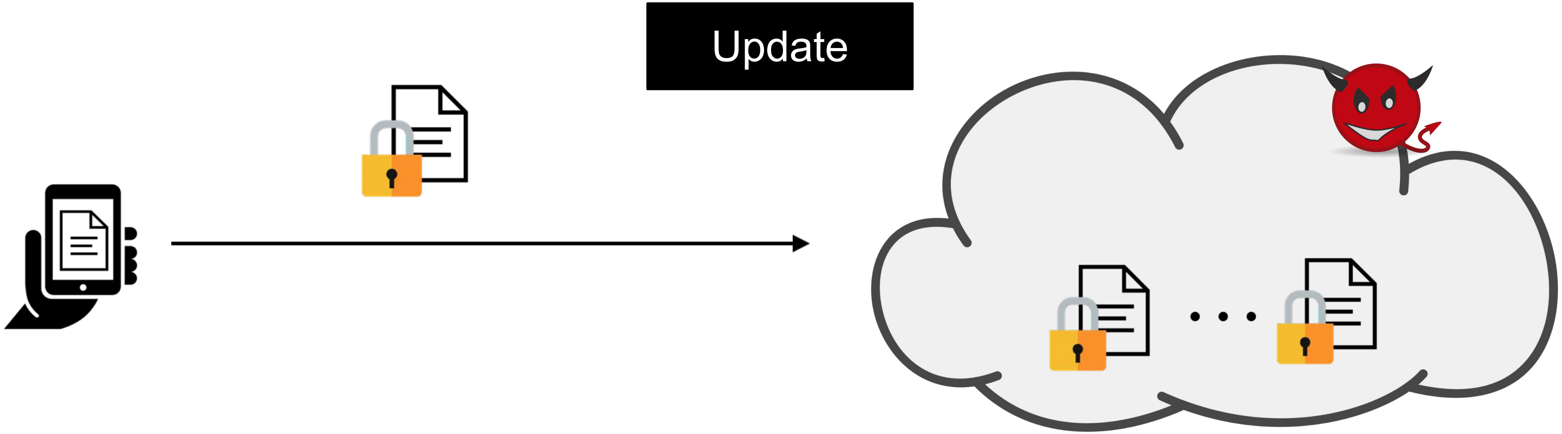
Searching Encrypted Data with Size-Locked Indexes

Min Xu, David Cash
University of Chicago

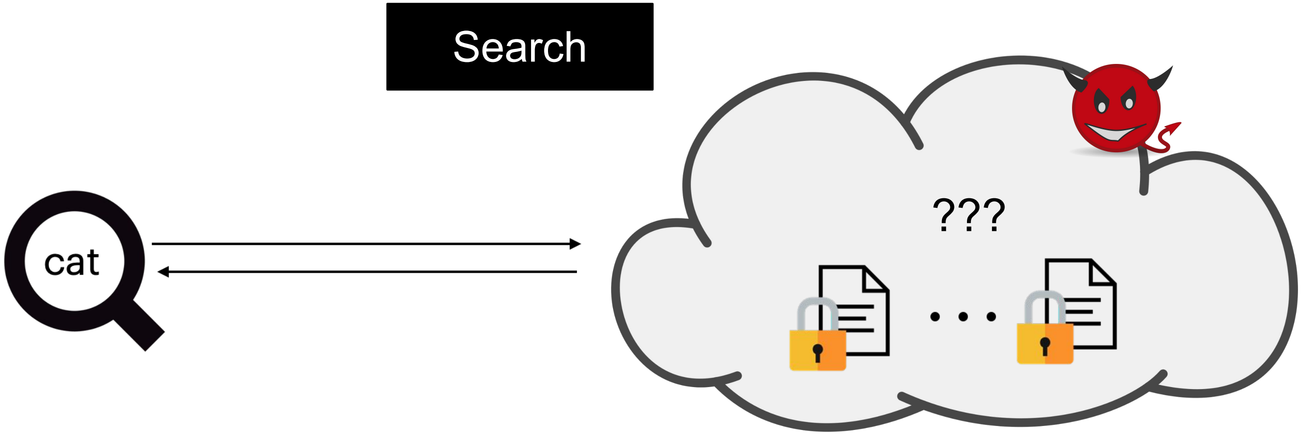
Armin Namavari, Thomas Ristenpart
Cornell (Tech)

E2E Encrypted Keyword Search



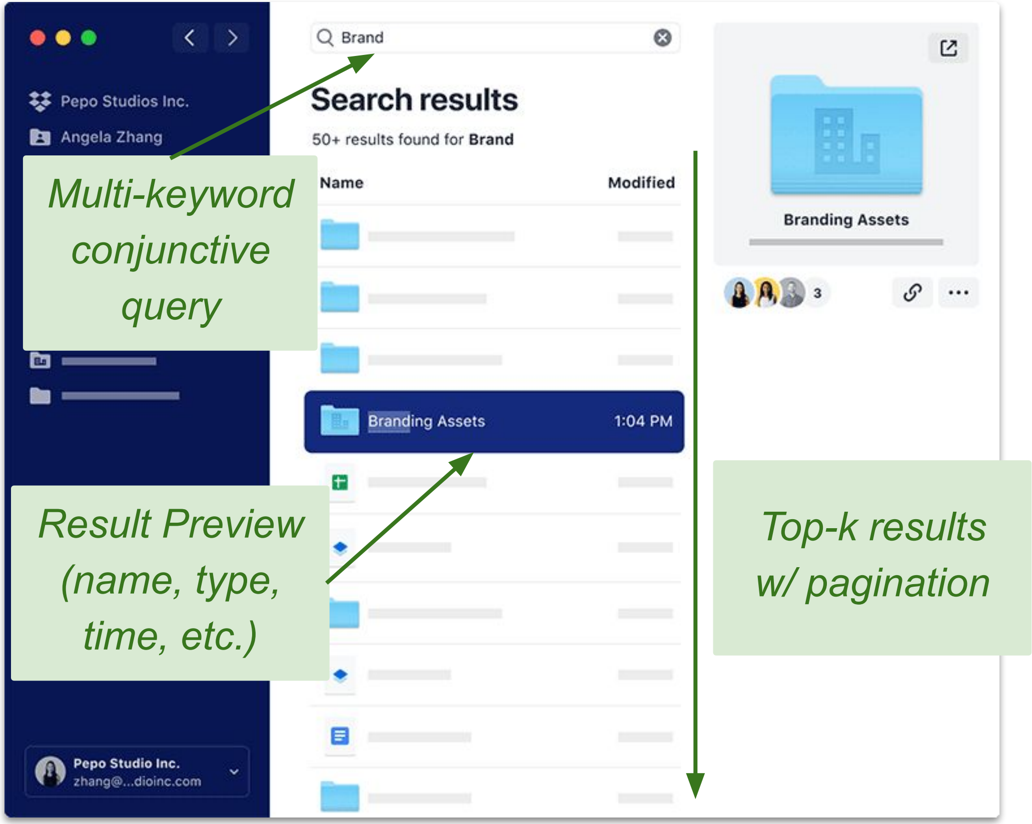
- Users outsource documents to the cloud for cost-effective storage and convenient access
  - Limited amount of client-side storage capacity, via web/mobile interface
  - End-to-end encryption for full privacy

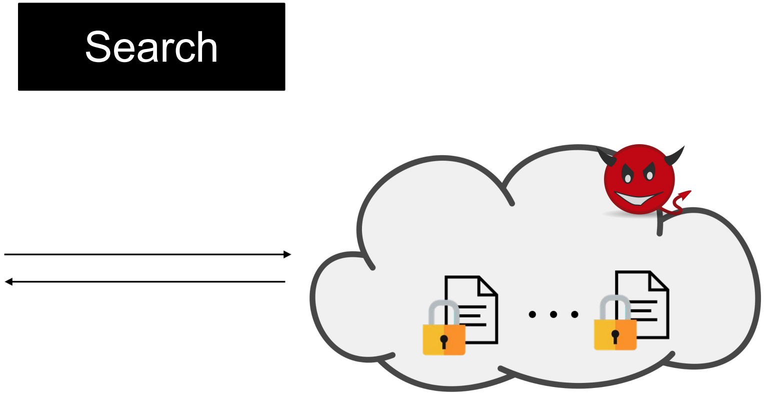
E2E Encrypted Keyword Search



- Users outsource documents to the cloud for cost-effective storage and convenient access
  - Limited amount of client-side storage capacity, via web/mobile interface
  - End-to-end encryption for full privacy
- End-to-end encryption makes key service utilities difficult, if not impossible
  - Keyword search — find the uploaded documents most relevant to the user’s keywords

