Privacy and Efficiency Tradeoffs for Multiword Top K Search with Linear Additive Rank Scoring

Daniel Agun, Jinjin Shao, Shiyu Ji, Stefano Tessaro, Tao Yang

Department of Computer Science
University of California at Santa Barbara
Open Problem for Private Search

- **Privacy challenge on the cloud**
  - Client offloads data to the cloud, and wants to exploit cloud computing resource
  - Server is honest-but-curious: correctly executes protocol but observes/infers private information

- **Privacy requirement**: Store client-owned data on the cloud, and have the free-text keyword search on the data, without leaking the plaintext
  - Open problem: how to design and implement efficient private ranking for multi-keyword search?
Top K Search Problem Definition

• Given a set of documents feature vectors
  ▪ Each document $d$ has many encrypted features denoted as $E(f_i^d)$

• Indexing and top K search scheme so that
  ▪ Server can access encrypted document features
  ▪ Rank them within a reasonable response time without knowing underlying feature values
  ▪ E.g. $\text{RankScore}(E(f_1^{d1}), E(f_2^{d1}))$ vs $\text{RankScore}(E(f_1^{d2}), E(f_2^{d2}))$

Cloud server

Client

Encrypted query keywords

Ranked results
Our Approach and Contributions

- Private ranking scheme with linear additive scoring for efficient top K keyword search
- Support modest sized cloud datasets –
  - Bigger dataset requires faster internet connection between server and client (or trusted client-proxy)
- Strike for tradeoffs between privacy and efficiency
- Single-round client-server collaboration
- Server-side partial ranking using blinded feature weights with random masks to reduce result size

![Diagram showing the flow of encrypted query keywords from client to cloud server, with partial ranking, and finally ranked results to client.](image-url)
Design Considerations

• Additive Linear Ranking Formula
  ▪ Weighted liner combination of features: \( Score = \sum \alpha_t f_t^d \)
  ▪ Simplify to \( \sum f_t^d \) by embedding \( \alpha_t \) in feature

• Ranking features that can be accommodated
  ▪ Term-frequency based (TFIDF, BM25)
  ▪ Proximity composite (word pair distance)
  ▪ Document-query specific (click-through rate)
  ▪ Document-specific (freshness, quality)

• Handling sparsity of raw ranking features
  ▪ Explicit storage with uniform representation
    – too expensive
  ▪ Separate required and optional features
    – Handling of optional features without leakage is a challenge:
• Previous work
  - TFIDF-based query/document dot product
    - Multiply a query vector and document with a matrix
    - Unscable even for small datasize: prohibitive search cost for datasets over a few thousand documents/terms

• Homomorphic Encryption – still not practical for reasonable response time, no efficient comparison

• Order Preserving Encryption – does not support arithmetic operations

• Searchable encryption – does not address ranking

• Multi-round client-server communication – slow

• Our solution: Feature encryption with mask blinding
  - Encrypt feature $E(f_i^d) = f_i^d + R_i^d \mod N$, $R_i^d$ is random mask
Ordering Masked Rank Scores Without Knowing Rank Values

- **Scoring function** – linear sum of features
  - Separate required features and optional features
    - handle feature sparsity and retain space efficiency
  - Blinded score:
    - \((\text{Sum of features} + \text{sum of offsets}) \mod N\)

- **How can server order two documents without knowing real scores with wraparound from mod?**
  - **Theorem:**
    - If blinded score difference of two documents is \(< N/2\),
      order of unblinded scores = order of blinded score
    - Otherwise order is reversed
  - **Requirement:** same mask and unblinded score \(< N/2\)
Server-Side Partial Ranking

- **Per-document random masks**
  - Stored feature: $f_i^d + R_i^d \mod N$
  - Completely private, server cannot rank

- **Chunk-wide random masks**
  - Stored feature: $f_i^d + R_i^d + R_c^i \mod N$, where $c$ is the posting chunk of term $i$
  - Query-dependent deblinding
    - Server only able to remove $R_i^d$ when client sends it
    - Only leak feature difference within a chunk to server when such a word is searched and partial ranking is triggered

- **Term posting size restriction**
  - Only trigger partial ranking when length > 10000
Query Decomposition and Subquery Handling

Query Decomposition

- **Query:** cd rate
- **Client-side earlier intersection to generate subqueries:**
  - CD₁ rate₂, CD-rate₁
  - CD₃ rate₃, CD-rate₁
  - CD₄ rate₄
  - CD₄ rate₅

For each subquery, compare documents within each optional matching case
- Incomparable across subqueries
- Maximize comparable documents among optional feature matching cases by exploiting their lattice relationship
Indexing and Runtime Processing for Conjunctive Queries

