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

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

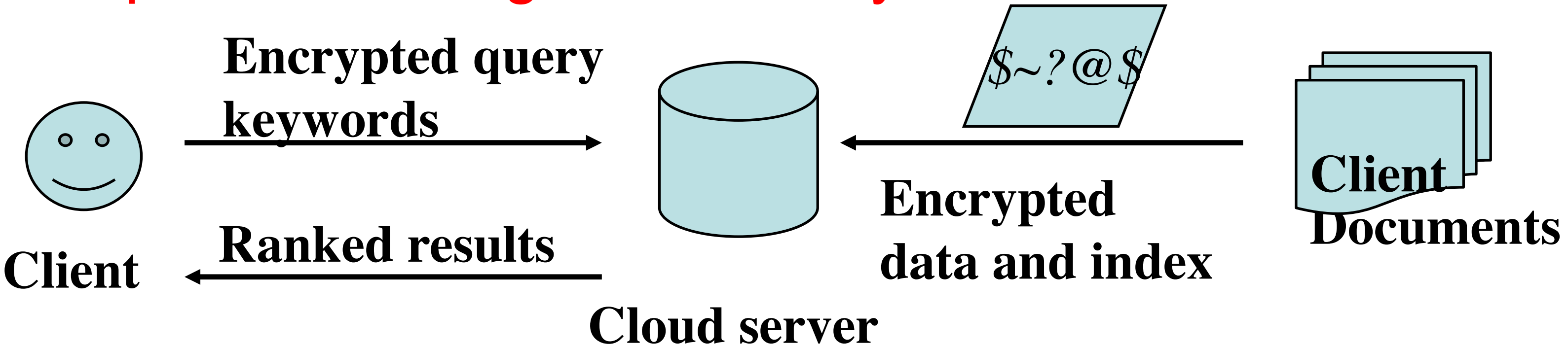
Department of Computer Science

University of California at Santa Barbara

UCSB

Open Problem for Private Search

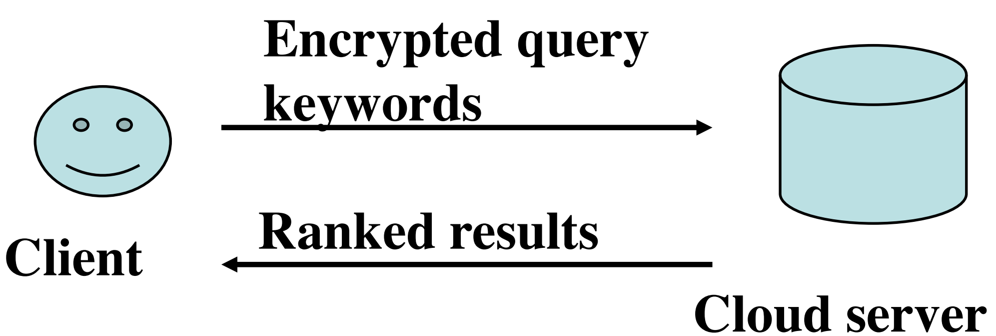
- **Privacy challenge on the cloud**
  - Client offloads data to the cloud, and wants to exploit cloud computing resource
  - Server is honest-but-curious: correctly executes protocol but observes/infers private information
- **Privacy requirement: Store client-owned data on the cloud, and have the free-text keyword search on the data, without leaking the plaintext**
  - Open problem: how to design and implement efficient private ranking for multi-keyword search?



