Lecture 24: Private Web Search with Tiptoe

Lecturer: Alexandra Henzinger **Date:** May 5, 2026
Scribe: Yanchen Liu

1. Overview

This lecture studied *Tiptoe*, a system for private web search at web scale. The motivating problem is that ordinary web-search queries reveal sensitive information about users: health conditions, financial status, relationships, location, and other personal concerns. In today's search architecture, the search provider directly observes the user's query, may store it, may share or sell derived information, and may use it to train downstream machine-learning models. Tiptoe asks whether we can keep the utility of web search while hiding the user's query from the search engine.

At a high level, Tiptoe combines two ideas:

1. **Semantic embeddings from machine learning** turn semantic document ranking into an inner-product nearest-neighbor problem.
2. **Linearly homomorphic encryption** lets the server compute these inner products while the user's query remains encrypted.

Main idea

Instead of privately evaluating an entire modern search engine, Tiptoe changes the representation of search. Semantic embeddings reduce private full-text search to private nearest-neighbor search, and private nearest-neighbor search can be implemented using fast encrypted matrix-vector multiplication.

This is an interesting inversion of a common theme in cryptography and machine learning. Often, cryptography is used to secure ML workloads. In Tiptoe, ML representations make a new cryptographic system practical: embeddings convert semantic retrieval into a form that is friendly to linear cryptographic computation.

2. Motivation: Why Private Search?

Search queries can be highly sensitive. A query such as "MRI cost," "how do I know if I have diabetes," or "why did my credit score drop" may reveal information about the user's health, finances, or life circumstances. The lecture emphasized that this is not merely a hypothetical privacy concern. Query logs can be exposed in data breaches; companies may use search or browsing data to train ML models; and third parties such as advertisers or insurers may infer sensitive attributes from browsing behavior.

The ideal goal is a private search engine. The client should send an encryption of its query, the server should compute an encrypted response, and the server should learn nothing about the query or the result. Informally, the interaction transcript should look like random noise to the server.

Definition 2.1 (Informal private-search privacy)**.** A private search protocol protects query privacy if, for any two possible queries q_0 and q_1, the server's view when the user searches for q_0 is computationally indistinguishable from the server's view when the user searches for q_1.

The lecture also clarified several limitations that are outside this privacy goal:

- **Timing is not hidden.** The protocol does not hide when a user makes a query.
- **Result integrity is not guaranteed.** A malicious server could return bogus or incomplete results.
- **Follow-up browsing may leak information.** If the user clicks a returned URL, the subsequent HTTP(S) request may reveal information outside the private-search protocol.

Thus, the model protects the private interaction between the user and the search service; it is not a complete anonymous-browsing system.

3. Prior Approaches and Their Limitations

Prior work on private search falls into two broad categories: systems with relaxed privacy guarantees and systems with strong cryptographic privacy but limited scalability.

3.1. Relaxed privacy guarantees

One approach is to anonymize queries using a proxy, VPN, or Tor-like system. This may hide the user's network identity from the search provider, but the provider still observes the raw query. Moreover, repeated or semantically related queries may allow the provider to link requests and de-anonymize users.

A second approach is query obfuscation: the user sends fake queries alongside the real query. For example, a sensitive medical query might be mixed with unrelated queries about food or entertainment. This provides only heuristic protection. A sufficiently strong search provider may identify the real query from context, similarity, or user history.

A third approach is trusted hardware, such as Intel SGX. This can reduce the trusted computing base, but it shifts security to assumptions about hardware isolation and remains vulnerable to side channels or implementation attacks. These systems do not give the same clean cryptographic guarantee as semantic security.

3.2. Strong cryptographic privacy at limited scale

A stronger approach is to use cryptography to ensure that the search engine learns nothing about the user's query. Prior systems such as Coeus [1] pursue this direction. A natural baseline is to implement a classic keyword-search algorithm such as TF-IDF under encryption.

Let N be the number of documents and W be the vocabulary size. The server can represent the corpus as a term-document matrix

$$\mathbf{X} \in \mathbb{Z}^{N \times W},$$

where X_{ij} counts how often word j appears in document i. The client represents its query as a sparse vector

$$\boldsymbol{v} \in \{0,1\}^{W},$$

where $v_j = 1$ if word j appears in the query. The relevance scores are then

$$\mathbf{X}\boldsymbol{v} \in \mathbb{Z}^{N}.$$

If the client sends $\mathsf{Enc}(\boldsymbol{v})$, a linearly homomorphic encryption scheme lets the server compute

$$\mathbf{X} \cdot \mathsf{Enc}(\boldsymbol{v}) = \mathsf{Enc}(\mathbf{X}\boldsymbol{v}).$$

The client decrypts the resulting score vector and selects the highest-scoring document.