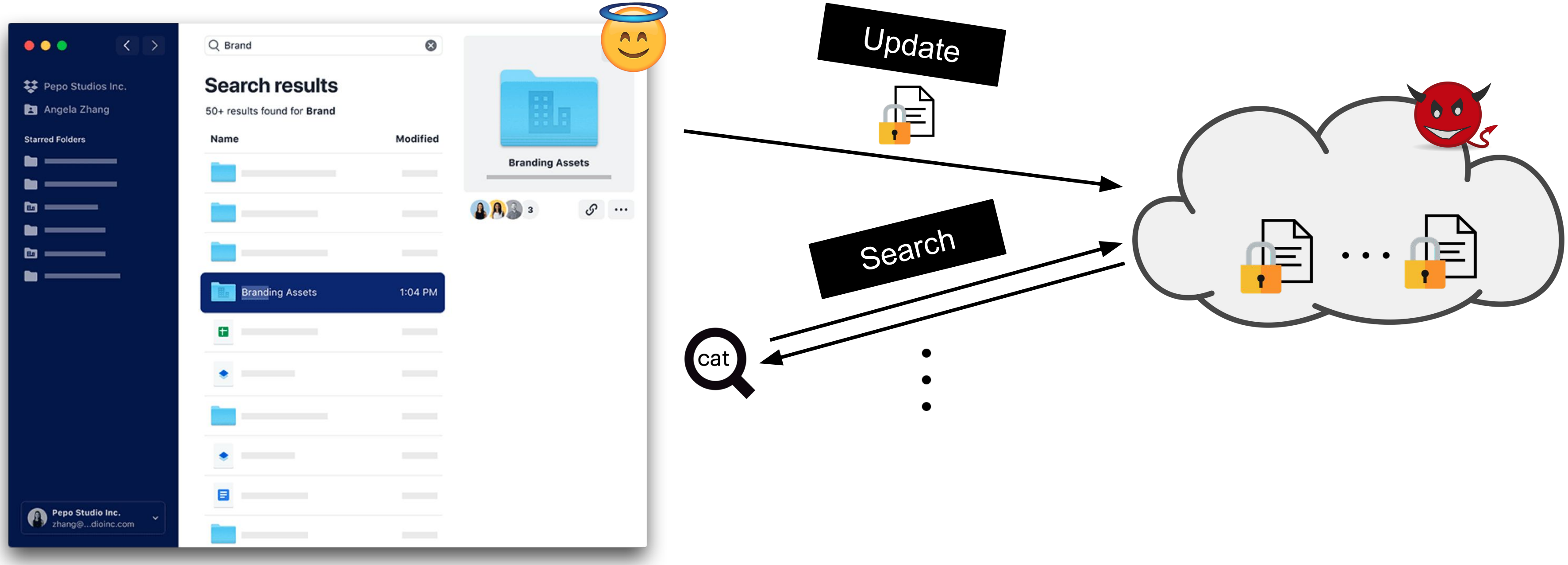
Our Target Search Interface



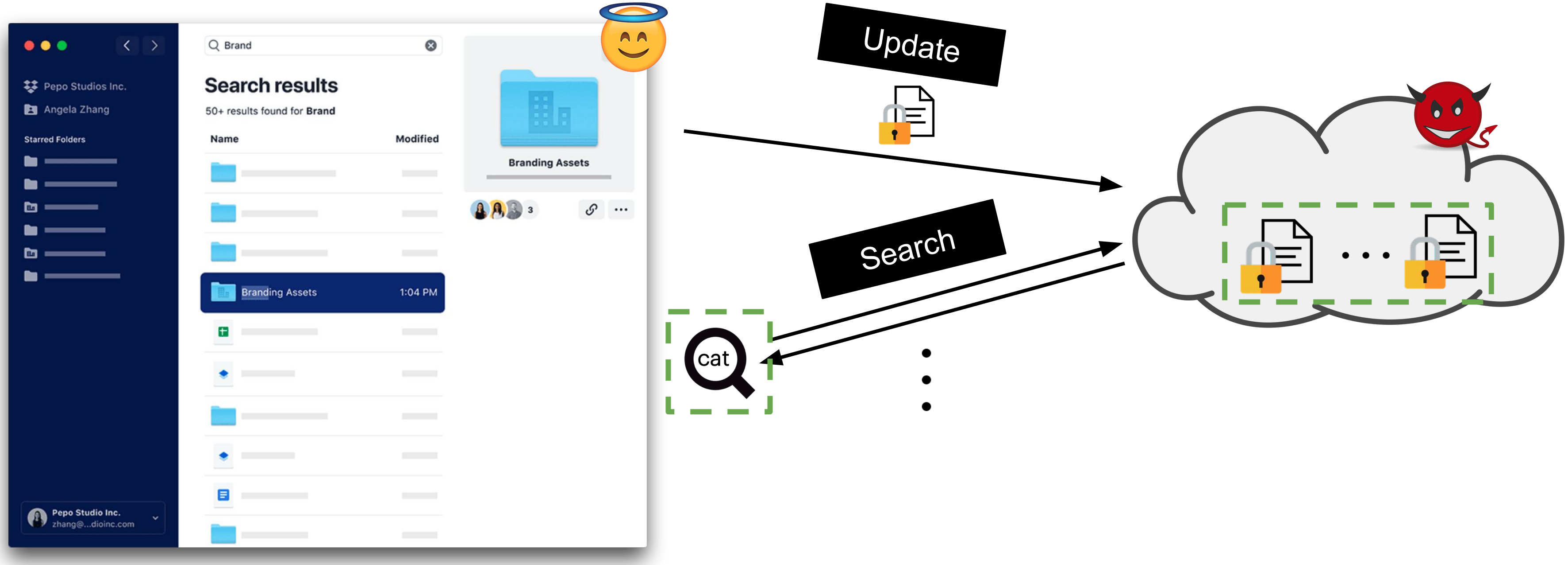


Similar interface is shared by Box, Google Drive, Microsoft OneDrive, etc.

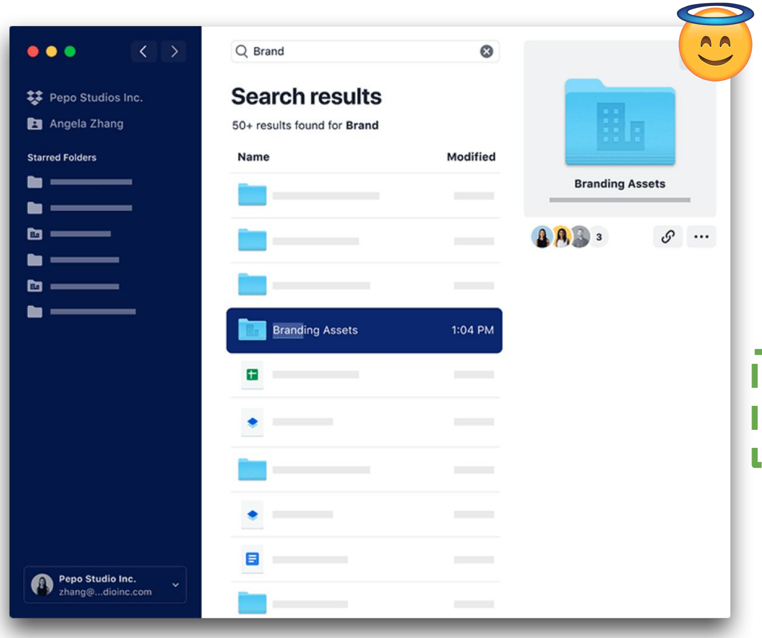
Our Threat Model

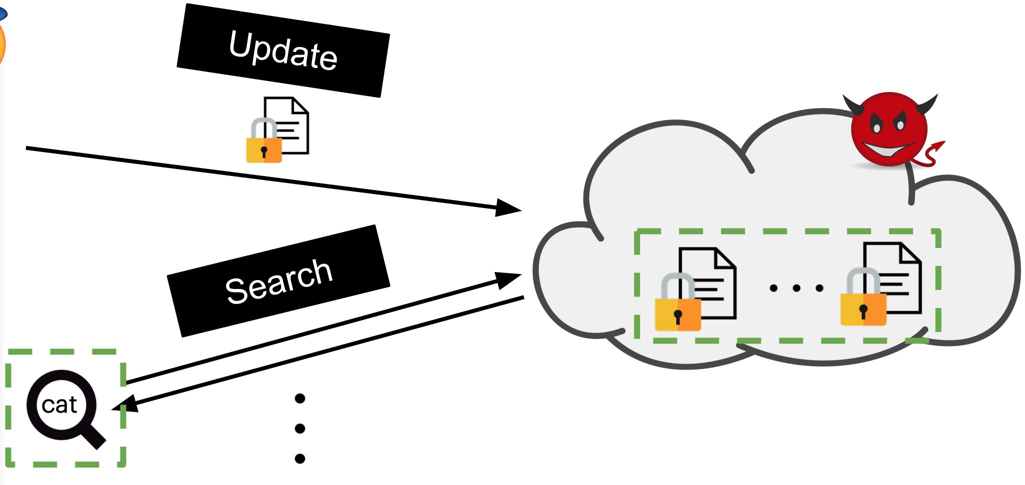


Our Threat Model



Our Threat Model

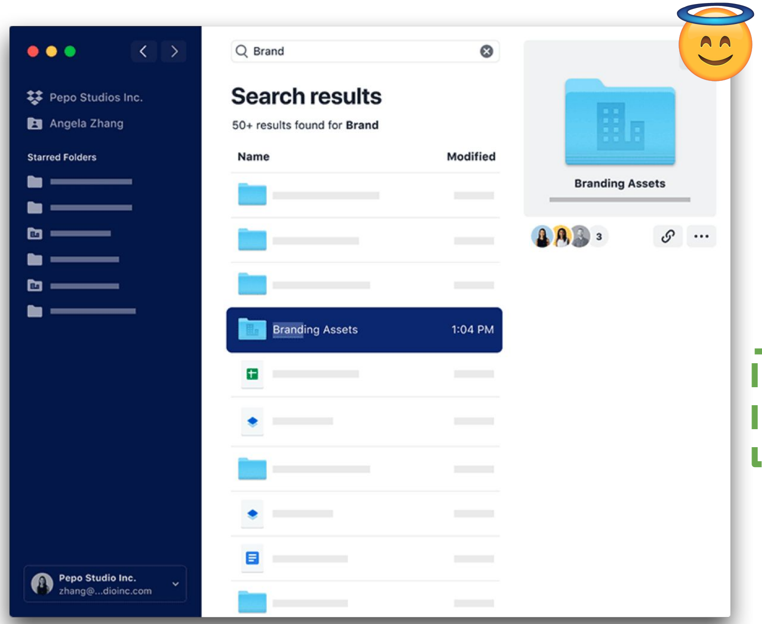


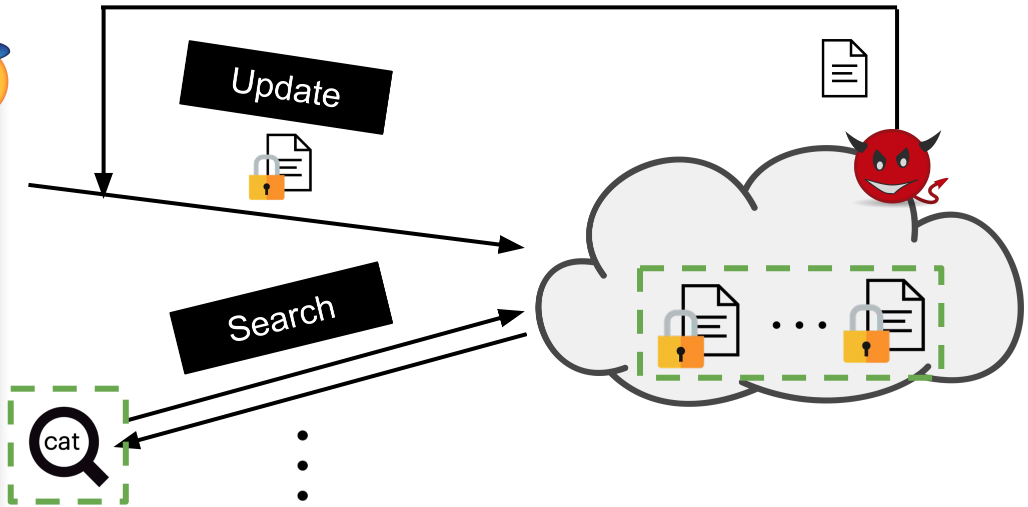


Persistent Active Adversary

- Full transcript of the update and search history

Our Threat Model





Persistent Active Adversary

- Full transcript of the update and search history
- *File-injection*: Adaptively inject documents of chosen words under user's key [CGPR15], [ZKP16]

Dynamic Searchable Symmetric Encryption (DSSE)

Time

Dynamic
[KPR'12]

- K, st; EDB $\leftarrow$ `setup`(λ)
- (K, st'; EDB') $\leftarrow$ `update`(K, st, op, (id, w); EDB)
- (DB(q);) $\leftarrow$ `search`(K, st, q; EDB)

Dynamic Searchable Symmetric Encryption (DSSE)

Leakage Profile $\Leftarrow$ Provably no more leaked

- **Result Pattern** the set of updates on the same keyword as the query
- **Volume** the number of documents containing the query keyword
- **Query Pattern** the set of queries on the same keyword

Time

Dynamic
[KPR'12]

- K, st; EDB $\leftarrow$ `setup`(λ)
- (K, st'; EDB') $\leftarrow$ `update`(K, st, op, (id, w); EDB)
- (DB(q);) $\leftarrow$ `search`(K, st, q; EDB)

Dynamic Searchable Symmetric Encryption (DSSE)

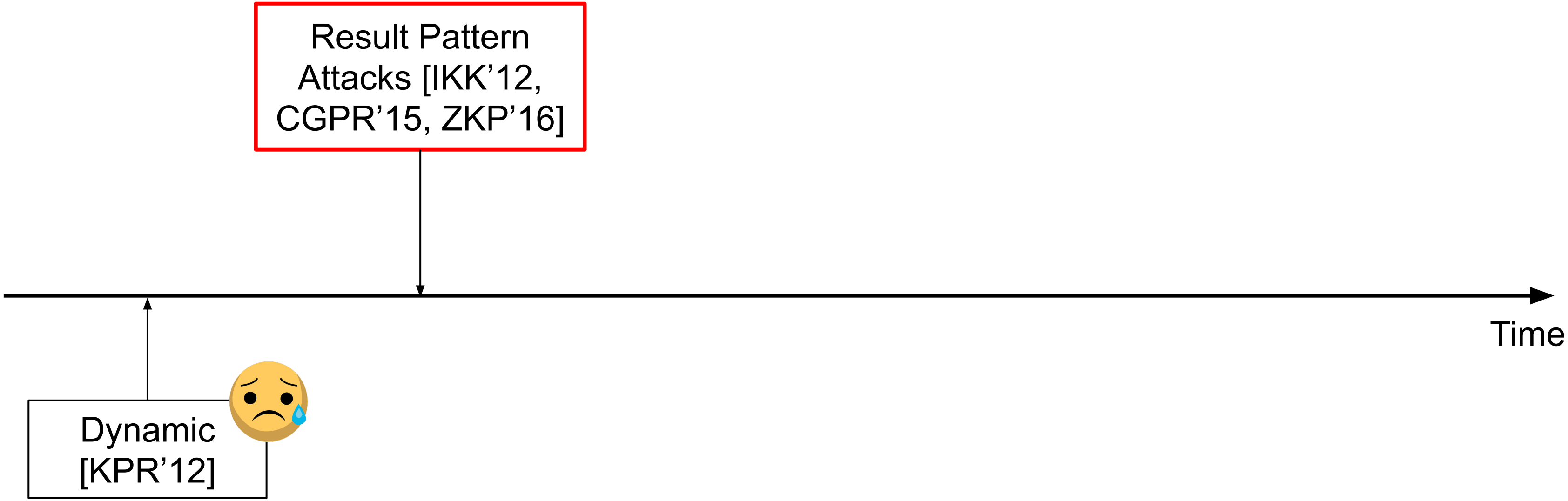
Time

Dynamic
[KPR'12]

