

Security of Searchable Encrypted Cloud Storage

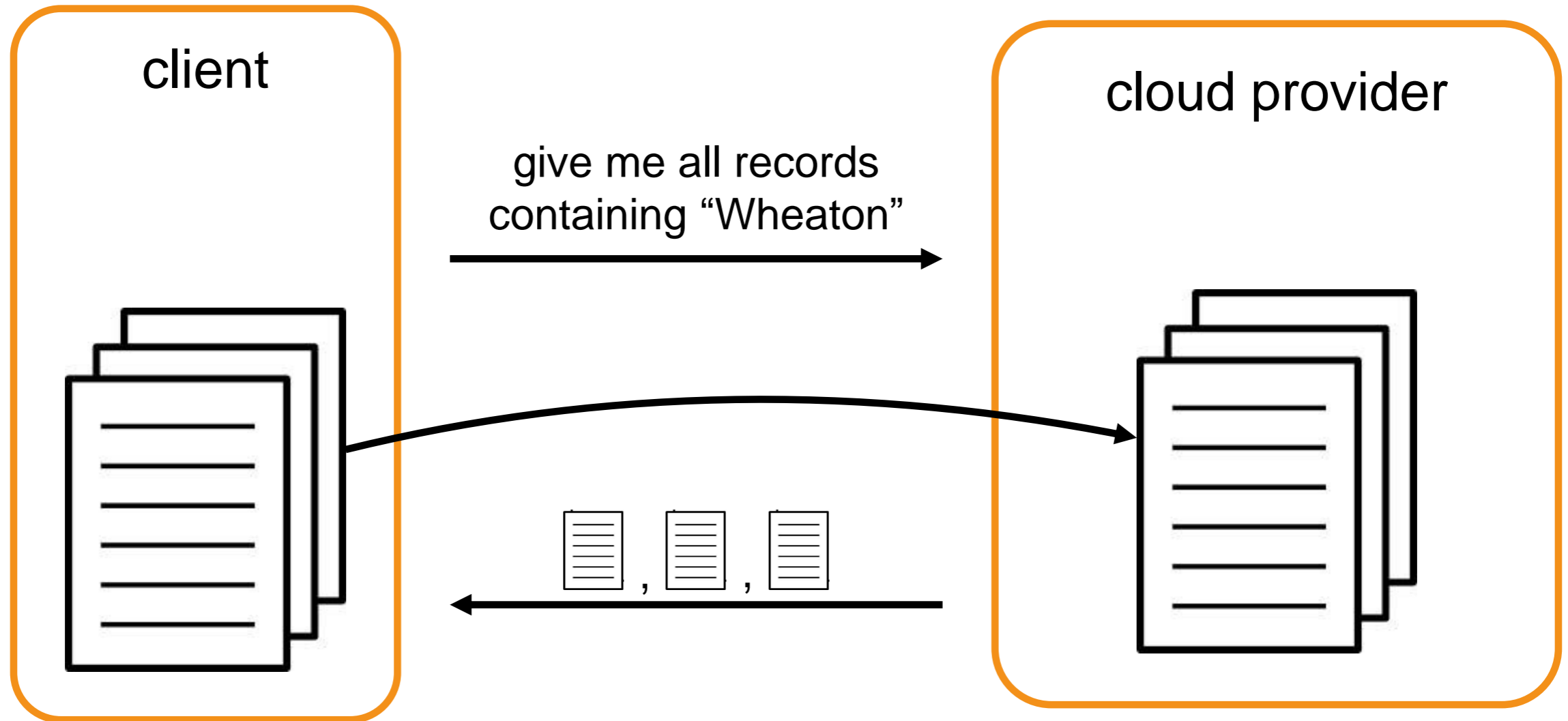
David Cash
Rutgers University

Paul Grubbs
Cornell University

Jason Perry
Lewis University

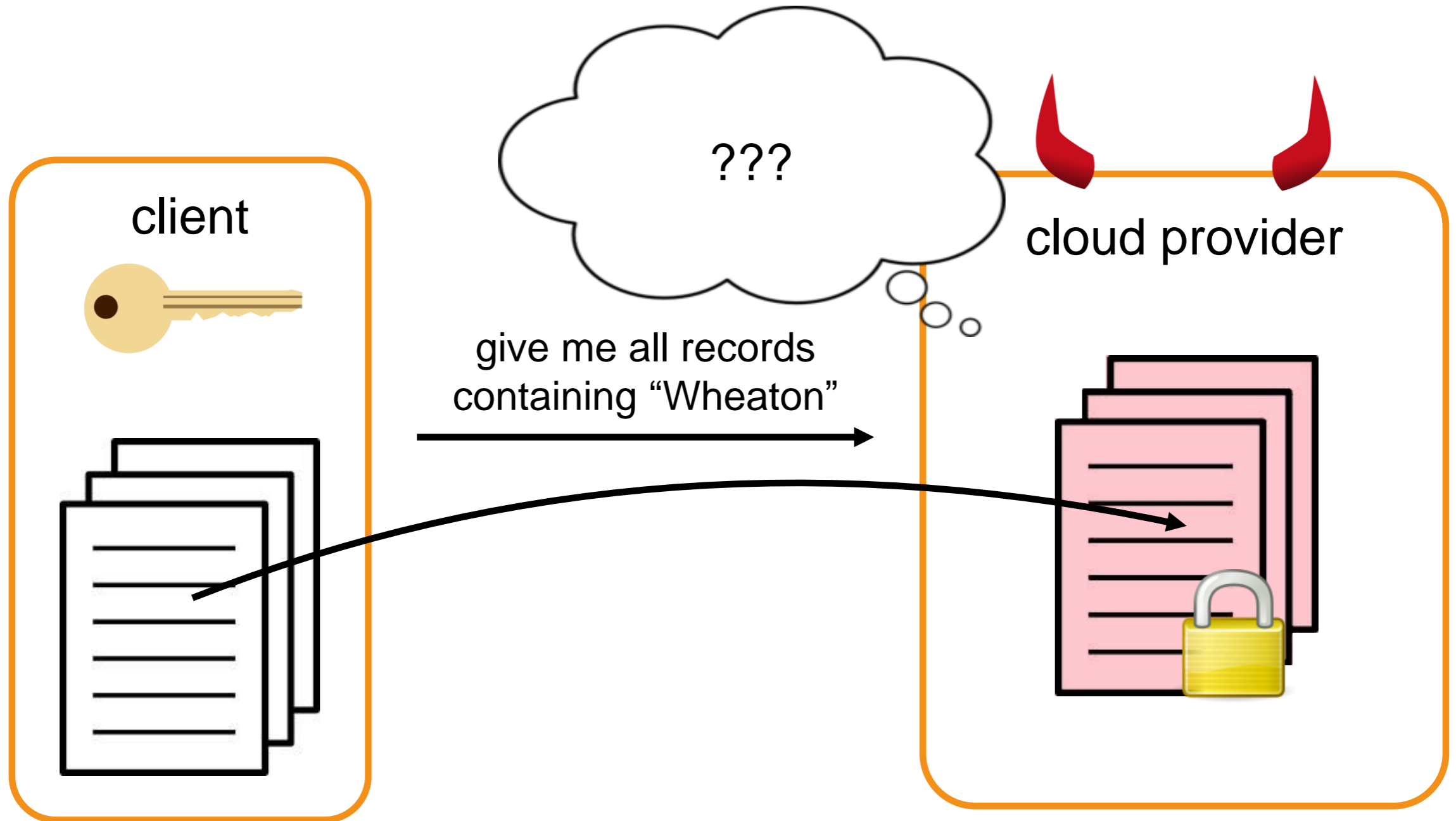
Tom Ristenpart
Cornell Tech

Outsourced storage and searching



- “records” could be emails, text documents, Salesforce records, ...
- searching is performed efficiently in the cloud via standard techniques

