, and our test data is not exactly a simple Zipf distribution. The evaluation, however, still agrees on the trend that MII reduces the matching time significantly, including all the cases when the test query length varies.

Oblivious search vs. unprotected BMW. MII is slower than BMW by 3.32x and 4.90x mainly because of block-max WAND skipping documents based on top k thresholds and block-max scores. When compared with exhaustive OR, MII is 1.04x and 1.09x slower because there are posting duplicates in masked buckets. Notice that our implementation of block-max WAND is 3.18x and 4.48x faster than that of exhaustive OR, which is not as high as the ratio claimed by Ding and Suel [13]. This is because our query processing time includes the time of loading posting lists from disks, which is not considered in the block-max WAND paper.

Storage cost. As mentioned above, all indexes were compressed using Simple-9 compression technique. Using different compression methods did affect our reported storage sizes [42], but not by much. BMW and exhaustive OR use the same unencrypted index, which is 0.2GB for TREC4&5, and 11.8GB for ClueWeb. The storage size for MII is 3.1GB for TREC4&5, and 207.5GB for ClueWeb, which includes masked posting lists, vocabulary, and term buckets. These costs are around 15.5x and 17.6x as big as those of the unencrypted index in both datasets because of posting duplication. For REARGUARD, with the group size g as 85, the index size with group-wide posting padding is 8.5GB for TREC4&5, and 709.1GB for ClueWeb. Thus the index sizes of REARGUARD are about 2.7x and 3.4x as big as those of MII for our two datasets. These ratios are not too far away from the predicted ratio $\Theta(\frac{g}{k})$ in Section 6, which is around 4.7x.

Table 3: Comparing document retrieval time (milliseconds)

# Query Words		1	2	3	4-5	Average
TREC4&5	BMW	1	2	3	4	2.8
	Exhaustive OR	2	7	10	11	8.9
	REARGUARD	21	45	72	83	62.6
	MII	3	8	10	11	9.3
ClueWeb	BMW	31	73	134	254	125.7
	Exhaustive OR	116	311	583	1,212	560.1
	REARGUARD	620	2,112	6,409	16,723	6,557.9
	MII	133	318	631	1,327	612.4

Table 4: Comparing different term bucketing settings in MII

MII Settings		Avg. Query Time (ms)	Index Size (GB)
TREC4&5	k = 18, b = 6	9.3	3.1
	k = 6, b = 18	11.4 (1.2x)	0.9 (0.29x)
ClueWeb	k = 18, b = 6	612.4	207.5
	k = 6, b = 18	924.7 (1.5x)	67.1 (0.32x)

Impact of different term bucket settings. Table 4 shows the impact of different term bucket settings, k and b, average query processing time and storage cost for our test datasets. The first setting, $k = 18$ and $b = 6$, is the one used in Table 3. The second one, $k = 6$ and $b = 18$, has the approximately same privacy level as that of the first setting. For storage sizes, close to what we predicted earlier, when k being 0.33x, MII costs around 0.29x and 0.32x as big as the storage cost of the first setting. This observation is consistent with Table 2 in Section 6 where MII space cost is proportional to k. With b being 3x larger, the second setting costs 1.2x and 1.5x query processing time compared to the first one on two datasets. The increasing trend modestly agrees with Table 2 and the increasing ratio is smaller than expected due to the discrepancy in posting length distribution estimation.

Evaluating oblivious feature extraction. As discussed at the end of Section 4, our oblivious feature extraction method is expected to reduce the storage cost greatly, in a situation where a document in a bucket posting record only matches some of those b terms in the bucket, compared with the naive oblivious method described in Section 4.3. Our experiments show that, when $k = 18$ and $b = 6$, the naive method costs around 2.5x and 2.6x storage space as those of our proposed oblivious method in two datasets, which shows the advantage of our space-conscious oblivious design. In addition, the time cost difference is insignificant to the total query processing time, as both of them are linear with the value of b.

8 CONCLUDING REMARKS

This paper proposes an oblivious document retrieval scheme with an obfuscated inverted index to hide document-term association and avoid pattern leakage of memory access operations for privacy protection. Our evaluation shows MII achieves an up-to 18.9x matching time speed-up over REARGUARD while the storage size is up-to 3.4x smaller. The oblivious but exhaustive approach in MII is still significantly slower than BMW [13], which represents a degradation of efficiency traded for privacy because WAND-based optimization in BMW designed without privacy constraints [4, 13, 26] is vulnerable to leakage-abuse attacks as studied in Section 3.