This is conceptually simple and private, but it scales poorly. The client must upload an encrypted vector of length W and download an encrypted score for every document. The server must essentially perform work proportional to the number of word-document pairs. For web-scale corpora, the matrix is enormous, so per-query computation and communication become impractical.

Bottleneck

Cryptography can already express private search in principle. The obstacle is not expressiveness but efficiency: directly evaluating old-style keyword search under encryption produces costs that grow with the full vocabulary and the full corpus.

4. First Insight: Semantic Embeddings Linearize Search

Tiptoe departs from the TF-IDF view of search. Instead of representing documents by high-dimensional sparse word-count vectors, it uses semantic embeddings. An embedding model maps a query or document to a dense vector

$$\boldsymbol{e} \in \mathbb{R}^{d},$$

where semantically related inputs are close in vector space. In the lecture, one can think of $d \approx 192$, and similarity is measured by inner product. When vectors are normalized, inner product similarity is equivalent to cosine similarity.

Let $\boldsymbol{q} \in \mathbb{R}^d$ be the query embedding and let $\boldsymbol{e}_i \in \mathbb{R}^d$ be the embedding of document i. Ranking can be approximated by

$$\arg\max_{i \in [N]} \langle \boldsymbol{q}, \boldsymbol{e}_i \rangle.$$

This turns private web search into private nearest-neighbor search.

Linearization

Embeddings *linearize* semantic search. A vague semantic question such as "which document best answers this query?" becomes a dot-product computation, which is exactly the kind of operation that linearly homomorphic encryption can support efficiently.

This gives two benefits. First, embeddings are more expressive than exact keyword matching: they can capture synonyms and semantic similarity. Second, embeddings are compact and general. The same framework can apply not only to text but also to images, code, audio, or other modalities for which embedding models exist.

There is a useful separation between *quality* and *privacy*. The embedding model is responsible for search quality, and it does not provide worst-case semantic guarantees. The cryptography is responsible for query privacy, and it provides a formal security guarantee under lattice assumptions.

5. Private Nearest-Neighbor Search

After embedding the corpus, the server holds document vectors

$$\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_N \in \mathbb{R}^d.$$

The client locally embeds its query to obtain $\boldsymbol{q} \in \mathbb{R}^d$. The ideal search objective is

$$i^* = \arg\max_{i \in [N]} \langle \boldsymbol{q}, \boldsymbol{e}_i \rangle,$$

without revealing $\boldsymbol{q}$ to the server.

If the server stores all embeddings in a matrix

$$\mathbf{E} = \begin{bmatrix} \boldsymbol{e}_1^\top \\ \boldsymbol{e}_2^\top \\ \vdots \\ \boldsymbol{e}_N^\top \end{bmatrix} \in \mathbb{R}^{N \times d},$$

then all similarity scores are

$$\mathbf{E}\boldsymbol{q} = \begin{bmatrix} \langle \boldsymbol{e}_1, \boldsymbol{q} \rangle \\ \langle \boldsymbol{e}_2, \boldsymbol{q} \rangle \\ \vdots \\ \langle \boldsymbol{e}_N, \boldsymbol{q} \rangle \end{bmatrix}.$$

Thus private nearest-neighbor search reduces to private matrix-vector multiplication. The client encrypts $\boldsymbol{q}$, the server computes an encryption of $\mathbf{E}\boldsymbol{q}$, and the client decrypts the scores.

This is already much more compact than the TF-IDF formulation because d is small compared with the vocabulary size W. However, computing scores for all N documents is still too expensive for a web-scale corpus. Tiptoe therefore adds a second structural idea: clustering.

6. Second Insight: Cluster the Embedding Space

Tiptoe organizes document embeddings into clusters. In preprocessing, the server clusters the N document vectors, for example using an off-the-shelf algorithm such as k-means. Let there be C clusters with centroids

$$\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_C \in \mathbb{R}^d.$$

The client downloads and stores these centroids locally. For a query embedding $\boldsymbol{q}$, the client first computes the closest centroid:

$$j^* = \arg\max_{j \in [C]} \langle \boldsymbol{q}, \boldsymbol{c}_j \rangle.$$

Then the client performs an exact private search only within cluster j^*.

The privacy subtlety is that the cluster identity must also remain hidden. If the server learns that the user is searching cluster j^*, and if that cluster corresponds to medical or financial documents, then the server has already learned information about the query.