End-to-end encryption breaks searching

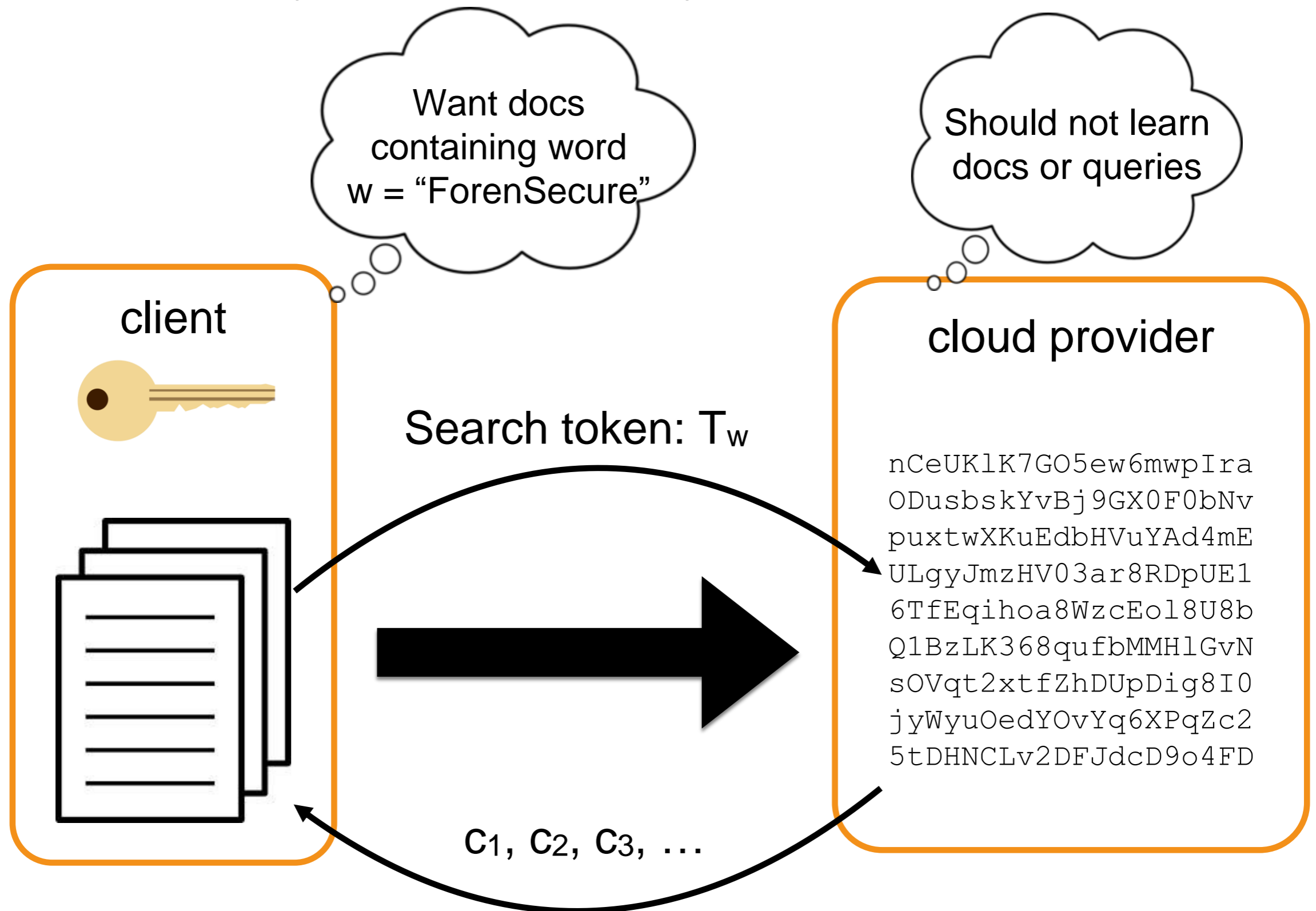


- Searching incompatible with privacy goals of traditional encryption

Companies actively trying to navigate this tension,
providing **Searchable Encryption**