- Adopt document matching algorithm from Cash et al. (CRYPTO’13) for secure intersection
- Support feature blinding with dynamic chunk-wide random masking: prevent server from learning about features if such a word is not searched/not triggered for partial ranking
- **3 key-value stores for encrypted inverted index setup**
  - R-store saves meta information in feature posting chunks such as document ID range of chunks: facilitates query decomposition at the client side
  - S-store contains required feature values and is used by the search algorithm to identify the candidate documents
  - X-store contains feature values accessible using a pair of document ID and feature ID
S-store and X-Store Setup

• S-store
  ▪ Key is called \( stag \) used for starting search
    – Based on word ID and chunk ID
    – Formally \( stag = PRF(k_7, w \| c) \)
  ▪ Value is a chunk list of posting entries and each posting entry is an encrypted tuple \((e, y, f_s)\)
    – Encrypted document ID \( e \)
    – Blinded bridging number \( y \) to enable client-authorized X-store key derivation
    – Blinded feature \( f_s \)

• X-store
  ▪ Key called \( xtag \) used as hash table key for intersection
    – Key is based on word ID and doc ID that contains this word
    – Formally \( g^{PRF(k_5, w)PRF(k_2, d)} \)
  ▪ Value is encrypted feature value
    – Formally \( X(xtag) = f_i^d + R_i^d + R_i^c \mod N \)
• **Phase 1 – client side**
  - Form required and optional features; derive subqueries with earlier intersection
  - Form encrypted tokens including $stag$ for each subquery

• **Phase 2 – server side**
  - Use client-$stag$ to access S-store and fetch posting chunks
  - Dynamically compute client-authorized $xtag$ to access features from X-store
  - Perform server-side partial ranking if authorized

• **Phase 3 – client side**: Remove random mask for final ordering
Properties of Search Time and Privacy

• **Search Complexity**
  - Index space: proportional to all non-zero features
  - Search time: $O((n - 1) \sum |Posting(w_1)|)$ for all subqueries with $n$ required/optional features

• **Privacy properties**
  - Theorem 4.1: If feature has not been used in any search query, the server cannot learn corresponding weight for any document.
  - Theorem 4.2: The server cannot learn document feature weights for any unpopular word (<10k docs in its posting) during or after query processing.
  - Also true for any popular word which has only been involved in searches with at least one unpopular word.
Implementation and Evaluation of Private Search

- Prototype built in C++
- Evaluation on Linux Ubuntu 16.04 servers with 8 cores and 2.4GHz AMD FX8320, 16GB RAM

### Dataset size characteristics

<table>
<thead>
<tr>
<th>Dataset</th>
<th>CSIRO</th>
<th>TREC45</th>
<th>Aquaint</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Doc</td>
<td>0.37M</td>
<td>0.53M</td>
<td>1.03M</td>
</tr>
<tr>
<td>word-doc</td>
<td>22M</td>
<td>109M</td>
<td>216M</td>
</tr>
<tr>
<td>wordpair-doc</td>
<td>146M</td>
<td>712M</td>
<td>1,357M</td>
</tr>
<tr>
<td>R-Store</td>
<td>0.31GB</td>
<td>1.25GB</td>
<td>1.13GB</td>
</tr>
<tr>
<td>S-Store</td>
<td>1.12GB</td>
<td>5.56GB</td>
<td>11.02GB</td>
</tr>
<tr>
<td>X-Store</td>
<td>2.42GB</td>
<td>11.82GB</td>
<td>22.53GB</td>
</tr>
<tr>
<td>Total Size</td>
<td>3.85GB</td>
<td>18.63GB</td>
<td>34.68GB</td>
</tr>
</tbody>
</table>

### Query processing costs

<table>
<thead>
<tr>
<th># Query words q</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>30.53</td>
<td>59.98</td>
<td>74.89</td>
<td>101.16</td>
</tr>
<tr>
<td>S-store</td>
<td>58.31</td>
<td>121.57</td>
<td>140.40</td>
<td>37.60</td>
</tr>
<tr>
<td>X-store</td>
<td>0</td>
<td>59.21</td>
<td>137.87</td>
<td>64.09</td>
</tr>
<tr>
<td>Total(ms)</td>
<td>89.62</td>
<td>283.86</td>
<td>427.98</td>
<td>248.51</td>
</tr>
</tbody>
</table>

| Client          | 18.89| 45.18| 65.56| 107.22|
| S-Store         | 146.79| 191.30| 222.33| 119.67|
| X-Store         | 0    | 85.56| 284.15| 260.11|
| Total(ms)       | 166.42| 405.64| 717.34| 693.00|

### Cost increases with more query words and optional features

Overall query response time is reasonable (<1s)
Effectiveness of Server Partial Ranking

Return result reduction in top-10 search with different chunk sizes
Threshold to trigger partial ranking: 10,000+ results