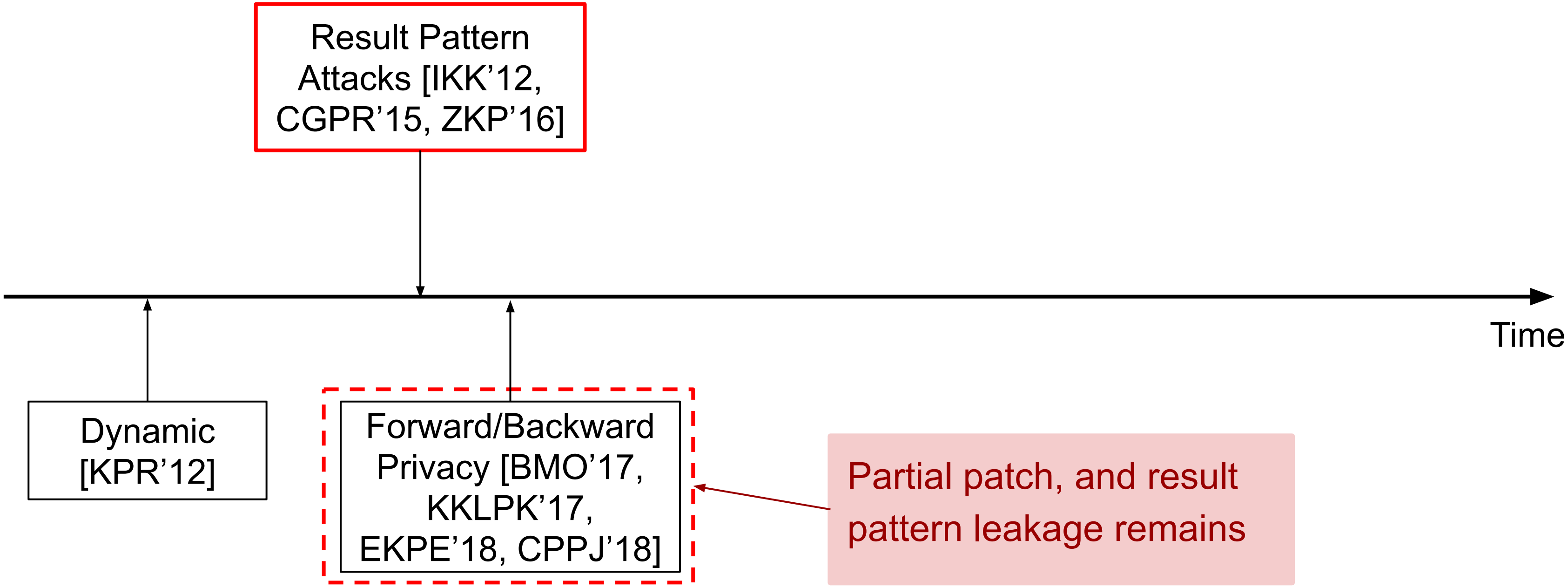
- K, st; EDB ← `setup`(λ)
- (K, st'; EDB') ← `update`(K, st, op, (id, w); EDB)
- (DB(q);) ← `search`(K, st, q; EDB)

- Leakage profile are vulnerable to various *leakage-abuse attacks* (LAAs)
- Limited search functionality, i.e., no ranking or previewing

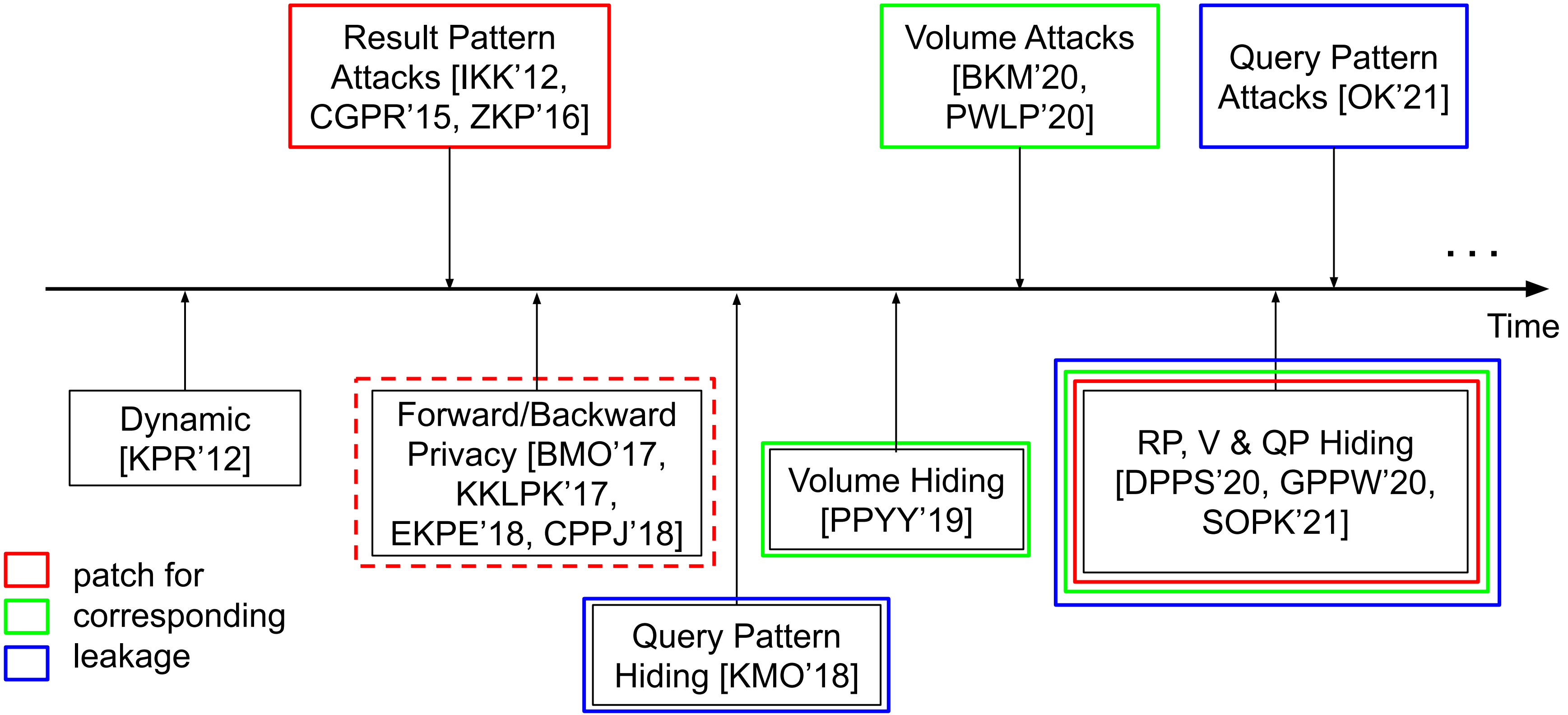
Dynamic Searchable Symmetric Encryption (DSSE)



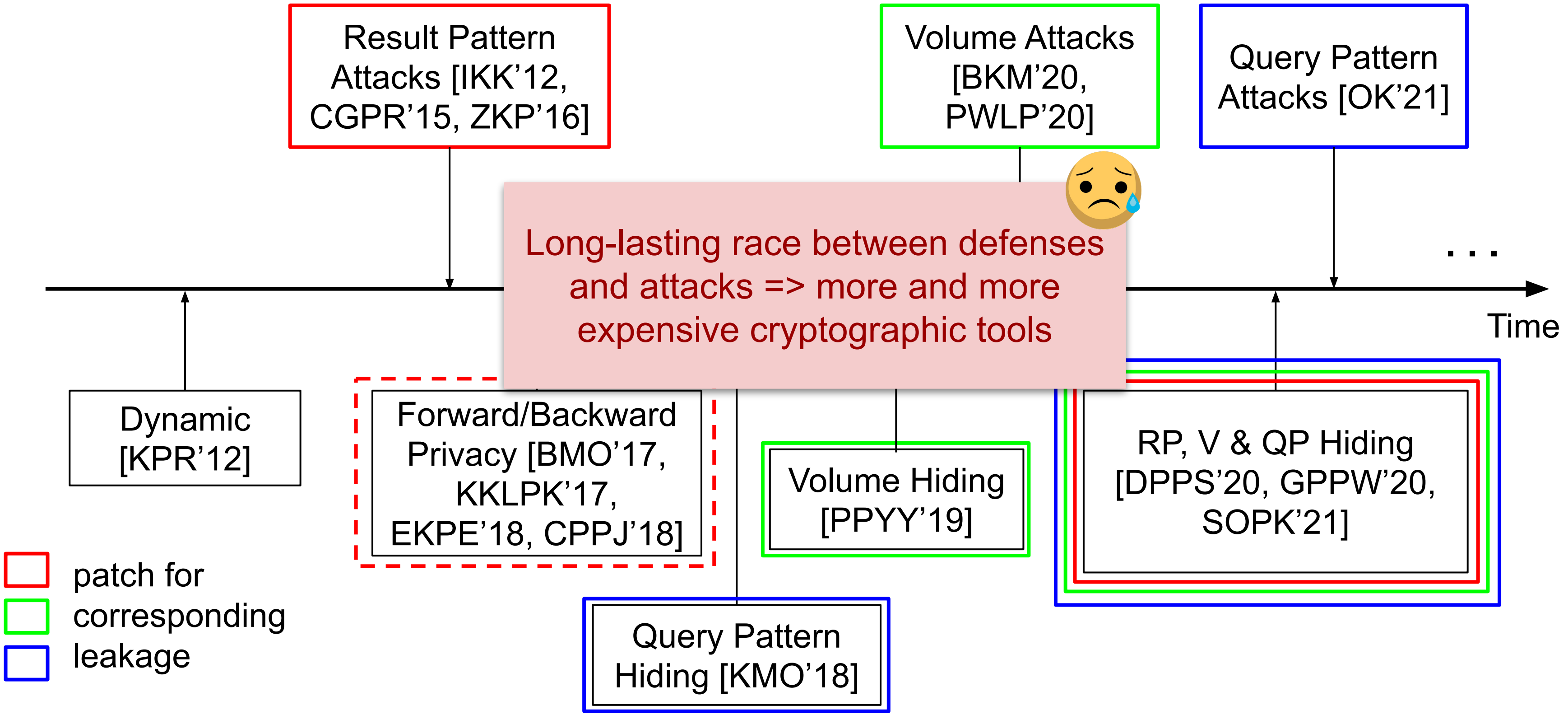
Dynamic Searchable Symmetric Encryption (DSSE)



Dynamic Searchable Symmetric Encryption (DSSE)



Dynamic Searchable Symmetric Encryption (DSSE)