UCSB

Top K Search Problem Definition

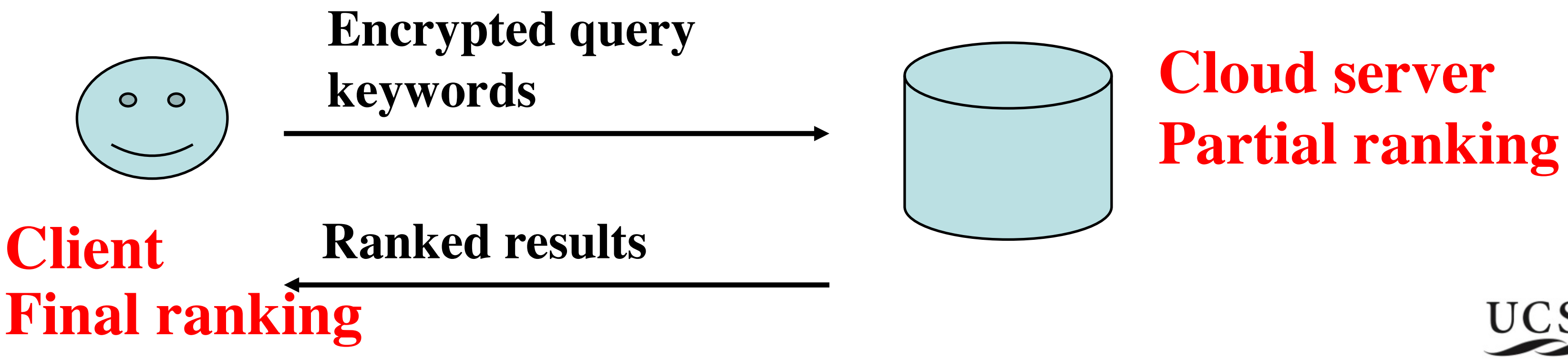
- **Given a set of documents feature vectors**
  - Each document d has many encrypted features denoted as E(f_i^d)
- **Indexing and top K search scheme so that**
  - Server can access encrypted document features
  - Rank them within a reasonable response time without knowing underlying feature values
  - E.g. RankScore(E(f_1^{d1}), E(f_2^{d1})) vs RankScore(E(f_1^{d2}), E(f_2^{d2}))



UCSB

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



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 α_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:

UCSB

Design Considerations – Private Features

- **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

UCSB

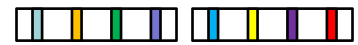
Ordering Masked Rank Scores Without Knowing Rank Values

- **Scoring function – linear sum of features**
  - Separate required features and optional features
    - handle feature sparsity and retain space efficiency
  - Blinded score:
    - (Sum of features + 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

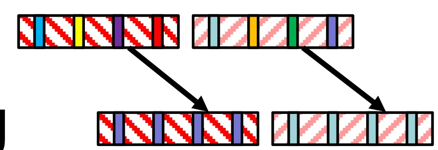
UCSB

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: $f_i^d + R_i^d + R_i^c \; mod \; N$, 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

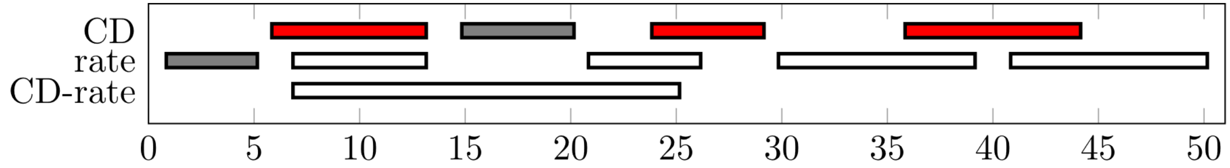
Query Decomposition and Subquery Handling

Query Decomposition

- Query: cd rate
- Client-side earlier intersection to generate subqueries:
  - CD_1 $rate_2$, $CD\text{-}rate_1$
  - CD_3 $rate_3$, $CD\text{-}rate_1$
  - CD_4 $rate_4$
  - CD_4 $rate_5$

Chunked postings

CD	6 7 8 10 13	15 16 18 19 20	24 25 27 28 29	36 40 41 44	
rate	1 2 3 4 5	7 9 10 12 13	21 22 23 25 26	30 31 33 36 39	41 46 48 49 50
CD-rate	7 25				

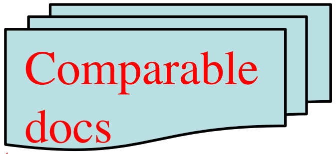


For each subquery, compare documents within each optional matching case

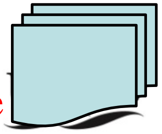
Incomparable across subqueries

Maximize comparable documents among optional feature matching cases by exploiting their lattice relationship

$f_{CD} + f_{Rate}$
$+R_{CD} + R_{Rate}$



$f_{CD} + f_{Rate} + f_{CD\text{-}rate}$
$+R_{CD} + R_{Rate} + R_{CD\text{-}rate}$