Intel SGX has around 90MB usable protected buffer memory [10], which is big enough to contain all posting records of the searched term buckets for our tested datasets. For hosting buckets with longer lists, there could be some slowdown in matching due to buffer pages being swapped into/out of the untrusted memory. TEEs like SGX still reside on the server machines and the risk such as physical attacks [10] exists, which is outside the scope of this study. Our solution provides an alternative approach to the software-only privacy-preserving solutions for document matching. A future work is to integrate with privacy-preserving ranking.

ACKNOWLEDGMENTS

This work is supported in part by NSF IIS-1528041 and 2040146, and by a Google faculty research award. It has used the NSF-supported resource in the Extreme Science and Engineering Discovery Environment under allocation IRI190005. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] Daniel Agun, Jinjin Shao, Shiyu Ji, Stefano Tessaro, and Tao Yang. 2018. Privacy and efficiency tradeoffs for multiword top k search with linear additive rank scoring. In *Proceedings of the 2018 World Wide Web Conference.* 1725–1734.

[2] Vo Ngoc Anh and Alistair Moffat. 2005. Inverted index compression using word-aligned binary codes. *Information Retrieval* 8, 1 (2005), 151–166.

[3] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure: SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies, 2017.*

[4] Andrei Z Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. 2003. Efficient query evaluation using a two-level retrieval process. In *Proc. of CIKM'03.* 426–434.

[5] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. 2014. Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data. *IEEE Trans. Parallel Distrib. Syst.* 25, 1 (2014), 222–233.

[6] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. leakage-abuse attacks against searchable encryption. In *CCS'15.* ACM, 668–679.

[7] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2014. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation.. In *NDSS*, Vol. 14. 23–26.

[8] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. 2013. Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In *CRYPTO 2013.* 353–373.

[9] Gordon V Cormack, Mark D Smucker, and Charles LA Clarke. 2011. Efficient and effective spam filtering and re-ranking for large web datasets. *Information retrieval* 14, 5 (2011), 441–465.

[10] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* (2016), 1–118.

[11] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2006. Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions *(CCS '06).* ACM, 79–88.

[12] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. 2011. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security* 19, 5 (2011), 895–934.

[13] Shuai Ding and Torsten Suel. 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.* 993–1002.

[14] Benny Fuhry, Raad Bahmani, Ferdinand Brasser, Florian Hahn, Florian Kerschbaum, and Ahmad-Reza Sadeghi. 2017. HardIDX: Practical and secure index with SGX. In *IFIP Ann. Conf. on Data and App. Security and Privacy.* 386–408.

[15] Craig Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *STOC '09.* ACM, 169–178.

[16] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.

[17] Thang Hoang, Muslum Ozgur Ozmen, Yeongjin Jang, and Attila A Yavuz. 2019. Hardware-Supported ORAM in Effect: Practical Oblivious Search and Update on Very Large Dataset. *Proc. on Privacy Enhancing Technologies* 1 (2019), 172–191.

[18] Haibo Hu, Jianliang Xu, Chushi Ren, and Byron Choi. 2011. Processing private queries over untrusted data cloud through privacy homomorphism. In *ICDE.* 601–612.

[19] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *NDSS 2012.*

[20] Shiyu Ji, Jinjin Shao, Daniel Agun, and Tao Yang. 2018. Privacy-aware Ranking with Tree Ensembles on the Cloud. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval.* ACM, 315–324.

[21] Karen Spärck Jones, Steve Walker, and Stephen E. Robertson. 2000. A probabilistic model of information retrieval: development and comparative experiments. In *Information Processing and Management.* 779–840.

[22] Seny Kamara and Tarik Moataz. 2017. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 94–124.

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

[24] Robert Krovetz. 2000. Viewing morphology as an inference process. *Artificial intelligence* 118, 1-2 (2000), 277–294.

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

[26] Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto, and Rossano Venturini. 2017. Faster BlockMax WAND with variable-sized blocks. In *Proc. of SIGIR'2017.* 625–634.

[27] Oliver Masters, Hamish Hunt, Enrico Steffinlongo, Jack Crawford, Flavio Bergamaschi, Maria E. Dela Rosa, Caio C. Quini, Camila T. Alves, Feranda de Souza, and Deise G. Ferreira. 2019. Towards a Homomorphic Machine Learning Big Data Pipeline for the Financial Services Sector. Cryptology ePrint Archive, Report 2019/1113. (2019).

[28] Irina Matveeva, Chris Burges, Timo Burkard, Andy Laucius, and Leon Wong. 2006. High Accuracy Retrieval with Multiple Nested Ranker. In *SIGIR (SIGIR '06).* Association for Computing Machinery, New York, NY, USA, 437–444.

[29] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. 2018. Oblix: An efficient oblivious search index. In *2018 IEEE Symposium on Security and Privacy (SP).* 279–296.

