Private web search

Appeared at SOSP 2023
Web-search queries reveal our sensitive data

Health  ballet knee problem

Finances  job opportunities in west palm beach

Religion  african american churches in norfolk va

Citizenship  application forms us citizen

https://trec.nist.gov/data/million.query07.html
Today: Search engines learn our queries

“knee problem”

hopkinsmedicine.org/health/knee-pain
Today: Search engines learn our queries

- "knee problem"
- hopkinsmedicine.org/health/knee-pain

Google

Microsoft Bing

Baidu

Yandex

Your queries

Attacker
(data breach)

SketchyCo
(resale)

LearningCo
(training)
Goal: Search **without revealing** query

“knee problem”

```
Enc("knee problem")
```

```
Enc(hopkinsmedicine.org/health/knee-pain)
```

hopkinsmedicine.org/health/knee-pain

Private search engine
Goal: Search **without revealing** query

“knee problem”

hopkinsmedicine.org/health/knee-pain
Goal: Search **without revealing** query

“knee problem”

hopkinsmedicine.org/health/knee-pain

Private search engine

Attacker (data breach)
SketchyCo (resale)
LearningCo (training)
Goal: Search *without revealing* query

Non-goals:  
- does not hide *when* the client makes searches  
- does not guarantee *integrity* of search results  
- does not hide subsequent HTTP(S) requests
Goal: Search **without revealing** query

"knee problem"

hopkinsmedicine.org/health/knee-pain

Private search engine

Attacker (data breach)

SketchyCo (resale)

LearningCo (training)

Theoretically possible: Fully homomorphic encryption [RAD’78, Gen’09]

But, classic search algorithms are **very expensive** to express as circuits
Goal: Search **without revealing** query

"knee problem"

Private search engine

Attacker (data breach)
SketchyCo (resale)
LearningCo (training)

hopkinsmedicine.org/health/knee-pain

This work: **Linearly** homomorphic encryption suffices
Modern ML turns messy search computations into cheap, linear ones
Tiptoe: A private search engine

+ Search engine learns no information about the client’s queries i.e., semantic security relying on LWE and ring-LWE

+ Supports text & image search

+ Searches over public web crawl (364M pages) in 2.7s of latency with 145 core-s of compute, 57 MiB of traffic, and 0.3 GiB of client storage

- Search results not yet as good as with non-private search engines
Private search on private data

- Searchable Encryption
  [SWP’00, CGKO’11, CryptDB’11, SPS’14, …]

- Oblivious RAM
  [GO’96, O’90, SVSRYD’13, Dory’20, …]

Private search on public data

- Private information retrieval
  only key-value lookups

- Google over Tor
  [DMS’04]
  leaks query contents

→ Query-private search:
  Tiptoe, Coeus
  [ASAEG’21]
  expressive queries, hides query contents
**Tiptoe: Architecture**

- **Encryption**:
  - Enc($f(\text{query})$)
  - Enc(doc=92)
  - Enc(webmd.com)

- **Document Embeddings**:
  - doc91: 27%
  - doc92: 84%
  - doc93: 02%

- **Ranking Service**

- **URL Service**

- **Indexing Batch Job**

- Client-facing Tiptoe services:
  - The tiptoe client uses a new cryptographic protocol to obtain private information.
  - This protocol requires a single logical server.
  - It relies only on cryptographic data structures used for the cryptographic protocols.

- **Finding the Best Documents**:
  - Once the client knows the query embedding, it computes the inner product distance and returns only an approximate result.
  - The client must find the documents whose embedding vectors are nearest to its query embedding.
  - The distance between its query embedding and all the cluster centroids is taken.
  - The client finds the cluster that best matches its query within a particular cluster.

- **Embeddings and Preprocessing**:
  - The embeddings we use for preprocessing the document urls, the document ids, and not on the corpus itself.
  - The corpus indexing system requires a single logical server.
  - It relies only on cryptographic

- **Relevance Retrieval**:
  - Text search are vectors of floating-point numbers.
  - The indexing jobs output by the indexing batch jobs.