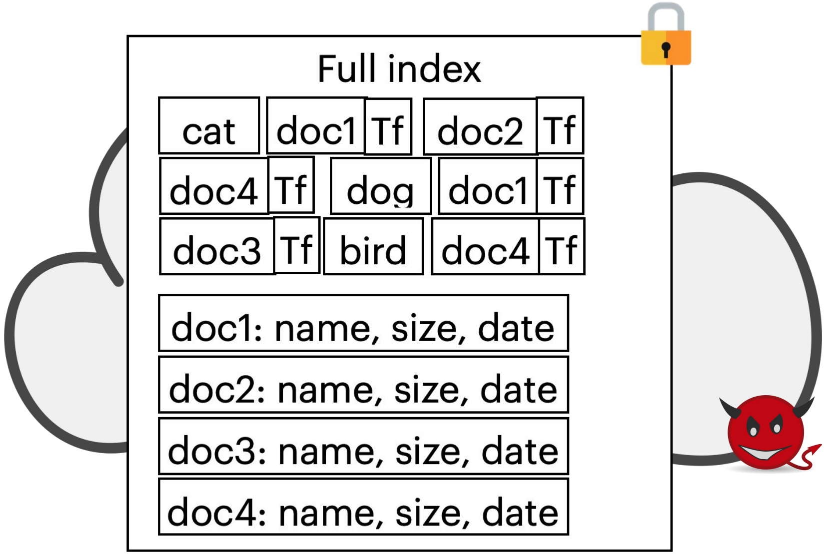


Our Approach - Download-then-Search-Locally

A Different Idea against File-Injection Attacks

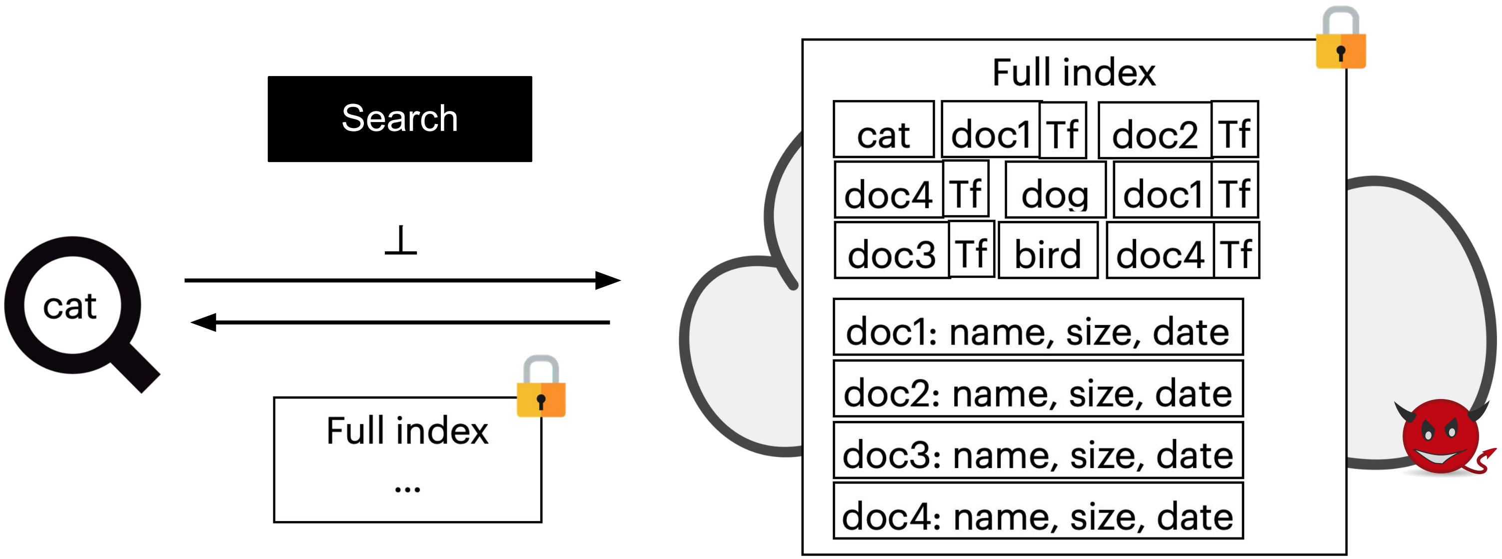
Our Approach - Download-then-Search-Locally

A Different Idea against File-Injection Attacks



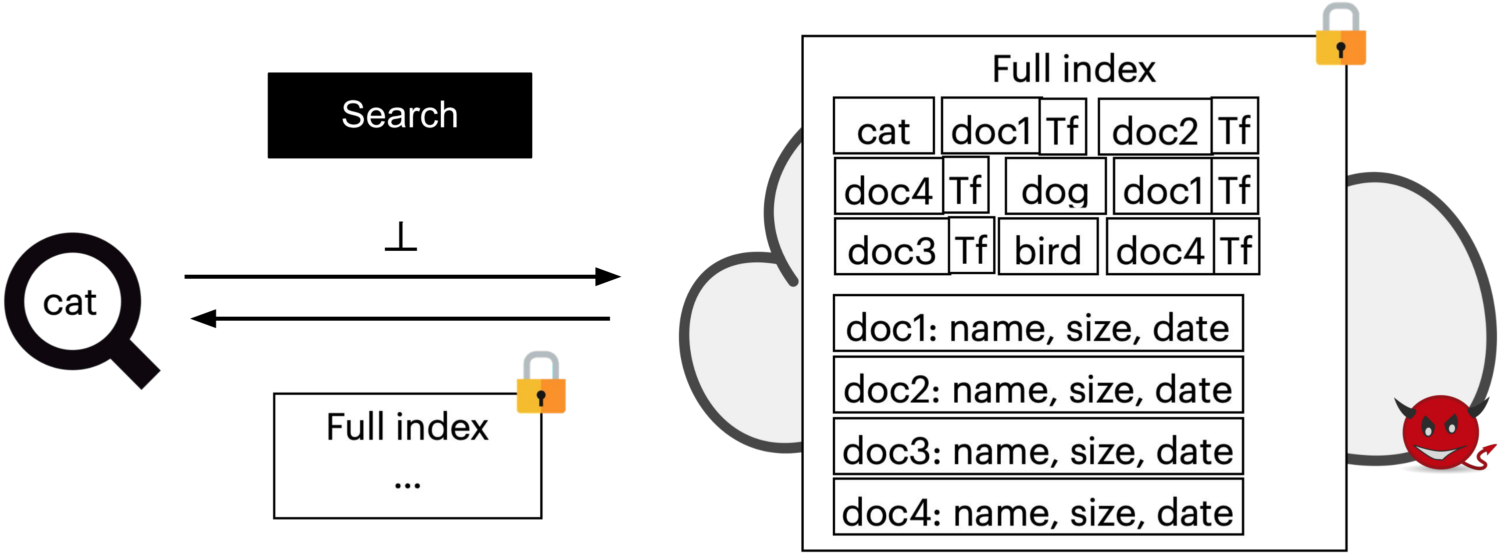
Our Approach - Download-then-Search-Locally

A Different Idea against File-Injection Attacks



Our Approach - Download-then-Search-Locally

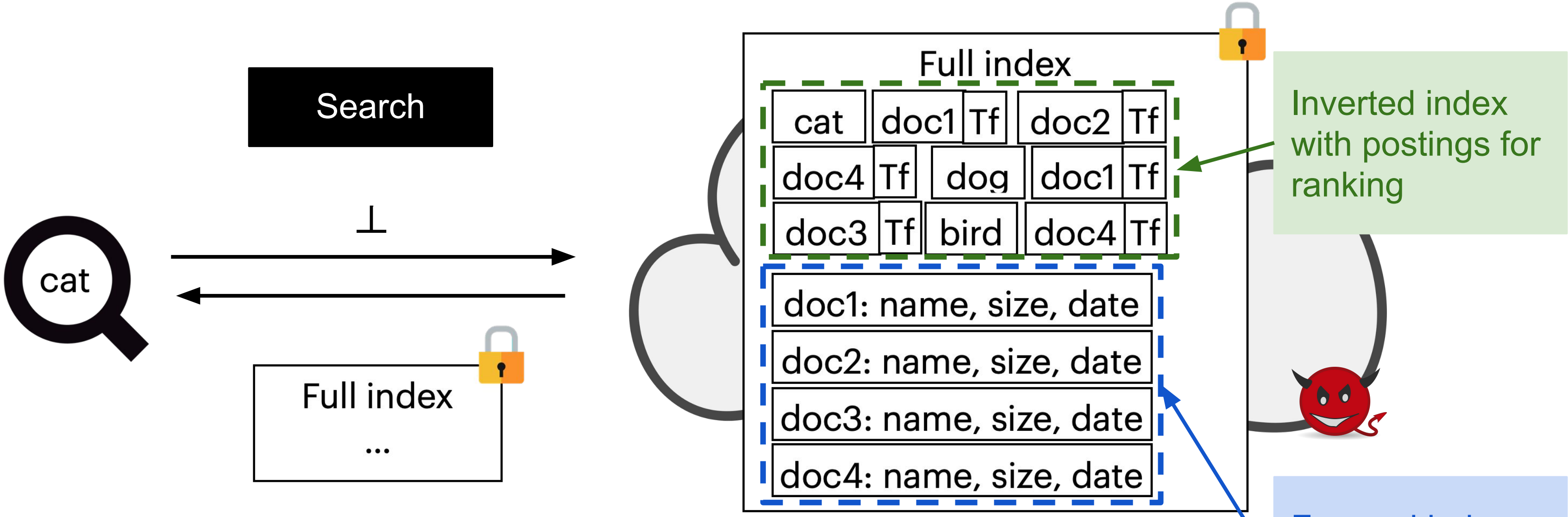
A Different Idea against File-Injection Attacks



- No result pattern, volume or query pattern leakage whatsoever!

Our Approach - Download-then-Search-Locally

A Different Idea against File-Injection Attacks

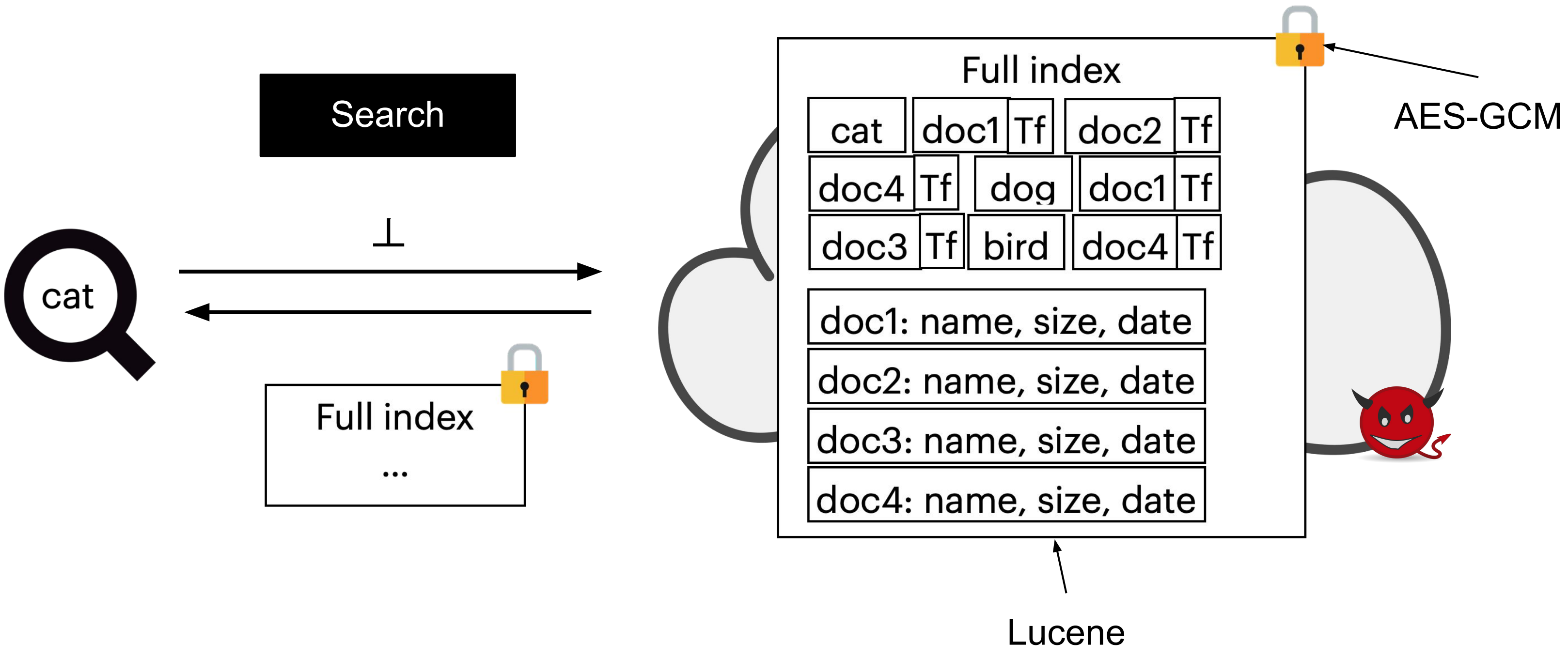


- ❏ No result pattern, volume or query pattern leakage whatsoever!
- ❏ Straightforward support for ranking and preview!