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

[31] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious multi-party machine learning on trusted processors. In *Proc. of 2016 USENIX Security Symp.* 619–636.

[32] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT '99.* 223–238.

[33] Jinjin Shao, Shiyu Ji, and Tao Yang. 2019. Privacy-Aware Document Ranking with Neural Signals. In *Proc. of 2019 SIGIR.* 305–314.

[34] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. 2000. Practical Techniques for Searches on Encrypted Data *(SP '00).* IEEE Computer Society.

[35] Emil Stefanov, Marten Van Dijk, Elaine Shi, T-H Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2018. Path oram: An extremely simple oblivious ram protocol. *J. ACM* 65, 4 (2018), 1–26.

[36] Trevor Strohman and W Bruce Croft. 2007. Efficient document retrieval in main memory. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval.* 175–182.

[37] Wenhai Sun, Bing Wang, Ning Cao, Ming Li, Wenjing Lou, Y. Thomas Hou, and Hui Li. 2014. Verifiable Privacy-Preserving Multi-Keyword Text Search in the Cloud Supporting Similarity-Based Ranking. *IEEE Trans. Parallel Distrib. Syst.* 25, 11 (2014), 3025–3035.

[38] Wenhai Sun, Ruide Zhang, Wenjing Lou, and Y Thomas Hou. 2018. Rearguard: Secure keyword search using trusted hardware. In *IEEE INFOCOM 2018.* 801–809.

[39] Guofeng Wang, Chuanyi Liu, Yingfei Dong, Kim-Kwang Raymond Choo, Peiyi Han, Hezhong Pan, and Binxing Fang. 2018. Leakage Models and Inference Attacks on Searchable Encryption for Cyber-Physical Social Systems. *IEEE Access* 6 (2018), 21828–21839.

[40] Lidan Wang, Jimmy Lin, and Donald Metzler. 2011. A Cascade Ranking Model for Efficient Ranked Retrieval. In *SIGIR (SIGIR '11).* Association for Computing Machinery, New York, NY, USA, 105–114. https://doi.org/10.1145/2009916.2009934

[41] Zhihua Xia, Xinhui Wang, Xingming Sun, and Qian Wang. 2016. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE Transactions on Parallel and Distributed Systems* 27, 2 (2016), 340–352.

[42] Jiangong Zhang, Xiaohui Long, and Torsten Suel. 2008. Performance of compressed inverted list caching in search engines. In *Proceedings of the 17th International Conference on World Wide Web, WWW.* 387–396.

[43] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An oblivious and encrypted distributed analytics platform. In *NSDI 17*. 283–298.

A APPENDIX

A.1 Mapping from term copies to buckets

We investigate the probability of restarting a random mapping from kV term copies to the kV/b buckets defined in Section 4.2. This mapping groups b consecutive term copies in a randomly shuffled sequence of kV term copies: $\{(i,j) : 1 \le i \le V, 1 \le j \le k\}$ where tuple (i,j) represents the j-th copy of the i-th term. Recall that a collision pair is defined as two copies of the same term mapped to the same bucket.

We also call **(k,b)-allocation** as a parameter pair with bucket size b and k duplicate degree of each term. We define **random variable** $Y_i^{(k,b)}$ as the number of collision pairs among the b term copies sent to the i-th bucket for this (k,b)-allocation.

Lemma A.1. *For (k,b)-allocation,*

$$\Pr[Y_i^{(k,b)} \le 1] = \frac{\binom{V}{b}k^b + \binom{V}{b-1}(b-1)\binom{k}{2}k^{b-2}}{\binom{kV}{b}}.$$

Proof. The total number of all possible combinations of the term copies sent to the i-th bucket is $\binom{kV}{b}$. The number of combinations that there is no collision among the b term copies sent to the i-th bucket is $\binom{V}{b}k^b$. Hence $\Pr[Y_i^{(k,b)} = 0] = \binom{V}{b}k^b/\binom{kV}{b}$. The number of combinations that there is exactly one collision among the b term copies sent to the i-th bucket is $\binom{V}{b-1}(b-1)\binom{k}{2}k^{b-2}$, which can be shown using a counting argument:

(1) choose $b-1$ terms;
(2) choose one term among $b-1$ for where the collision happens;
(3) the collision can happen between any two of k copies of the chosen term;
(4) each of the other $b-2$ terms has k copies to choose from.