- **Document Matching**:
  - The client uses the centroids to find the cluster with the best match its query within a particular cluster.
  - The distance between its query embedding and all the cluster centroids is taken.
  - The client finds the cluster that best matches its query within a particular cluster.

- **Scalability**:
  - This function to embed its query string into a vector.
  - This function depends only on the type of document being indexed.
  - The indexing jobs take $RVU \text{mib}$ to represent.
  - Other popular embedding services require $XXU \text{mib}$ of storage for our text embedding function.

- **Privacy and Security**:
  - The client uses the ids of the best-matching documents within cluster.
  - The client uses the url service to query the url service for this.
  - The tiptoe client uses the url service to find the ids of the documents that scale beyond a few thousand items require multiple non-colluding servers for security.

- **TIPTOE Text Search**
  - The corresponding text search engines require between one and two gigabytes of storage.
  - The documents in its chosen cluster.

- **TIP TOE Protocol**
  - This protocol requires a single logical server.
  - It relies only on cryptographic

- **Performance**:
  - The distance between its query embedding and all the cluster centroids is taken.
  - The client finds the cluster that best matches its query within a particular cluster.
  - The distance between its query embedding and all the cluster centroids is taken.
  - The client finds the cluster that best matches its query within a particular cluster.

- **Evaluation**:
  - The distance between its query embedding and all the cluster centroids is taken.
  - The client finds the cluster that best matches its query within a particular cluster.
  - The distance between its query embedding and all the cluster centroids is taken.
  - The client finds the cluster that best matches its query within a particular cluster.
Tiptoe: Architecture

1. Client provides a query (query) to the Ranking service.
2. The Ranking service computes the distance between the query and all documents in the corpus indexing.
3. The Ranking service ranks the documents and returns the top results.

- **Indexing batch job**: Encrypts the contents before storing in the corpus indexing.
- **URL service**: Decrypts the URL and associated metadata for a document. Encrypted (webmd.com)
- **Client-facing Tiptoe services**: Allows users to search for documents without revealing their query.

**Private nearest-neighbor search**
- Encrypted query is compared against cluster centers.
- Upon each query, the client uses its locally cached set of cluster centroids to identify the cluster nearest to its query embedding.
- For more retrieval protocols, see the figure on the right.
Tiptoe: Architecture

Enc\( f(\text{ query })\)

Enc (doc=92)

Enc (webmd.com)

Doc91: 27%
Doc92: 84%
Doc93: 02%
...

Ranking service

URL service

Indexing batch job

Clients.

Corpus indexing.

Clients.

Clients.

About related topics. Clustering is a common technique in presentation of the document. The output of this step is one indexed document and not on the corpus itself. Function depends only on the type of document being embedded into a vector.

Cluster the vectors and generate cluster centers. We use the tiptoe batch jobs to compute the centroids of each cluster.

The tiptoe client uses a new cryptographic protocol to obtain the best documents within that cluster. It embeds the query string into a vector, identifies the cluster nearest to its query embedding, and sends this list to the tiptoe service for search over these inner-product scores to identify the documents that are closest to its query vector. It must find the documents whose embedding vectors are nearest to its query vector, or its chosen cluster to the tiptoe service. The tiptoe client uses the url service to query the url service for this list, as our text-embedding function takes \(8\) mib to represent other popular embedding data structures used for the cryptographic protocols.

Since nearby embedding vectors represent documents that are close in content, the documents within each cluster are embeded in a vector. Group the embedding vectors into clusters of tens or thousands. First, the indexing jobs run each document through text search are vectors of \(\times\) floats. Second, the indexing jobs assign services run on a cluster of tens of physical machines.

Figure RZ, the tiptoe system architecture, shows the tiptoe batch jobs use a pretrained embedding function to map the raw query information retrieval systems to a query vector. In vector space, this is a private nearest-neighbor search problem, more precisely, since we measure closeness by inner-product distance and return only an approximate result. The tiptoe client uses the url service to query the url service for this list, as our protocol requires a single logical server. It relies only on cryptographic data structures, and the client and servers each store the cryptographic data structures. The output by the indexing batch jobs is the tiptoe client uses the url service to query the url service for this list, as our protocol requires a single logical server. It relies only on cryptographic data structures, and the client and servers each store the cryptographic data structures.
**Tiptoe: Design steps**

1. **Standard technique:** Reduce text search to nearest-neighbor search  
   Key tool: *Semantic embeddings* [Osgood’57, ...]