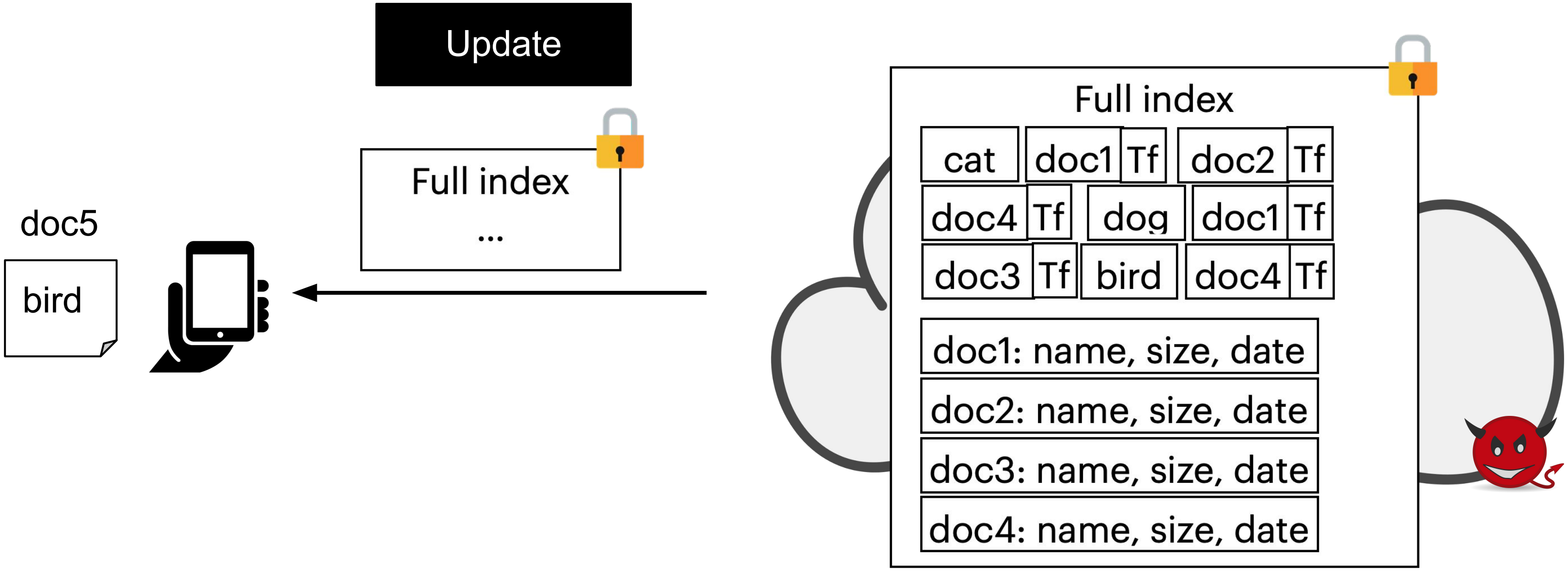
Our Contributions

- Develop previously under-treated technique: download-then-search-locally
- Identify attacks against naive construction and give solutions with security proofs
- New constructions for feature-rich, scalable search on E2E encrypted data
- Real-world prototype-based evaluation

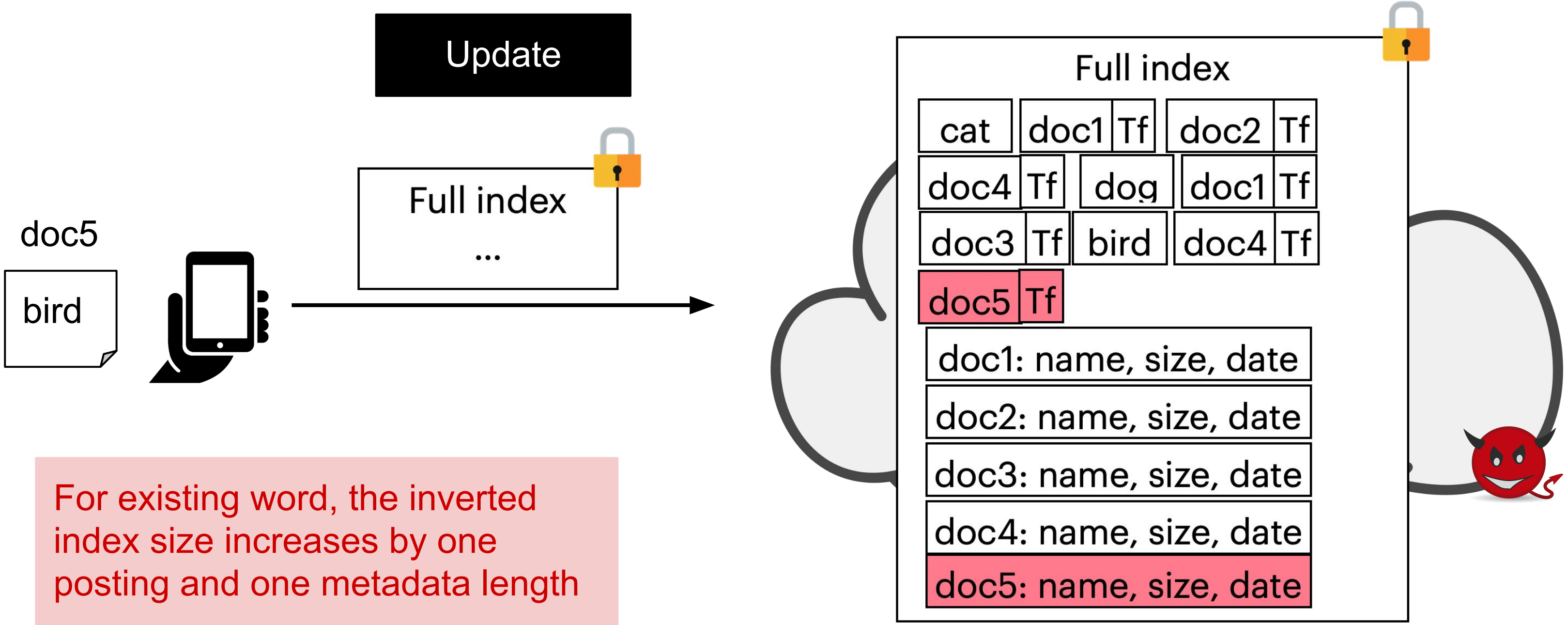
Naïve Download-then-Search-Locally



Naïve Download-then-Search-Locally

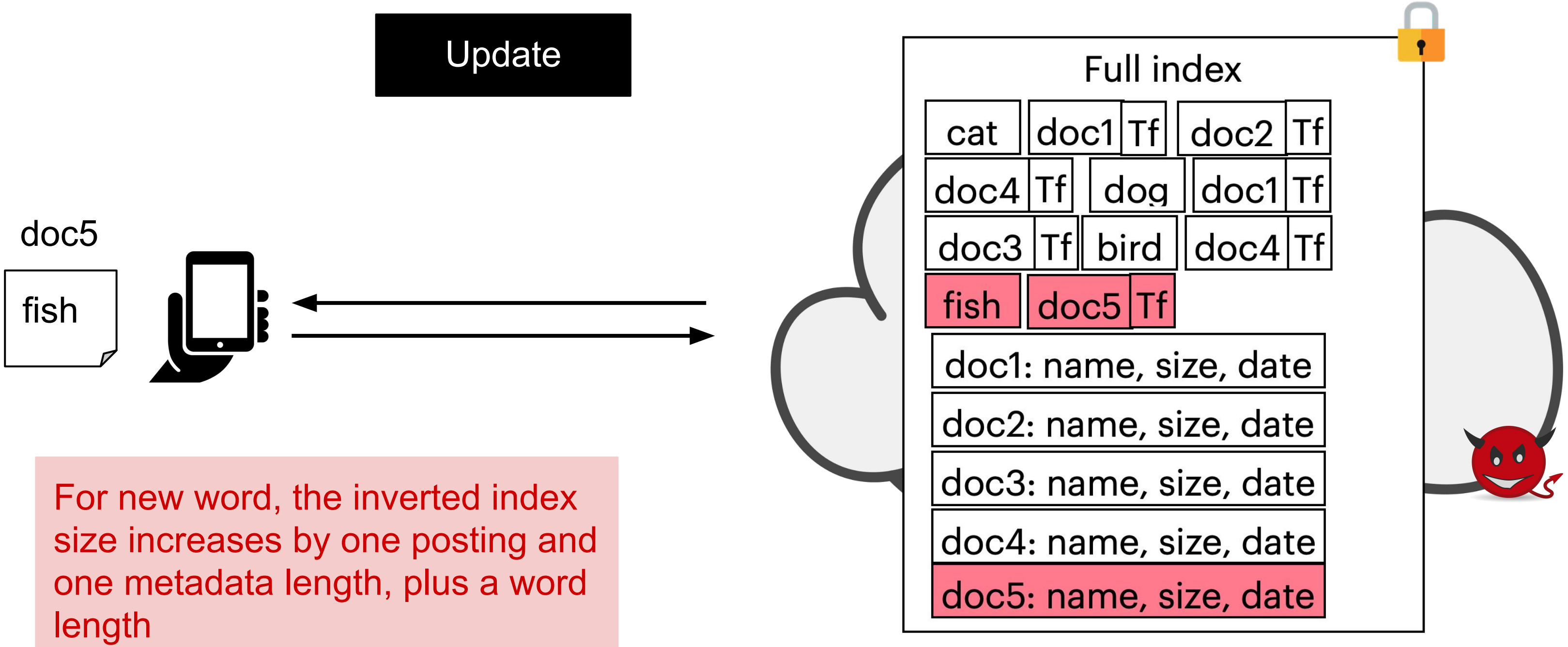


Naïve Download-then-Search-Locally



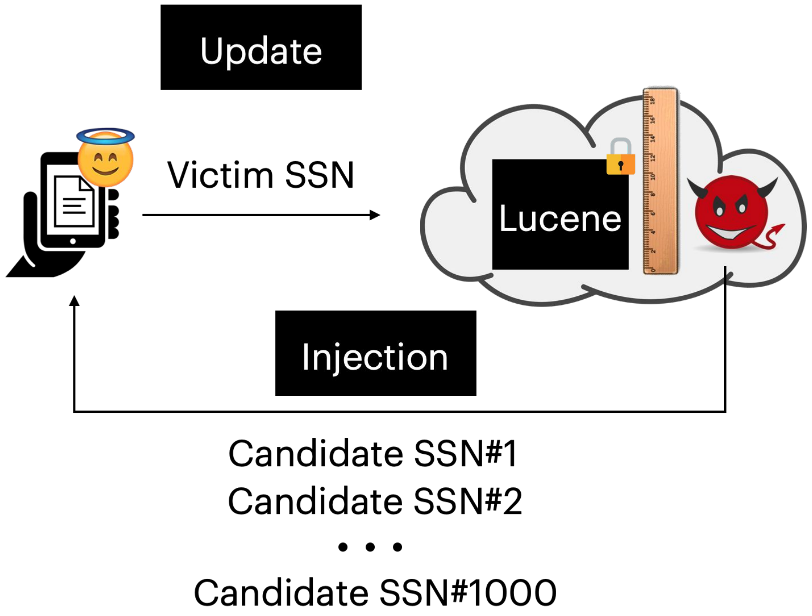
For existing word, the inverted index size increases by one posting and one metadata length