To hide both the query embedding and the chosen cluster, Tiptoe stores the clustered document embeddings in a matrix. Conceptually, column j contains the embeddings of documents in cluster j. The client constructs an encrypted selection vector that is zero everywhere except at the chosen cluster position, where it contains the query embedding $\boldsymbol{q}$. The server multiplies its matrix by this encrypted vector. The result is an encrypted list of relevance scores for documents in the selected cluster, but the server does not learn which cluster was selected.

6.1. Balancing the matrix

The communication is minimized when the matrix used in the private matrix-vector multiplication is close to square. If there are N documents and each embedding has dimension d, Tiptoe chooses the number and size of clusters to balance the matrix dimensions. At a high level, this yields communication scaling like

$$O(\sqrt{N})$$

instead of requiring the client to download a score for every document.

Tradeoff

Clustering trades exact global nearest-neighbor search for a much cheaper local search. The system first identifies a promising cluster locally, then privately searches only within that cluster while hiding the cluster identity from the server. Searching more nearby clusters can improve quality but increases communication and server work.

7. Cryptographic Core: Fast Private Matrix Multiplication

The remaining bottleneck is private matrix-vector multiplication. Tiptoe uses the SimplePIR line of techniques [2], based on linearly homomorphic encryption. The lecture described the relevant idea through Regev encryption [3].

7.1. Linearly homomorphic encryption

A linearly homomorphic encryption scheme supports applying a public linear transformation to encrypted data. If the client encrypts a vector $\boldsymbol{v}$, then for a public matrix $\mathbf{M}$ the server can compute

$$\mathbf{M} \cdot \mathsf{Enc}(\boldsymbol{v}) = \mathsf{Enc}(\mathbf{M}\boldsymbol{v}).$$

This is exactly the primitive needed for private matrix-vector multiplication.

In Regev-style encryption, a ciphertext for vector $\boldsymbol{v}$ can be viewed as having two parts

$$(\mathbf{A}, \boldsymbol{b}),$$

where $\mathbf{A}$ is a large random matrix and $\boldsymbol{b}$ is the message-dependent component. The important facts for this lecture are:

1. Regev encryption is linearly homomorphic: applying $\mathbf{M}$ to both ciphertext components yields an encryption of $\mathbf{M}\boldsymbol{v}$.
2. Decryption is linear in the secret key. Informally, decrypting requires $\mathbf{A}$ only through the product of $\mathbf{A}$ with the secret key.

A naive use of Regev encryption is expensive because the random matrix $\mathbf{A}$ has dimension proportional to the security parameter, say $\lambda \approx 2048$. Multiplying by the full ciphertext can therefore introduce a large overhead over plaintext matrix-vector multiplication.

7.2. SimplePIR-style preprocessing

The key SimplePIR insight is that the random matrix $\mathbf{A}$ is independent of the encrypted message. Therefore, $\mathbf{A}$ can be fixed across queries. If the server's matrix is $\mathbf{M}$, then the server can precompute

$$\mathbf{MA}$$

once ahead of time. During the online query phase, the server only needs to compute

$$\mathbf{M}\boldsymbol{b},$$

where $\boldsymbol{b}$ is the compact, message-dependent part of the user's ciphertext.

Thus most of the expensive work moves to preprocessing. Online, private matrix-vector multiplication becomes close to ordinary matrix-vector multiplication. The lecture summarized this as follows: without privacy, the server might perform n^2 operations on 16-bit values; with the optimized private protocol, it performs comparable n^2 work on larger, e.g., 64-bit, values. The privacy overhead is therefore mostly in word size rather than in an extra multiplicative factor in the number of matrix operations.

7.3. Avoiding large client downloads

A subtle issue is that the precomputed value $\mathbf{MA}$ may be large. If clients had to download it frequently, this would be problematic, especially because the web corpus changes over time.

Tiptoe uses an additional trick inspired by fully homomorphic encryption: rather than sending the entire preprocessed object to the client, the client uploads an encryption of its secret key under another encryption scheme. The server can then compute the part needed for decryption under encryption. This replaces a large download with smaller upload/download costs while preserving the asymptotic benefit of preprocessing.

The result is a protocol for multiplying a public matrix by an encrypted vector at nearly the speed of unencrypted matrix multiplication, after preprocessing.

8. End-to-End Tiptoe Architecture

Tiptoe can be understood as three stacked layers:

1. **Private document ranking.** The application-level goal is to return relevant search results privately.
2. **Private nearest-neighbor search.** Semantic embeddings reduce ranking to inner-product search over vectors.
3. **Private matrix multiplication.** Linearly homomorphic encryption implements the required inner-product computations under encryption.