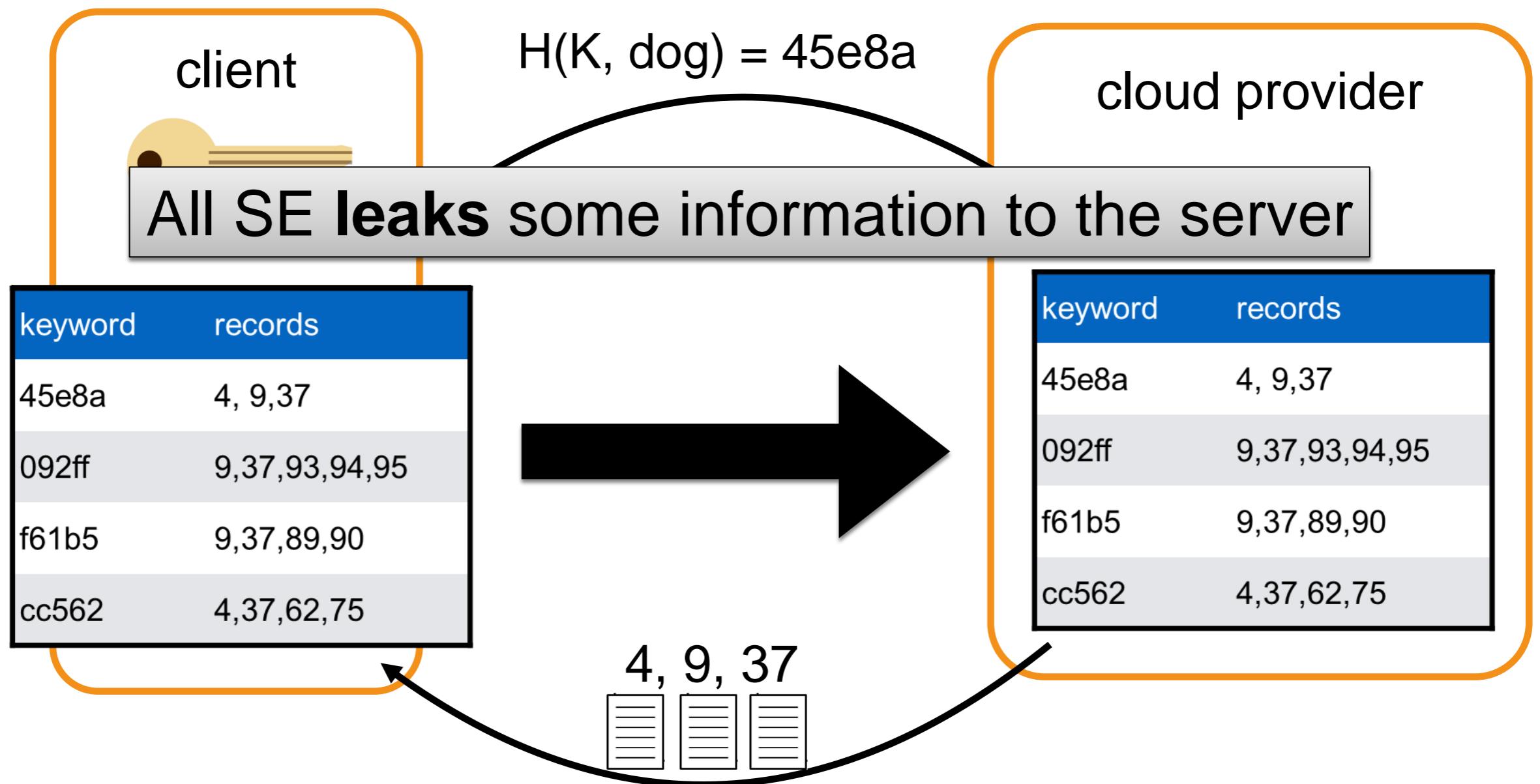
Searchable Symmetric Encryption [SWP'00, CGKO'06, ...]



- Also includes *Update Protocol* for adding new documents/records

Example SSE Construction

Client uploads an *encrypted inverted index*



What does SSE leakage look like?