Naïve Download-then-Search-Locally

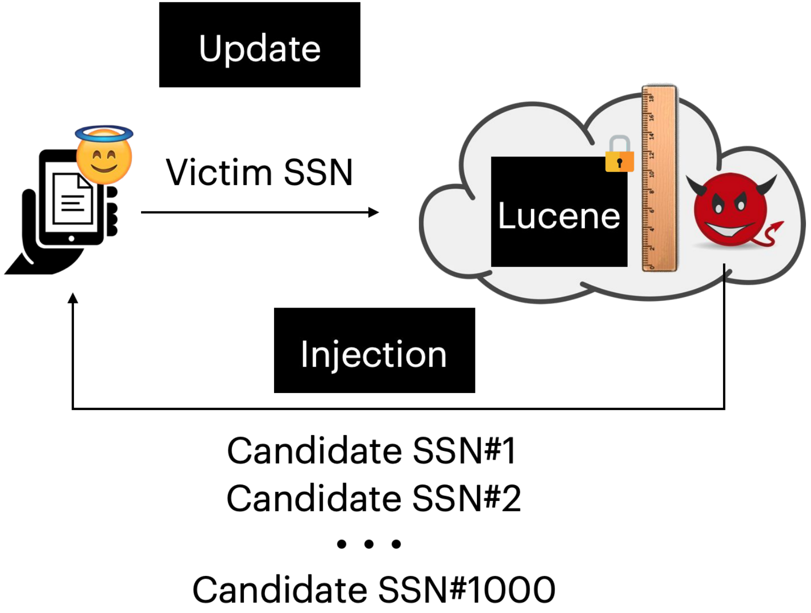


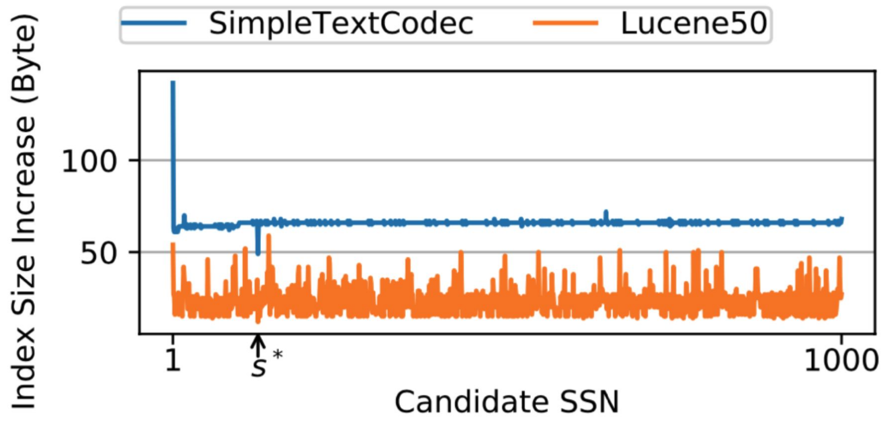
For new word, the inverted index size increases by one posting and one metadata length, plus a word length

LAAs w/ File-Injection

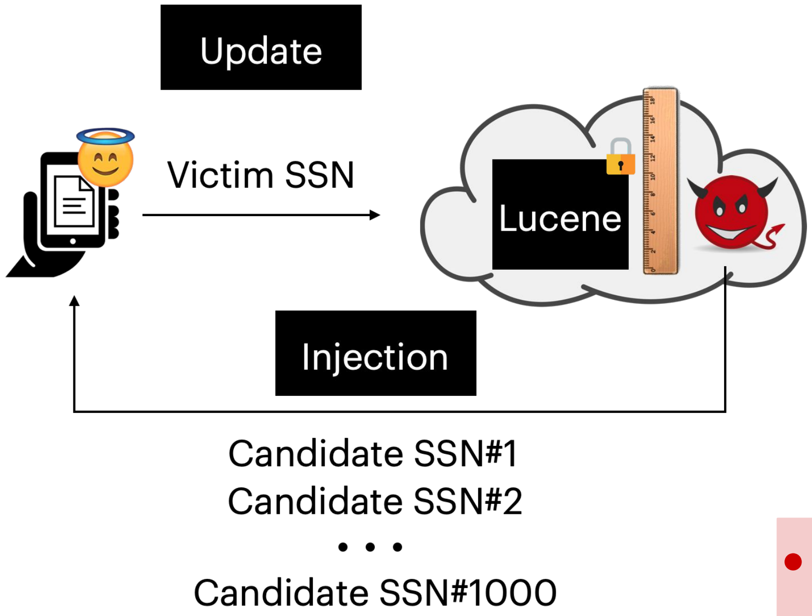


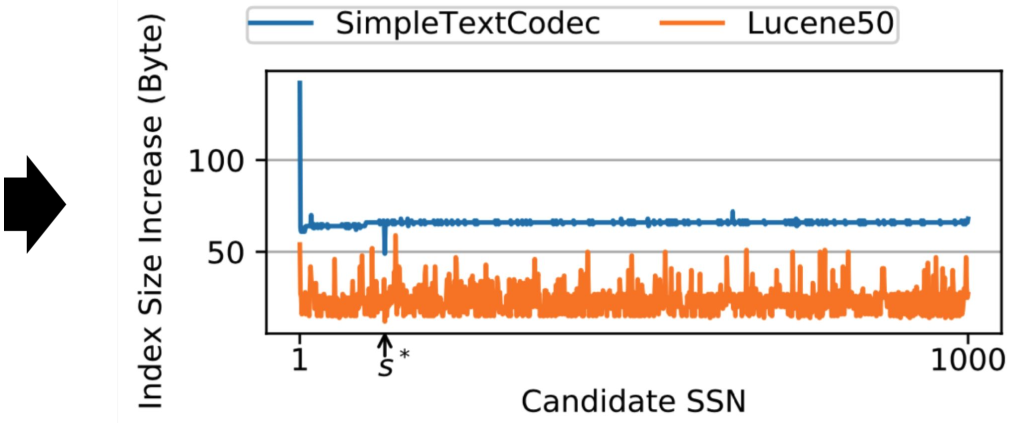
LAAs w/ File-Injection





LAAs w/ File-Injection





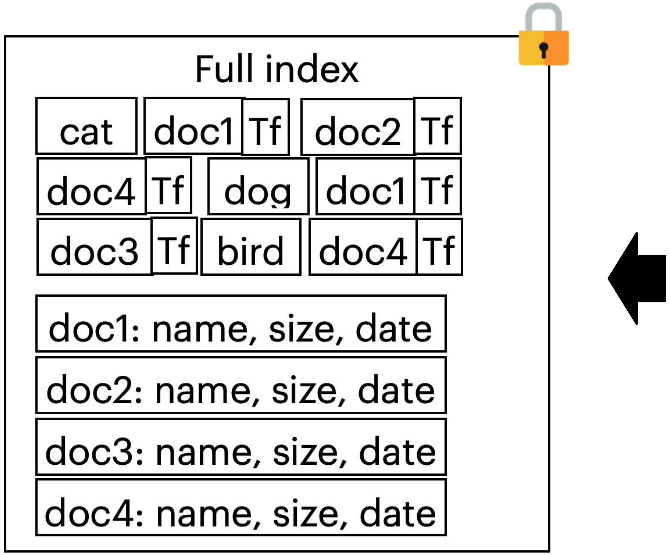
- Size of standard search index, encrypted using standard encryption, leaks sensitive information!
- File-injection is powerful to recover the data from the leakage!

Size-locked Indexing

- Size-locking - make the length of the index encoding a function of only the information we are willing to leak
  - ✓ N, the total number of postings
  - ✓ |D|, the total number of documents
  - X |W|, the total number of indexed keywords

Size-locked Indexing

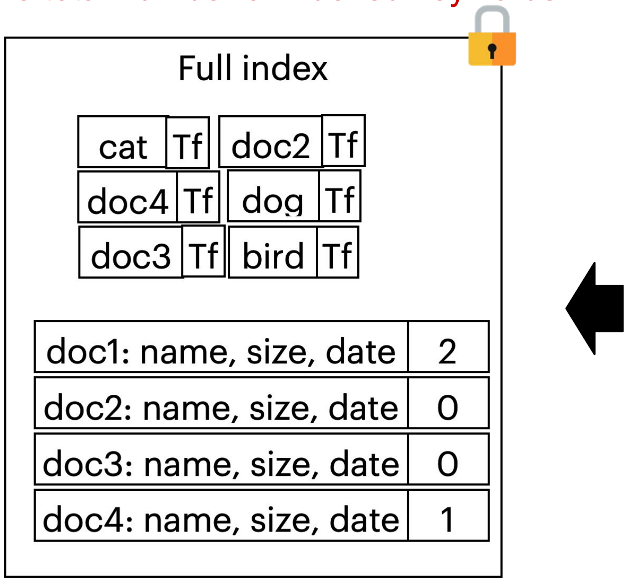
- Size-locking - make the length of the index encoding a function of only the information we are willing to leak
  - ✓ N, the total number of postings
  - ✓ |D|, the total number of documents
  - X |W|, the total number of indexed keywords



Lucene + AES-GCM has index size: **O(N+|W|+|D|)** ⇒ NOT size-locking

Size-locked Indexing

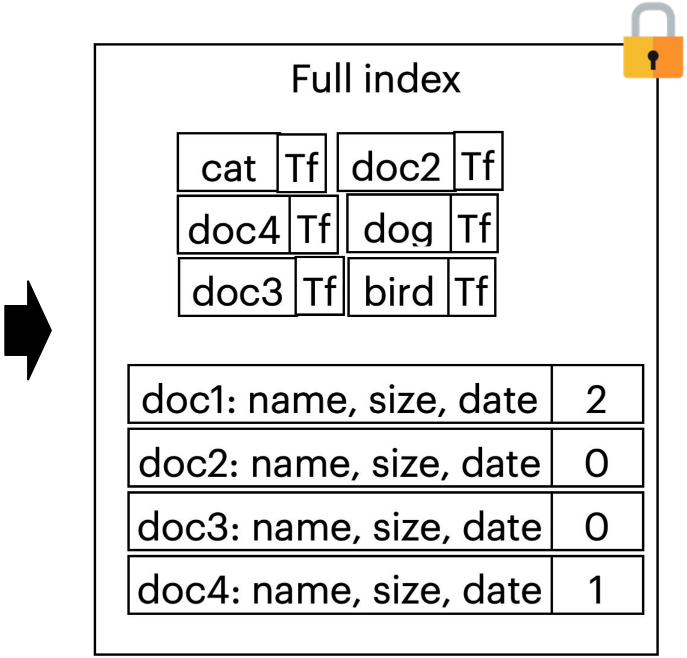
- Size-locking - make the length of the index encoding a function of only the information we are willing to leak
  - ✓ N, the total number of postings
  - ✓ |D|, the total number of documents
  - X |W|, the total number of indexed keywords



Our size-locked index:
O(N+|D|)

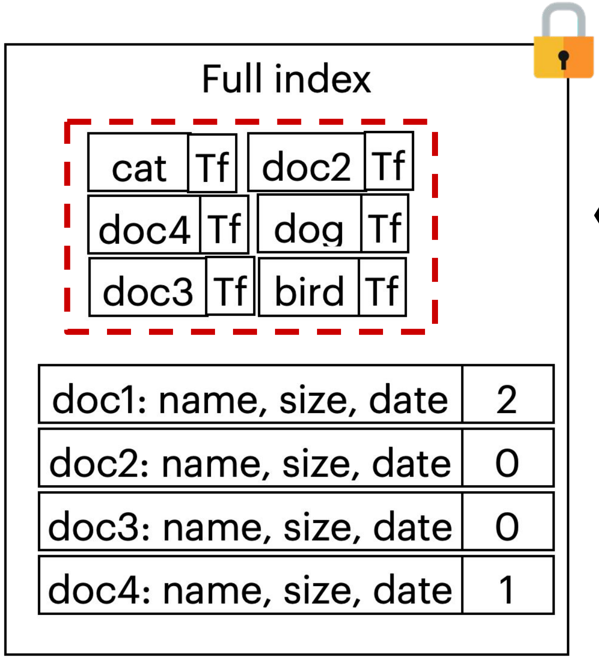
Scalability Challenge

For the entire Enron dataset, the full index gets as large as **228MB**! ⇒ Impractical to download for every search