Hence $\Pr[Y_i^{(k,b)} = 1] = \binom{V}{b-1}(b-1)\binom{k}{2}k^{b-2}/\binom{kV}{b}$. □

Lemma A.2. *For (k,b)-allocation, if $\frac{b(b+1)}{2} - 1 << V$, then*

$$\Pr[Y_i^{(k,b)} > 1] \le \frac{\binom{b}{3} + 3\binom{b}{4}}{V^2}.$$

Proof. By Lemma A.1 we have

$$\Pr[Y_i^{(k,b)} \le 1] = \frac{V\cdots(V-b+1) + V\cdots(V-b+2)b(b-1)\left(\frac{1}{2} - \frac{1}{2k}\right)}{V\cdots\left(V - \frac{b-1}{k}\right)}.$$

Since $\frac{b(b+1)}{2} - 1 << V$, we have $V - b + 1 >> \frac{b(b-1)}{2}$. Hence $\frac{b(b-1)}{2(V-b+1)} = o(1)$, where $o(1)$ denotes sufficiently small positive number given sufficiently large V. Then

$$\Pr[Y_i^{(k,b)} \le 1] = \frac{V\cdots(V-b+1)(1+o(1))}{V\cdots\left(V - \frac{b-1}{k}\right)},$$

which asymptotically decreases as k increases. Hence the probability $\Pr[Y_i^{(k,b)} > 1]$ is upper bounded by $\Pr[Y_i^{(\infty,b)} > 1]$ for (∞,b)-allocation, which has infinite buckets, and each bucket chooses b terms independently uniformly. For (∞,b)-allocation, $\Pr[Y_i^{(\infty,b)} > 1]$ can be upper bounded by the following argument:

(1) the probability that there are two pairs of copies, each of which comes from the same term, is $\binom{b}{4}\cdot 3V^{b-2}/V^b$, which can be shown using a counting argument: a) choose 4 copies for two collision pairs; b) there are 3 ways to arrange the chosen 4 copies for two pairs (see Figure 3); c) for each collision pair and each copy that is not chosen in a), its term has V possibilities;
(2) the probability that there are three copies from the same term is $\binom{b}{3}V^{b-2}/V^b$, which can be shown by a counting argument similar to 1);
(3) $Y_i^{(\infty,b)} > 1$ if and only if (1) or (2) happens.

Note that the events of (1) and (2) are overlapped, e.g., the case when more than 3 copies in one bucket come from the same term is in both (1) and (2). Hence we only claim an upper bound using union bound. □

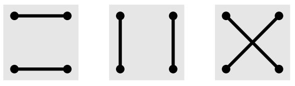
Figure 3: Three ways to arrange two collision pairs among four chosen term copies.

Theorem A.3. *If $\frac{b(b+1)}{2} - 1 << V$, the probability to start over the random mapping algorithm is at most*

$$\frac{b^2k}{V}\left(\frac{1}{3} + \frac{b+k}{8}\right).$$

Proof. Let X_i denote the number of collision pairs among the copies from the i-th term, and let Y_j denote the number of collision pairs among the copies to the j-th bucket. Then by Lemma A.2

$$\Pr[X_i > 1] \le \frac{\binom{k}{3} + 3\binom{k}{4}}{B^2} \le \frac{1}{B^2}\left(\frac{k^3}{6} + \frac{k^4}{8}\right),$$

$$\Pr[Y_i > 1] \le \frac{\binom{b}{3} + 3\binom{b}{4}}{V^2} \le \frac{1}{V^2}\left(\frac{b^3}{6} + \frac{b^4}{8}\right).$$

By union bound and the fact $B = kV/b$,

$$\Pr[\max_{1\le i\le V} X_i > 1] \le \sum_{i=1}^{V}\Pr[X_i > 1] = \frac{b^2}{V}\left(\frac{k}{6} + \frac{k^2}{8}\right),$$

$$\Pr[\max_{1\le i\le B} Y_i > 1] \le \sum_{i=1}^{B}\Pr[Y_i > 1] = \frac{k}{V}\left(\frac{b^2}{6} + \frac{b^3}{8}\right).$$

Based on the collision tolerance condition in Section 4.2, the random term-bucket re-mapping restarts if one or both of the following conditions hold:

- 1) Each term has at least $(k-1)$ unique bucket locations if and only if the number of collision pairs among the copies of each term is more than 1, namely, $\max_{1\le i\le V} X_i > 1$.
- 2) Similarly, each bucket has at least $(b-1)$ unique terms if and only if $\max_{1\le i\le B} Y_i > 1$.

Thus the probability to start over re-mapping is at most

$$\Pr[\max_{1\le i\le V} X_i > 1 \vee \max_{1\le i\le B} Y_i > 1] \le \frac{1}{V}\left(\frac{b^2k}{3} + \frac{b^2k^2 + b^3k}{8}\right).$$

□