Keywords/data not in the clear, but some info can be derived by server

keyword	records
45e8a	4, 9,37
992ff	9,37,93,94,95
f61b5	9,37,89,90
cc562	4,37,62,75

“this keyword is the most common”

“document #37 contains every keyword, and appears together with #9 often”



Highly unclear if/when this leakage is dangerous

How SE is analyzed in the literature

Crypto security definitions usually formalize e.g.:

“**nothing** is leaked about the input, except size”

SE uses a weakened type of definition:

- Identify a formal “leakage function” \mathcal{L}
- Allows server to learn info corresponding to \mathcal{L} , but no more

Example \mathcal{L} outputs:

- Result lengths (number of records matching a query term)
- Repeated Queries
- Repeated record IDs across search results (access pattern)
- Update information

Main question:

What can the cloud provider learn about your data from this leakage?

Currently almost no guidance in the literature.

A messy question that depends on many variables.

- Type of data, size of dataset, how data is processed, what queries look like, update frequency, adversary knowledge, attacker goal, etc., etc.

One prior work: Identifying queries

Under certain circumstances, a high percentage of queries (80%) can be deduced by a curious server

who knows the contents of all of the documents that were encrypted.

[Islam-Kuzu-Kantarcioglu '12]

(sketched later)

This work: Practical Exploitability of SE Leakage

Broaden and systematize the approach of [IKK '12]:

1. **Different adversary goals:** Document (record) recovery in addition to query recovery
2. **Different levels of adversary knowledge:** (full, partial, and distributional)
3. **Active adversaries:** planted documents

Simple **leakage-abuse attacks** for query recovery and document recovery, with experiments

- Attacks apply to all constructions with the same or greater *leakage profile*

Datasets for Attack Experiments

Enron Emails



- 30109 documents from employee `sent_mail` folders (to focus on intra-company email)
- When considering 5000 indexed keywords, average of 93 keywords/document

Apache Emails



- 50582 documents from Lucene project's `java-user` mailing list
- With 5000 keywords, average of 291 keywords/document

Processed with standard IR keyword extraction techniques
(Porter stemming, stopword removal)

Outline of Results

1. Simpler query recovery
2. Document text recovery from partial server content knowledge
3. Document text recovery via active attack (planted documents)

Query recovery using document knowledge [IKK '12]

Attack setting:

- Minimal leakage: Only record IDs matching each query, nothing revealed beforehand
- Server knows all document content ← e.g., public financial data
- k random queries issued
- Adversary Goal: Learn the queries

Server Observes:

keyword	records
Q1	4,37,62,75
Q2	9,37,93,94,95
Q3	4, 9,37
Q4	8,37,89,90
⋮	

Unknown term-doc matrix:

	rec1	rec2	rec3	rec4	
Q1	1				
Q2		1			
Q3		1	1	1	...
Q4	1		1		
Q5	1			1	
Q6	1				
					⋮

The IKK attack (sketch)

Term Co-occurrence matrix:

	Q1	Q2	Q3	Q4	
Q1	2				
Q2		1			
Q3		1	3	1	...
Q4	1		4		
Q5	3			1	
Q6	1				
		⋮			

- Server constructs term-term co-occurrence matrix
- Attempts to solve optimization problem of matching to known term co-occurrence matrix
- Over 80% of queries correctly identified for small index sizes (1500 keywords)

The IKK attack works only if the server has highly accurate knowledge of the document set

If that's the case, then why not just look at the *number* of document IDs returned by a query?