Scalability Challenge

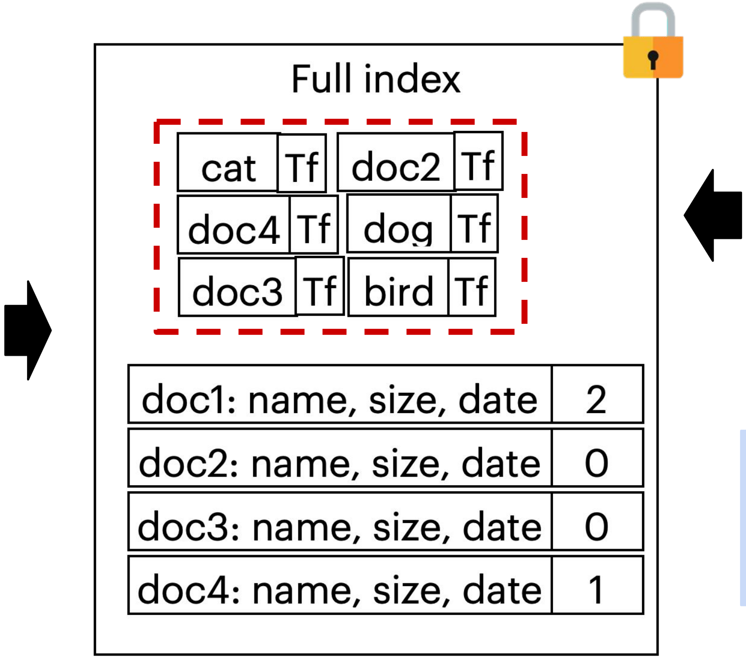
For the entire Enron dataset, the full index gets as large as **228MB**! ⇒ Impractical to download for every search



Out of the 228MB of index, **212MB** is the inverted index

Scalability Challenge

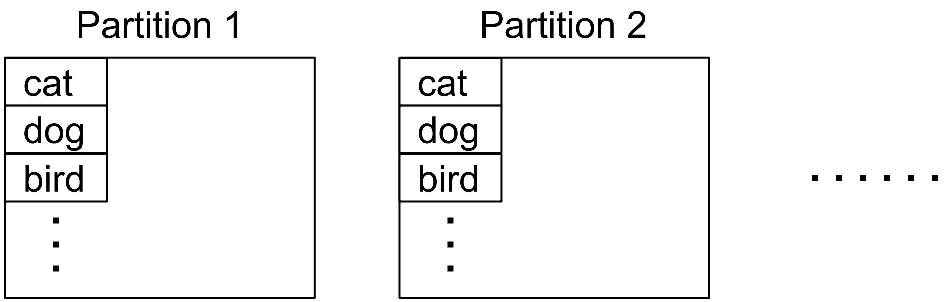
For the entire Enron dataset, the full index gets as large as **228MB**! ⇒ Impractical to download for every search



Out of the 228MB of index, **212MB** is the inverted index

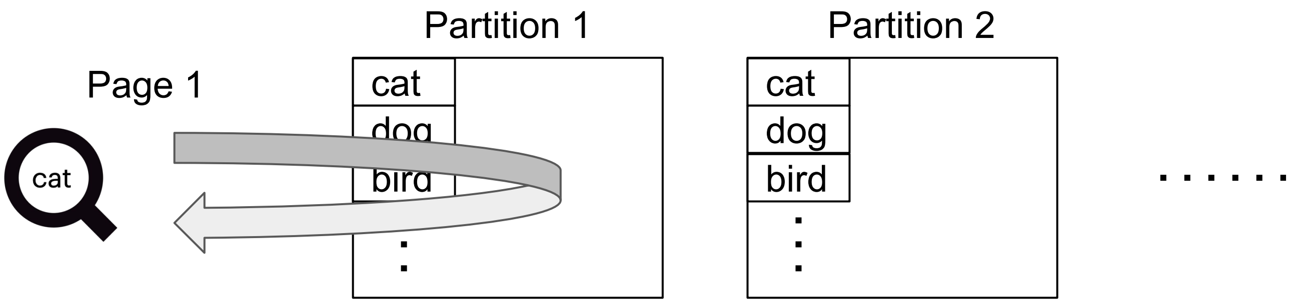
Can we reduce the cost down to the necessary, top-relevant postings?

Secure Vertical Index Partitioning



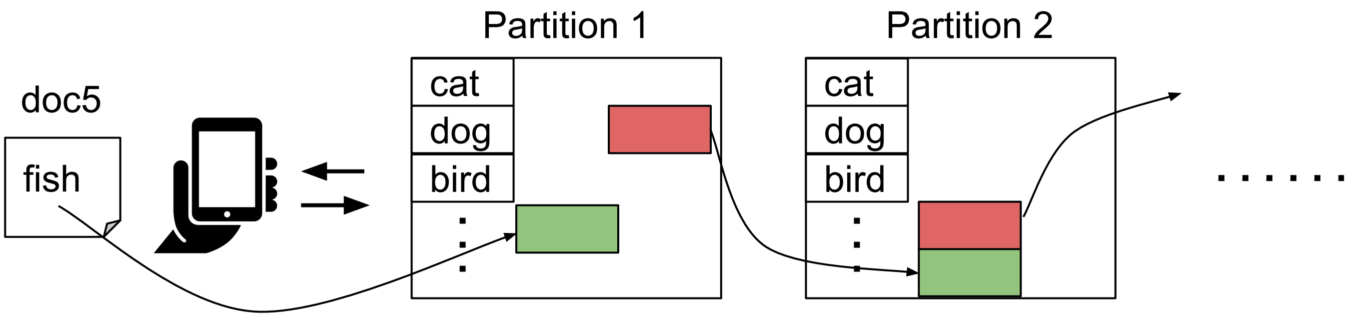
- Partition index into blobs by relevance to the indexed keywords
  - Top relevant results all in the first partition

Secure Vertical Index Partitioning



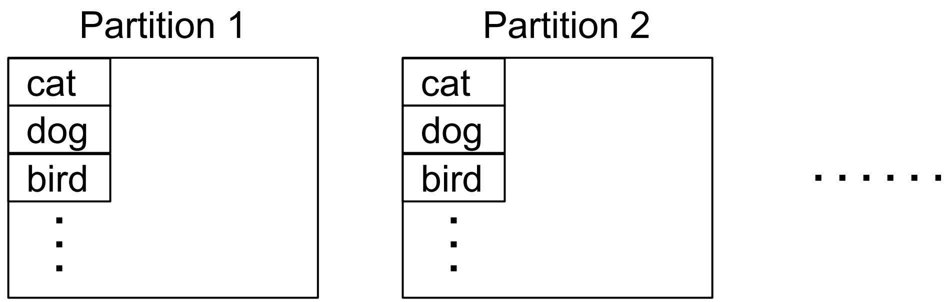
- Partition index into blobs by relevance to the indexed keywords
  - Top relevant results all in the first partition

Secure Vertical Index Partitioning



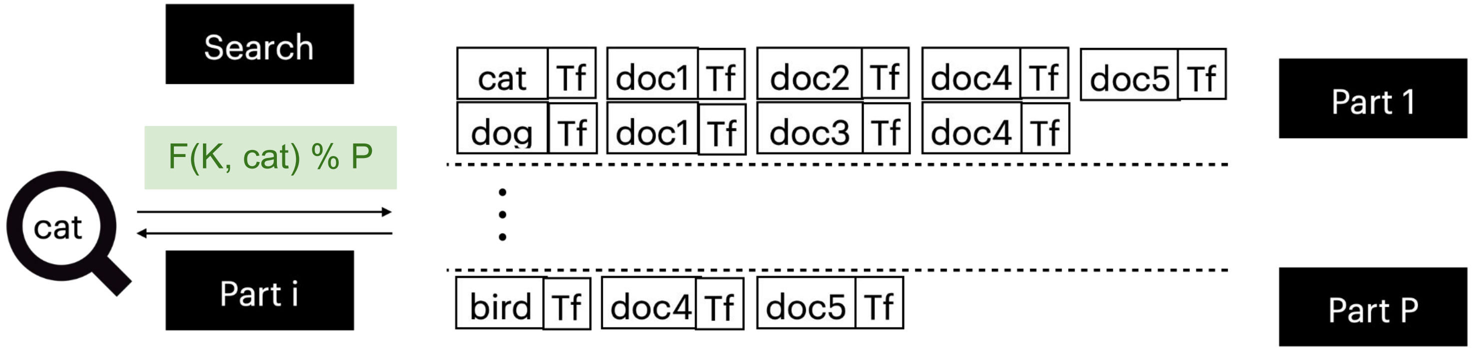
- Partition index into blobs by relevance to the indexed keywords
  - Top relevant results all in the first partition
- New postings from updates are merged, and less relevant ones are kicked to the subsequent partitions
  - When and how many postings to kick depend solely on the total number of postings, i.e., N

Secure Vertical Index Partitioning



- ❑ Partition index into blobs by relevance to the indexed keywords
  - ❑ Top relevant results all in the first partition
- ❑ New postings from updates are merged, and less relevant ones are kicked to the subsequent partitions
  - ❑ When and how many postings to kick depend solely on the total number of postings, i.e., N
- ❑ No result pattern, volume or query pattern leakage

Secure Horizontal Index Partitioning



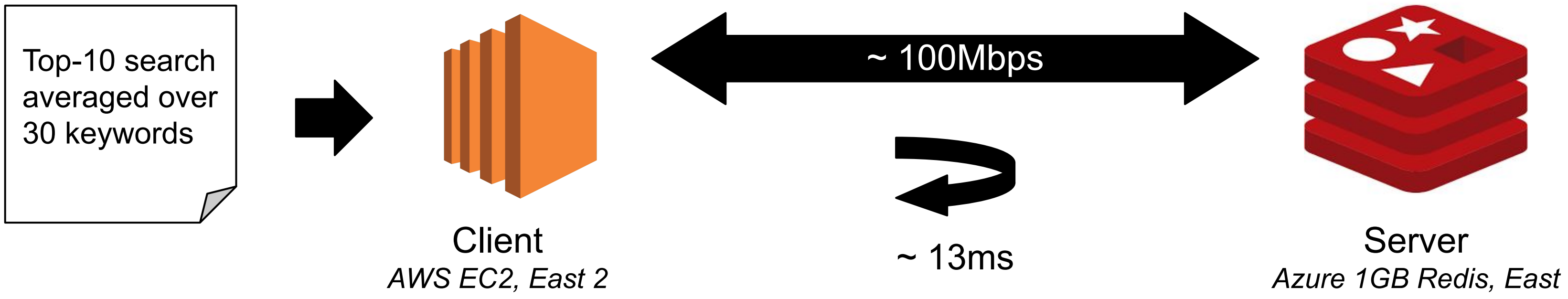
- ❏ Only 1 out of P partitions is needed for a single keyword query
- ❏ **Security Intuition** Randomly group words into buckets
  - ❏ Update leaks the # words of the update in each bucket
  - ❏ Search leaks the partition access pattern -> words in the same partition remain indistinguishable
- ❏ Can be combined with the vertical index partitioning for more efficiency