The system has an offline preprocessing phase and an online query phase.

8.1. Offline preprocessing

The server performs batch preprocessing over the public corpus:

1. embed each document using a semantic embedding model;
2. cluster the document embeddings;
3. build the search-index matrix whose entries encode clustered embeddings;
4. perform cryptographic preprocessing, such as computing $\mathbf{MA}$ for the matrix used in private matrix-vector multiplication.

This preprocessing can be expensive, but it is amortized across many queries.

8.2. Online query phase

For a user query, the online protocol is:

1. The client embeds the query locally, obtaining $\boldsymbol{q} \in \mathbb{R}^d$.
2. The client compares $\boldsymbol{q}$ to locally stored cluster centroids and chooses the closest cluster.
3. The client constructs an encrypted query vector that hides both $\boldsymbol{q}$ and the chosen cluster.
4. The server uses private matrix multiplication to compute encrypted relevance scores for documents in the hidden cluster.

5. The client decrypts the scores and selects the top result(s).

This protocol reveals neither the raw query nor the selected cluster to the server.

9. Evaluation

The paper and lecture evaluate Tiptoe at web scale. On a public web crawl of roughly 360 million pages, Tiptoe can privately search with about 145 core-seconds of server computation and about 57 MiB of client-server communication per query. The reported latency is under three seconds when the computation is parallelized across a cluster.

Metric	Reported scale / result
Corpus size	~360 million web pages
Server computation	~145 core-seconds per query
Communication	~57 MiB per query
Latency	under 3 seconds with parallelization
Search quality	best result appears in top 10 on average
Privacy basis	LWE/RLWE-based cryptographic assumptions

Table 1: High-level evaluation results for Tiptoe.

The search quality is not yet at the level of the best non-private search engines such as Google or Bing. The lecture described the quality as placing the best result in the top 10 on average. The main quality losses come from the embedding model and from searching only a limited number of clusters. Searching more clusters improves quality but increases cost.

10. Discussion

10.1. Why embeddings and cryptography fit together

The central conceptual lesson is that embeddings and cryptography are a surprisingly good match. Search is normally a complex, heuristic, and implementation-heavy task. Directly compiling a full search engine into fully homomorphic encryption would be far too expensive. Embeddings avoid this by changing the problem: semantic ranking becomes a sequence of linear operations.

This separation is powerful:

- ML provides quality by mapping messy inputs into useful vector representations.
- Cryptography provides privacy by allowing linear operations over encrypted vectors.

Tiptoe is therefore not simply "Google under encryption." It is a redesigned search architecture whose representation is chosen to make cryptographic privacy feasible.

10.2. Main tradeoffs

Tiptoe makes several important tradeoffs:

- **Quality vs. cost.** Searching more clusters improves recall but increases communication and server work.
- **Privacy vs. integrity.** The system hides the query but does not guarantee that the server returns correct or complete results.
- **Semantic quality vs. formal guarantees.** The cryptography gives formal privacy guarantees, but the embedding model gives empirical search-quality guarantees.
- **Preprocessing vs. freshness.** Heavy preprocessing makes online search fast, but the web changes over time, so the index must be updated.

10.3. Broader applications

The lecture concluded by noting that private nearest-neighbor search is useful beyond web search. Similar techniques could support private retrieval-augmented generation, private recommendation systems, private image or code search, and other vector-database applications. As more ML systems rely on embedding search, efficient private nearest-neighbor search may become a general building block for privacy-preserving ML services.

11. Summary

Tiptoe shows that web-scale private search is feasible by combining semantic embeddings with efficient cryptography. The system proceeds in three reductions:

$$\begin{aligned}&\text{private web search}\\ &\quad\longrightarrow \text{private nearest-neighbor search}\\ &\quad\longrightarrow \text{private matrix-vector multiplication.}\end{aligned}$$

Embeddings make the search computation linear; clustering reduces the amount of data that must be searched per query; and SimplePIR-style preprocessing makes encrypted matrix multiplication fast enough to be practical.

The resulting system is not a drop-in replacement for modern search engines in quality, but it demonstrates a major systems point: with the right representation, cryptographic privacy can scale to workloads that previously seemed out of reach.

References

[1] Taha Ahmad et al. Coeus: A system for oblivious document ranking and search. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2021.

[2] Alexandra Henzinger, Ke Wu Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. Simplepir: Simple and fast private information retrieval. In *USENIX Security Symposium*, 2023.

[3] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing (STOC)*, 2005.