2. **Our contribution:** Fast *private* nearest-neighbor search  
   Key tools: *Clustering* to reduce communication  
   + Linearly homomorphic encryption with preprocessing  
   to shrink the computation [SimplePIR’23]
Represent documents and queries using semantic embeddings

[DC’19, MYCG’19, YYZL’19, SKPZ’22, …]
Represent documents and queries using semantic embeddings
[DC’19, MYCG’19, YYZL’19, SKPZ’22, …]

Required property: when doc 1 and doc 2 are “similar” in meaning, their embedding inner-product score $\langle e_1, e_2 \rangle$ is large.
Represent documents and queries using semantic embeddings

[DC’19, MYCG’19, YYZL’19, SKPZ’22, …]

Required property: when doc 1 and doc 2 are “similar” in meaning, their embedding inner-product score $\langle e_1, e_2 \rangle$ is large.
Represent documents and queries using semantic embeddings
[DC’19, MYCG’19, YYZL’19, SKPZ’22, …]

“knee problem”

Embed
(270 MB)

Embedding space
(e.g., dense vectors of 192 floats)

Documents

-goal: privately find the doc that maximizes the score $\langle q, e \rangle$
Perform coarse nearest-neighbor search locally on the client
Perform coarse nearest-neighbor search locally on the client

Ahead of time: Server groups the $N$ docs into $\sqrt{N}$ clusters
Perform coarse nearest-neighbor search locally on the client

**Ahead of time:** Server groups the $N$ docs into $\sqrt{N}$ clusters

**At query time:** Client uses local list of centroids to find the closest cluster
Perform coarse nearest-neighbor search locally on the client

Ahead of time: Server groups the $N$ docs into $\sqrt{N}$ clusters

At query time: Client uses local list of centroids to find the closest cluster

Goal: privately fetch inner-product scores $\langle q, e \rangle$ for docs in Cluster 2
Perform exact search of the closest cluster under encryption

Best match:
cluster = 2
doc = ?
Perform exact search of the closest cluster under encryption

Best match:
cluster = 2
doc = ?
Perform exact search of the closest cluster under encryption

Best match:
cluster = 2
doc = 3

Enc

× Enc

Server

Enc

$\begin{pmatrix}
0 \\
q \\
0 \\
\vdots \\
0
\end{pmatrix}$ ← index of cluster 2

$\begin{pmatrix}
4 \\
10 \\
\vdots \\
3
\end{pmatrix}$ = score of doc 1

$\begin{pmatrix}
\vdots
\end{pmatrix}$

$\begin{pmatrix}
72
\end{pmatrix}$ = score of doc 2

$\begin{pmatrix}
\vdots
\end{pmatrix}$

$\begin{pmatrix}
3
\end{pmatrix}$ = score of doc k

$\begin{pmatrix}
e_{1,1} & e_{1,2} & e_{1,3} & \cdots & e_{1,C} \\
e_{2,1} & e_{2,2} & e_{2,3} & \cdots & e_{2,C} \\
e_{3,1} & e_{3,2} & e_{3,3} & \cdots & e_{3,C} \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
e_{k,1} & e_{k,2} & e_{k,3} & \cdots & e_{k,C}
\end{pmatrix}$
Perform exact search of the closest cluster under encryption

Best match:
cluster = 2
doc = 3

Communication: $O(\sqrt{Nd})$, on $N$ docs and embedding length $d$

Server work: fast with SimplePIR (2d 64-bit operations per doc)
Tiptoe: Life of a query

- The client needs three pieces of data: a set of cryptographic data structures required for our private search protocols.
- Jobs compute a set of cryptographic data structures required for our private search protocols.
- The embedding vectors are generated using an embedding function to generate a fixed-size vector representation.
- In a production deployment, many more vectors are used to scale up.
- The client interacts with two tip-toe services over the internet.
- The client uses a new cryptographic protocol to obtain file fetching services and the search corpus.
- The client interacts with one tip-toe service per query.
- The client encrypts the query string and identifies the cluster nearest to its query embedding.
- The client queries the nearest-neighbor service to find the best-matching documents.
- It queries the URL service with its query vector and the nearest-neighbor service finds the documents closest to its query embedding.
- The client uses the nearest cluster to query the URL service.
- The client uses the centroids to find the cluster.
- The client uses the nearest-neighbor service to find the documents closest to its query embedding.
- The client decrypts the answer to obtain the best-matching documents.
### Comparison

Tiptoe is cheaper than state-of-the-art private search.