<table>
<thead>
<tr>
<th>#Results returned</th>
<th>TREC Queries</th>
<th>Synthetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSIRO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No filter</td>
<td>11,607</td>
<td>33,093</td>
</tr>
<tr>
<td>Chunk 105</td>
<td>2,504</td>
<td>8,884</td>
</tr>
<tr>
<td>Chunk 210</td>
<td>1,288</td>
<td>6,159</td>
</tr>
<tr>
<td>TREC45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No filter</td>
<td>20,985</td>
<td>127,538</td>
</tr>
<tr>
<td>Chunk 105</td>
<td>14,151</td>
<td>28,876</td>
</tr>
<tr>
<td>Chunk 210</td>
<td>8,708</td>
<td>20,596</td>
</tr>
<tr>
<td>Aquaint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No filter</td>
<td>32,896</td>
<td>185,139</td>
</tr>
<tr>
<td>Chunk 105</td>
<td>25,561</td>
<td>38,333</td>
</tr>
<tr>
<td>Chunk 210</td>
<td>16,112</td>
<td>22,437</td>
</tr>
</tbody>
</table>

- Server-side partial ranking reduces network costs
  - Reduces returned result set significantly
    - For Aquaint, server filters out 88% of matched results
    - Cost 0.39 sec to deliver remaining 22K results on 7.2Mbps internet connection

Synthetic queries: Stop/popular words with high match size
### Evaluation of Ranking Relevance

<table>
<thead>
<tr>
<th></th>
<th>NDCG@10</th>
<th>L=2</th>
<th>L=5</th>
<th>L=∞</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CSIRO</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LambdaMART</td>
<td>0.4836</td>
<td>0.4842</td>
<td>0.4789</td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>0.4311</td>
<td>0.4046</td>
<td>0.4317</td>
<td></td>
</tr>
<tr>
<td><strong>TREC45</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LambdaMART</td>
<td>0.3823</td>
<td>0.3898</td>
<td>0.3866</td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>0.4121</td>
<td>0.4012</td>
<td>0.3808</td>
<td></td>
</tr>
<tr>
<td><strong>Aquaint</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LambdaMART</td>
<td>0.3256</td>
<td>0.3246</td>
<td>0.3131</td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>0.3118</td>
<td>0.3227</td>
<td>0.317</td>
<td></td>
</tr>
</tbody>
</table>

- **Restriction on optional term distance** $L$ has small impact on relevance
  - Restrict optional word distance pairs
  - Less optional features, more comparable documents, faster response time
## Impact of Growing Dataset Size

ClueWeb09 Category B dataset with 50 million web documents

Return result sizes in top-10 search with partial server-ranking triggering threshold 10,000, varying index size from 3M to 50M

<table>
<thead>
<tr>
<th># Results returned</th>
<th>3M</th>
<th>5M</th>
<th>10M</th>
<th>30M</th>
<th>50M</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Filter</td>
<td>65,449</td>
<td>81,445</td>
<td>113,173</td>
<td>218,027</td>
<td>321,769</td>
</tr>
<tr>
<td>Chunk 105</td>
<td>25,273</td>
<td>31,866</td>
<td>46,442</td>
<td>90,138</td>
<td>134,240</td>
</tr>
<tr>
<td>Chunk 210</td>
<td>19,992</td>
<td>25,394</td>
<td>37,413</td>
<td>73,017</td>
<td>108,892</td>
</tr>
<tr>
<td>Top 10%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Filter</td>
<td>393,650</td>
<td>506,452</td>
<td>752,328</td>
<td>1,619,203</td>
<td>2,465,084</td>
</tr>
<tr>
<td>Chunk 105</td>
<td>97,710</td>
<td>147,156</td>
<td>234,872</td>
<td>508,292</td>
<td>812,515</td>
</tr>
<tr>
<td>Chunk 210</td>
<td>69,056</td>
<td>107,376</td>
<td>173,195</td>
<td>374,309</td>
<td>608,736</td>
</tr>
</tbody>
</table>

- Take 0.46sec on average to send over internet at 7.2Mbps with chunk size of 210 for 5 million docs
- Sending top 10% largest result sizes needs 1.93sec with today’s average Internet connection (7.2Mbps)
  - With 5G mobile connection (490Mbps), only take 28millisecond
  - Also ideal for client-trusted proxy-server setting
Leakage Profile

- **Size patterns**
  - Chunk sizes
  - Count of matching documents

- **Rank and feature patterns**
  - Rank score and feature value difference within chunks when used in partial ranking

- **Intersection patterns**
  - Overlapping pattern of s-tags and encrypted tokens sent during search (intersection results)
  - Identification of subqueries sharing startup term (repeated start term)
  - S-term intersections from two subqueries sharing at least one x-term
Conclusions

• Contributions of this Work
  ▪ Private search with support for linear ranking scores
    – Server-side ranking substantially reduces result size
    – Still requires final ranking at client side
  ▪ A solution with tradeoff for this open private search ranking problem
  ▪ Prototype system implementation and evaluation

• Future Work
  ▪ Address Server client communication bottleneck
    – Less of a problem with high speed internet
    – Client trusted proxy
  ▪ Support more advanced ranking techniques
    – Our SIGIR 2018 paper for private search with tree-ensembles