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

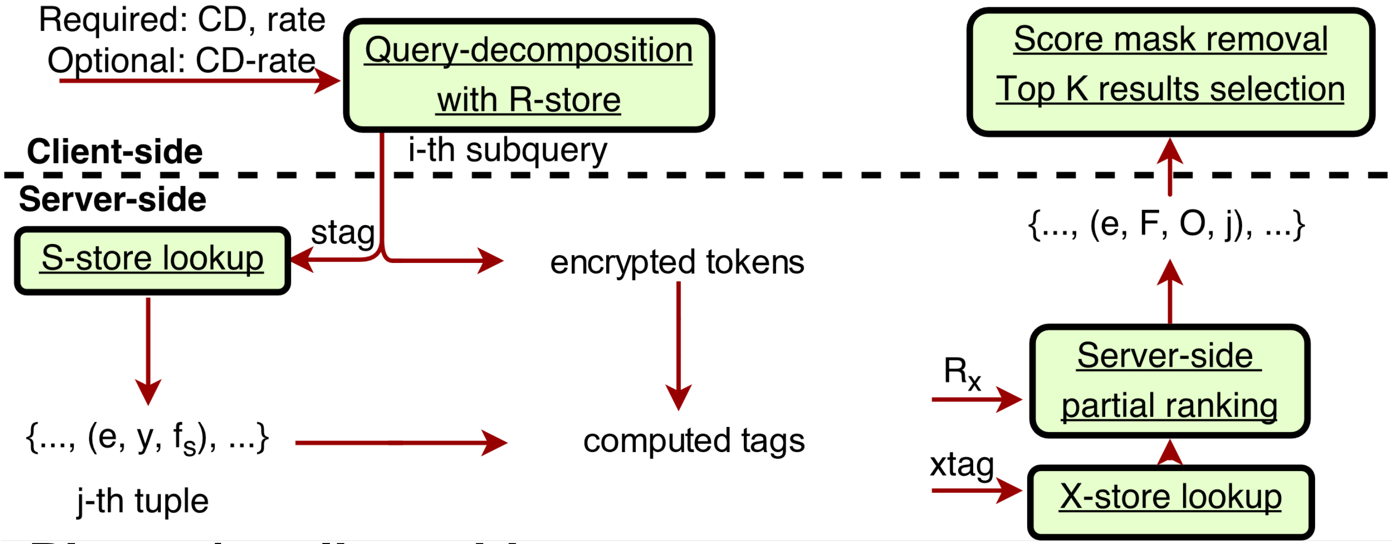
UCSB

S-store and X-Store Setup

- **S-store**
  - Key is called $stag$ used for starting search
    - Based on word ID and chunk ID
    - Formally $stag = PRF(k_7, w \parallel c)$
  - 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 $X(xtag) = f_i^d + R_i^d + R_i^c \; mod \; N$

UCSB

Query Processing Flow & Example



- **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

UCSB

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.

UCSB

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

Dataset size characteristics

# Query words q		1	2	3	4-5
CSIRO	Client	30.53	59.98	74.89	101.16
	S-store	58.31	121.57	140.40	37.60
	X-store	0	59.21	137.87	64.09
	Total(ms)	89.62	283.86	427.98	248.51
TREC45	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
Aquaint	Client	25.87	47.38	54.17	66.35
	S-store	261.81	147.60	146.67	90.60
	X-store	0	89.47	218.52	200.95
	Total(ms)	289.35	337.06	496.20	400.26

Query processing costs

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

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

Synthetic queries: Stop/popular words with high match size

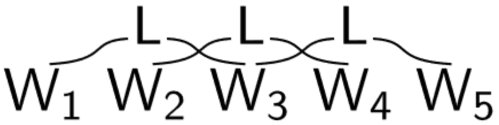
- **Server-side partial ranking reduces network costs**
  - Reduces returned result set significantly
    - For Aquaint, server filters out 88% of matched results
    - Cost 0.39 sec to deliver remaining 22K results on 7.2Mbps internet connection

UCSB

Evaluation of Ranking Relevance

NDCG@10		L=2	L=5	L=∞
CSIRO	LambdaMART	0.4836	0.4842	0.4789
	Linear	0.4311	0.4046	0.4317
TREC45	LambdaMART	0.3823	0.3898	0.3866
	Linear	0.4121	0.4012	0.3808
Aquaint	LambdaMART	0.3256	0.3246	0.3131
	Linear	0.3118	0.3227	0.317

- **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

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

- 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 28millisec
  - Also ideal for client-trusted proxy-server setting

UCSB

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

UCSB

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