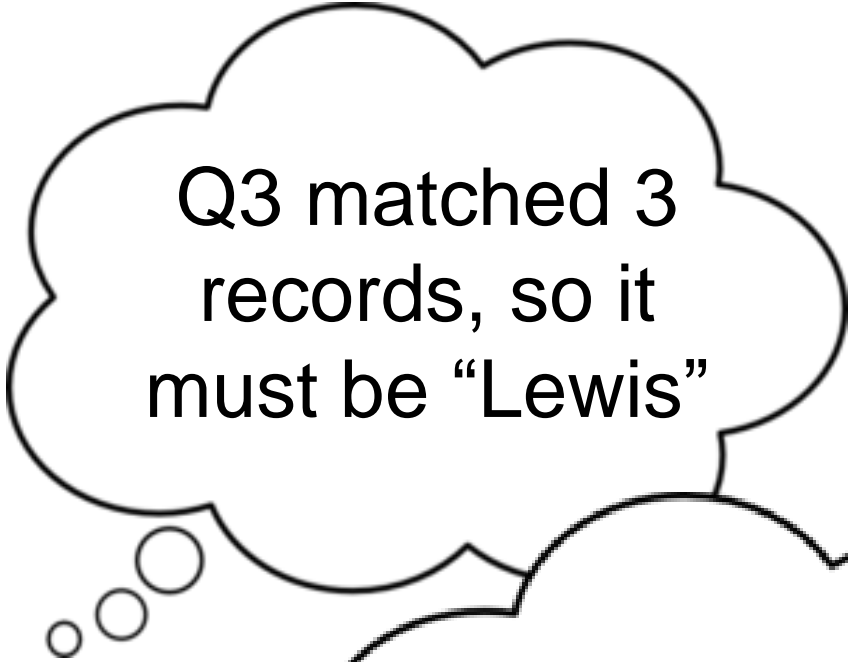
Observation: When there's only one query that returns a certain number of documents, then the server can immediately identify that query.

Query Recovery via Result Counts

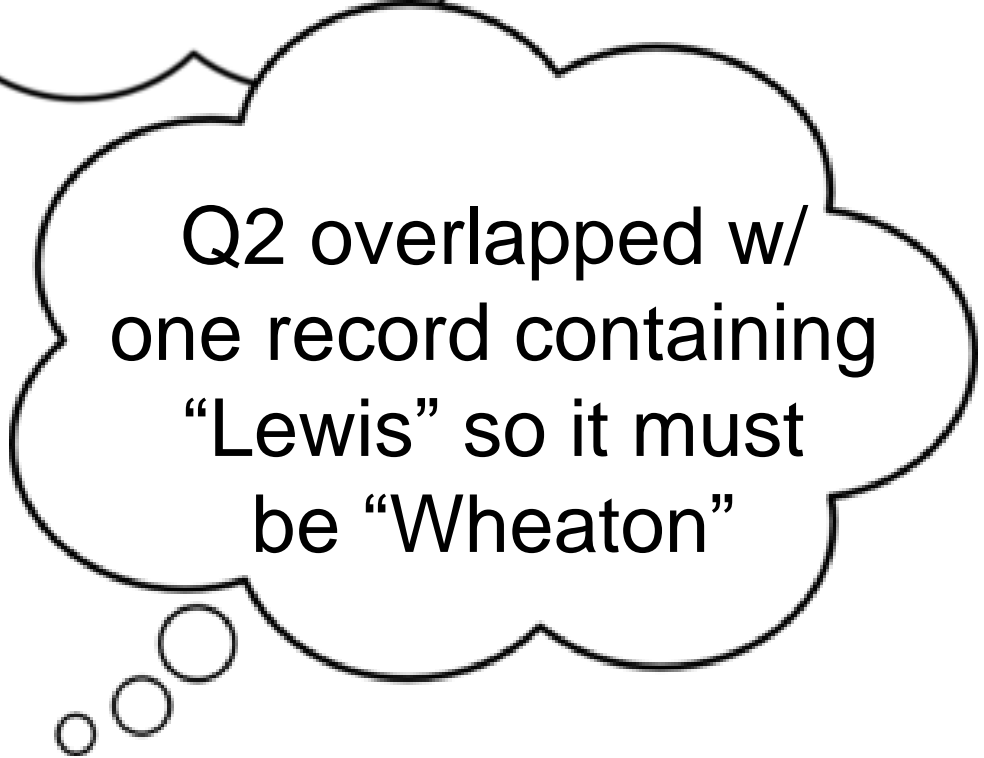
- After finding unique-match queries, we then “disambiguate” remaining queries by checking intersections