<table>
<thead>
<tr>
<th></th>
<th>Coeus (SOSP’21)</th>
<th>Tiptoe</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docs searched</td>
<td>5 million</td>
<td>364 million</td>
<td>72 x</td>
</tr>
<tr>
<td>Client storage</td>
<td>-</td>
<td>0.3 GiB</td>
<td></td>
</tr>
<tr>
<td>Server compute</td>
<td>2,580 core-s</td>
<td>0.4 core-s</td>
<td>6,450 x</td>
</tr>
<tr>
<td>Communication</td>
<td>10 MiB</td>
<td>0.16 MiB</td>
<td>62 x</td>
</tr>
<tr>
<td>End-to-end latency</td>
<td>-</td>
<td>2.7 s</td>
<td></td>
</tr>
</tbody>
</table>
Tiptoe is cheaper than state-of-the-art private search

<table>
<thead>
<tr>
<th></th>
<th>Coeus (SOSP’21)</th>
<th>Tiptoe</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docs searched</td>
<td>5 million</td>
<td>364 million</td>
<td>72 x</td>
</tr>
<tr>
<td>Client storage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server compute (per million docs)</td>
<td>2,580 core-s</td>
<td>6,450 x</td>
<td></td>
</tr>
<tr>
<td>Communication (per million docs)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>End-to-end latency</td>
<td>-</td>
<td>2.7 s</td>
<td></td>
</tr>
</tbody>
</table>

- **Semantic embeddings:** 100× smaller doc representations
- **SimplePIR:** 10× less computation
- **Clustering:** communication sublinear in $N$
Tiptoe is cheaper than state-of-the-art private search

<table>
<thead>
<tr>
<th></th>
<th>Coeus (SOSP’21)</th>
<th>Tiptoe</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docs searched</td>
<td>5 million</td>
<td>364 million</td>
<td>72 x</td>
</tr>
<tr>
<td>Client storage</td>
<td>-</td>
<td>0.3 GiB</td>
<td>– ∞ x</td>
</tr>
<tr>
<td>Server compute (per million docs)</td>
<td>2,580 core-s</td>
<td>0.4 core-s</td>
<td>6,450 x</td>
</tr>
<tr>
<td>Communication (per million docs)</td>
<td>10 MiB</td>
<td>0.16 MiB</td>
<td>62 x</td>
</tr>
<tr>
<td>End-to-end latency</td>
<td>-</td>
<td>2.7 s</td>
<td></td>
</tr>
</tbody>
</table>
Tiptoe’s search quality is acceptable

Better search quality (MRR@100)

ColBERT
 BM25
tf-idf
 Tiptoe

Best non-private:
Top result on average ranked 2.3 of 100

Private:
Top result on average ranked 7.7 of 100

On the MS MARCO doc-rank “dev” data set
Examples: Tiptoe works best on conceptual queries

how long before eagles get feathers

the meaning of haploid cell

On the Common crawl data set
... but Tiptoe’s exact-string search could improve on the Common crawl data set
Private search is within reach… what’s next?

Many directions for improvement

- Improve **quality**: run more powerful search under encryption?
- Reduce **cost**: shrink communication? increase throughput?

Many applications of private nearest-neighbor search

- Tiptoe can search over **products**, **ads**, **feeds**, and more
“knee problem”

Tiptoe private search

Attacker (data breach)
SketchyCo (resale)
LearningCo (training)

hopkinsmedicine.org/health/knee-pain

Alexandra Henzinger

Code: github.com/ahenzinger/tiptoe
Paper: eprint.iacr.org/2023/1438
Demo: come talk to me!