Performance Evaluations

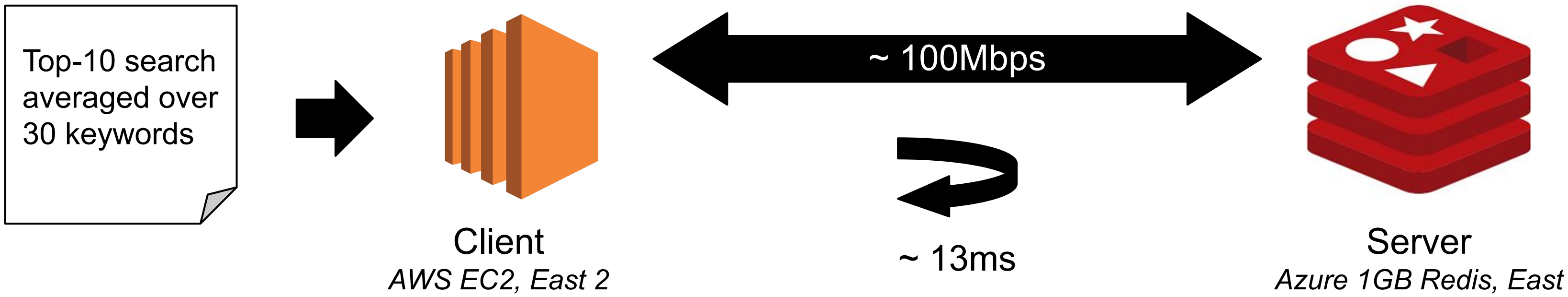
Dataset	Data Size	# Docs	# Keywords	# Postings
10% Enron	0.2 GB	51,731	131,903	4.3×10^6
50% Enron	0.8 GB	258,655	280,474	21.3×10^6
100% Enron	1.7 GB	517,310	338,913	42.5×10^6

- FULL: the basic size-locked download-then-search-locally
- VPart: the vertically partitioned size-locked download-then-search-locally
- VHPart-P: the vertically-and-horizontally partitioned size-locked download-then-search-locally with P horizontal partitions
- CTR-DSSE: the efficient forward private DSSE, named Diana from [BMO'17]

Search Performance

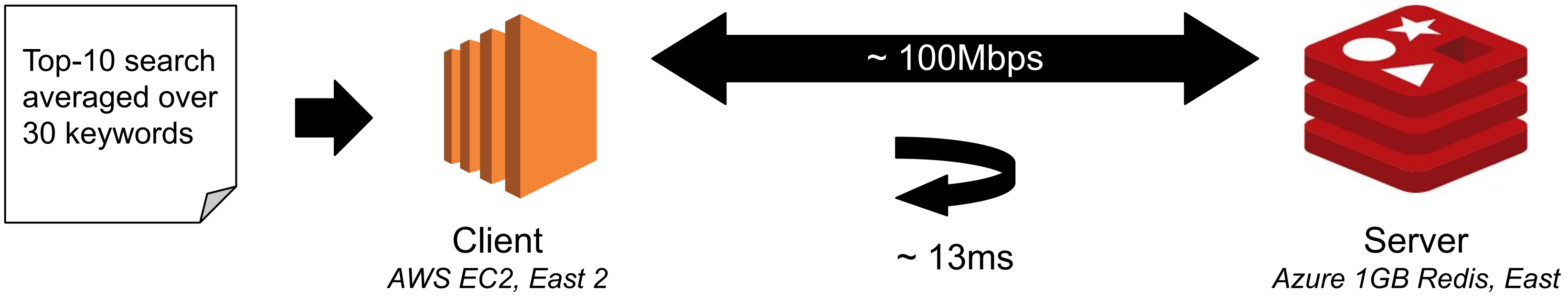


Search Performance - Bandwidth Cost



	Bandwidth (MB)			
	10% Enron	50% Enron	100% Enron	
Full	25.09	116.30	228.15	
VPart	6.72	16.68	25.38	9x
VHPart-10	1.17	4.16	7.51	30x

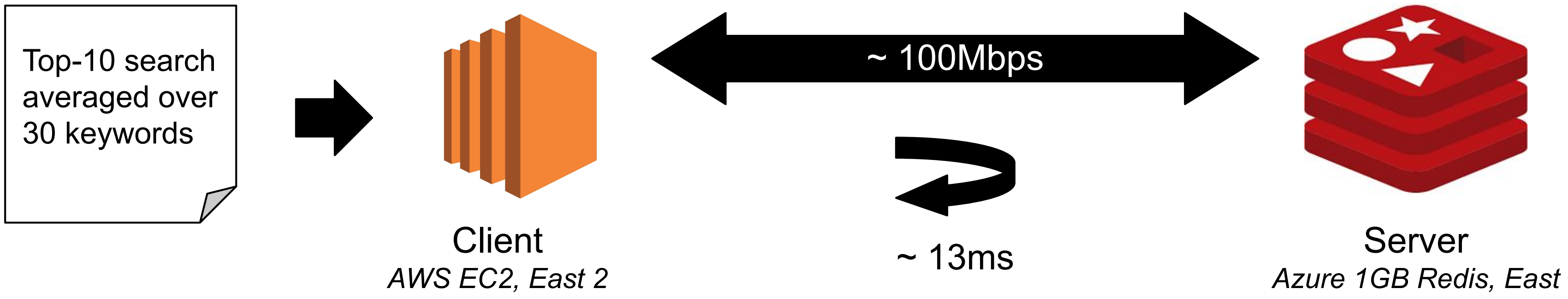
Search Performance - Bandwidth Cost



	Bandwidth (MB)		
	10% Enron	50% Enron	100% Enron
Full	25.09	116.30	228.15
VPart	6.72	16.68	25.38
VHPart-10	1.17	4.16	7.51
CTR-DSSE	1.48	4.35	--

- Cold start ⇒ O(|W|) counters downloaded for every search
- Download all matches

Search Performance - Latency



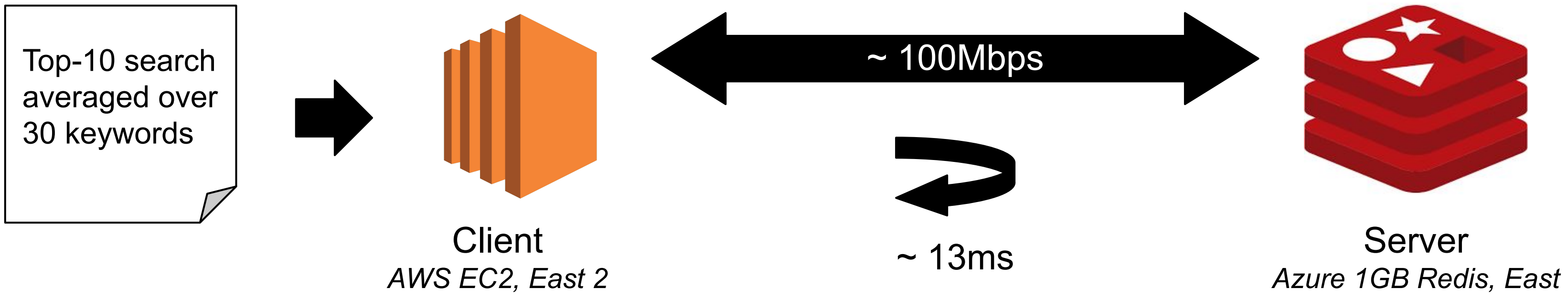
Latency (sec)

	10% Enron	50% Enron	100% Enron
Full	1.61	6.80	12.80
VPart	0.78	1.69	2.47
VHPart-10	0.16	0.33	0.46

5x

28x

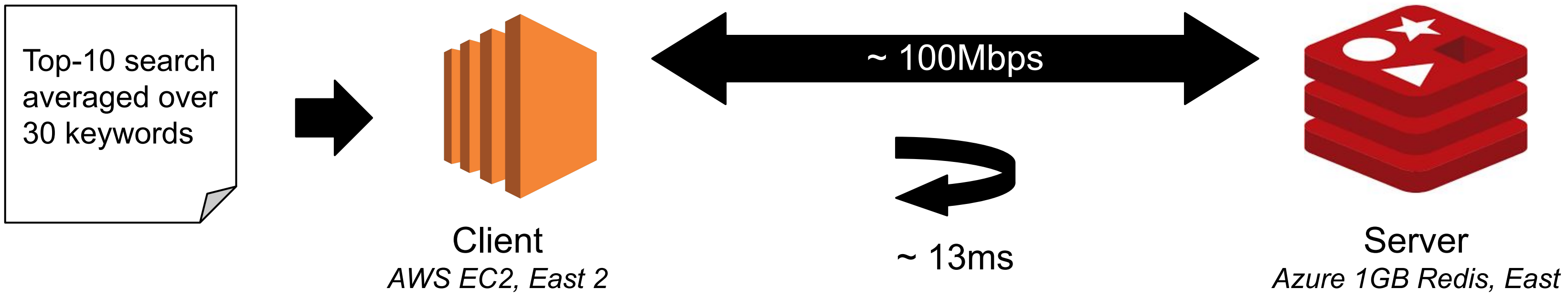
Search Performance - Latency



	Latency (sec)		
	10% Enron	50% Enron	100% Enron
Full	1.61	6.80	12.80
VPart	0.78	1.69	2.47
VHPart-10	0.16	0.33	0.46

Sub-second Latency

Search Performance - Latency



Latency (sec)

	10% Enron	50% Enron	100% Enron
Full	1.61	6.80	12.80
VPart	0.78	1.69	2.47
VHPart-10	0.16	0.33	0.46
CTR-DSSE	1.71	4.83	--

- Cold start ⇒ O(|W|) counters downloaded for every search
- Download all matches

Much more in the paper

- How to handle updates
- Progressive construction that transitions from Full to VPart, then from VPart to VHPart based on the number of postings
- More evaluation
  - Ubuntu and NYTimes datasets
  - Performance of search with index merge, update w/ and w/o index merge
  - Search quality based on the normalized discounted cumulative gain (NDCG)
  - End-to-end evaluation with synthetic workloads
- Formal security proofs
- Leakage-abuse analysis of the leakage

Thank you!

Min Xu
xum@uchicago.edu

Armin Namavari
ajn88@cornell.edu

David Cash
davidcash@uchicago.edu

Thomas Ristenpart
ristenpart@cornell.edu

References

- [KPR’12] S. Kamara, C. Papamanthou, T. Roeder. Dynamic Searchable Symmetric Encryption. CCS 2012
- [IKK’12] M. S. Islam, M. Kuzu, M. Kantarcioglu. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. NDSS 2012
- [CGPR’15] D. Cash, P. Grubbs, J. Perry, T. Ristenpart. Leakage-Abuse Attacks Against Searchable Encryption. CCS 2015
- [ZKP’16] Y. Zhang, J. Katz, C. Papamanthou. All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. USENIX Security 2016
- [BMO’17] R. Bost, B. Minaud, O. Ohrimenko. Forward and Backward Private Searchable Encryption from Constrained Cryptographic Primitives. CCS 2017
- [KKLPK’17] K. S. Kim, M. Kim, D. Lee, J. H. Park, W.-H. Kim. Forward Secure Dynamic Searchable Symmetric Encryption with Efficient Updates. CCS 2017
- [EKPE’18] M. Etemad, A. Küpçü, C. Papamanthou, D. Evans. Efficient Dynamic Searchable Encryption with Forward Privacy. PoPETS 2018
- [CPPJ’18] J. G. Chamani, D. Papadopoulos, C. Papamanthou, R. Jalili. New Constructions for Forward and Backward Private Symmetric Searchable Encryption. CCS 2018
- [KMO’18] S. Kamara, T. Moataz, O. Ohrimenko. Structured Encryption and Leakage Suppression. CRYPTO 2018
- [PPYY’19] S. Patel, G. Persiano, K. Yeo, M. Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. CCS 2019
- [BKM’20] L. Blackstone, S. Kamara, T. Moataz. Revisiting Leakage Abuse Attacks. NDSS 2020
- [PWLP’20] R. Poddar, S. Wang, J. Lu, R. A. Popa. Practical Volume-Based Attacks on Encrypted Databases. EuroSP 2020
- [DPPS’20] I. Demertzis, D. Papadopoulos, C. Papamanthou, S. Shintre. SEAL: Attack Mitigation for Encrypted Databases via Adjustable Leakage. USENIX Security 2020
- [GPPW’20] Z. Gui, K. G. Paterson, S. Patranabis, B. Warinschi. SWiSSSE: System-Wide Security for Searchable Symmetric Encryption. ePrint 2020
- [SOPK’21] Z. Shang, S. Oya, A. Peter, F. Kerschbaum. Obfuscated Access and Search Patterns in Searchable Encryption. NDSS 2021
- [OK’21] S. Oya, F. Kerschbaum. Hiding the Access Pattern is Not Enough: Exploiting Search Pattern Leakage in Searchable Encryption. USENIX Security 2021