Leakage:

	rec1	rec2	rec3	rec4
Q1	1			
Q2		1		
Q3		1	1	1
Q4	1		1	
Q5	1			1
Q6	1			



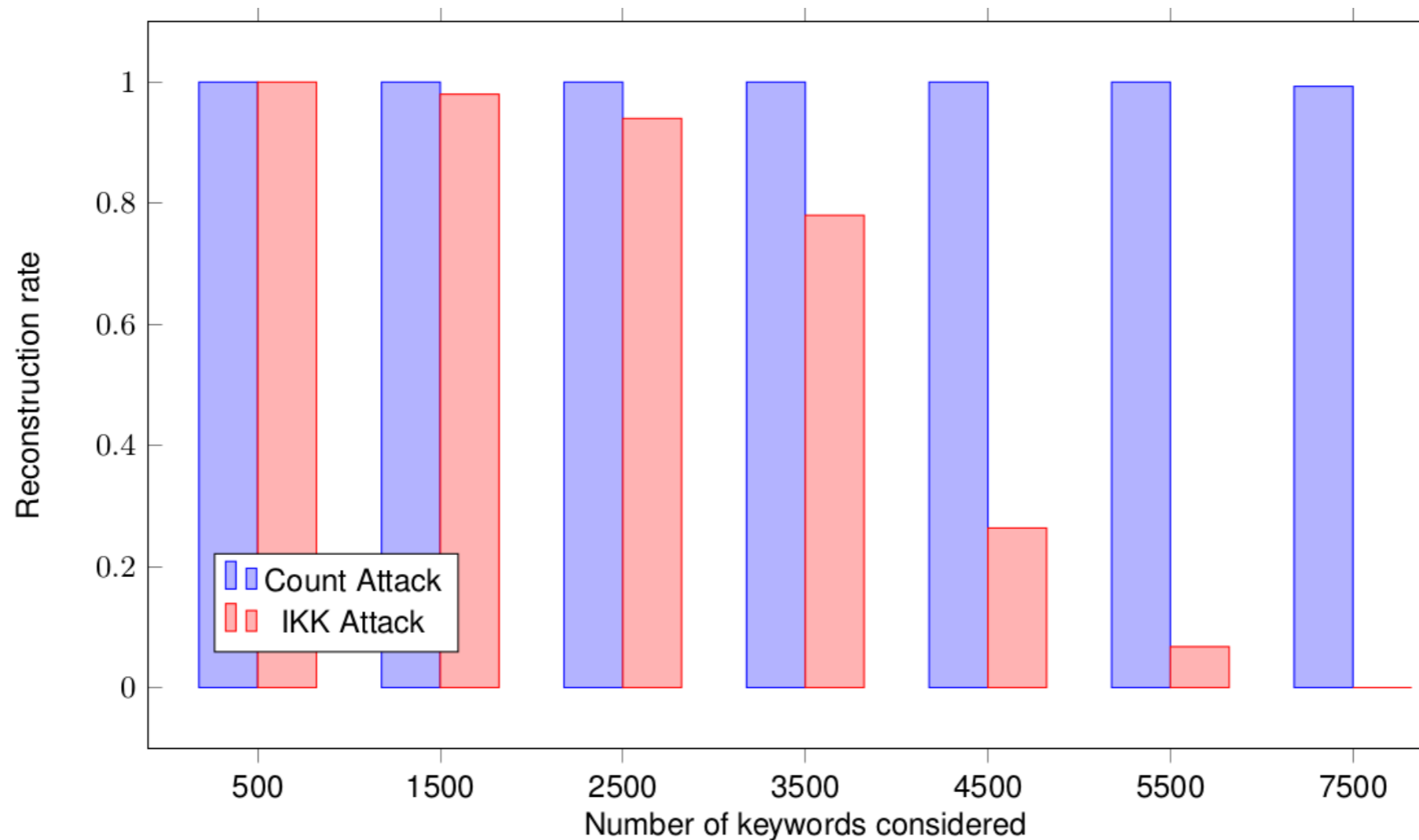
Q3 matched 3 records, so it must be “Lewis”



Q2 overlapped w/ one record containing “Lewis” so it must be “Wheaton”

Query Recovery Experiment

- Setup:
- Enron email dataset
 - 10% queried at random

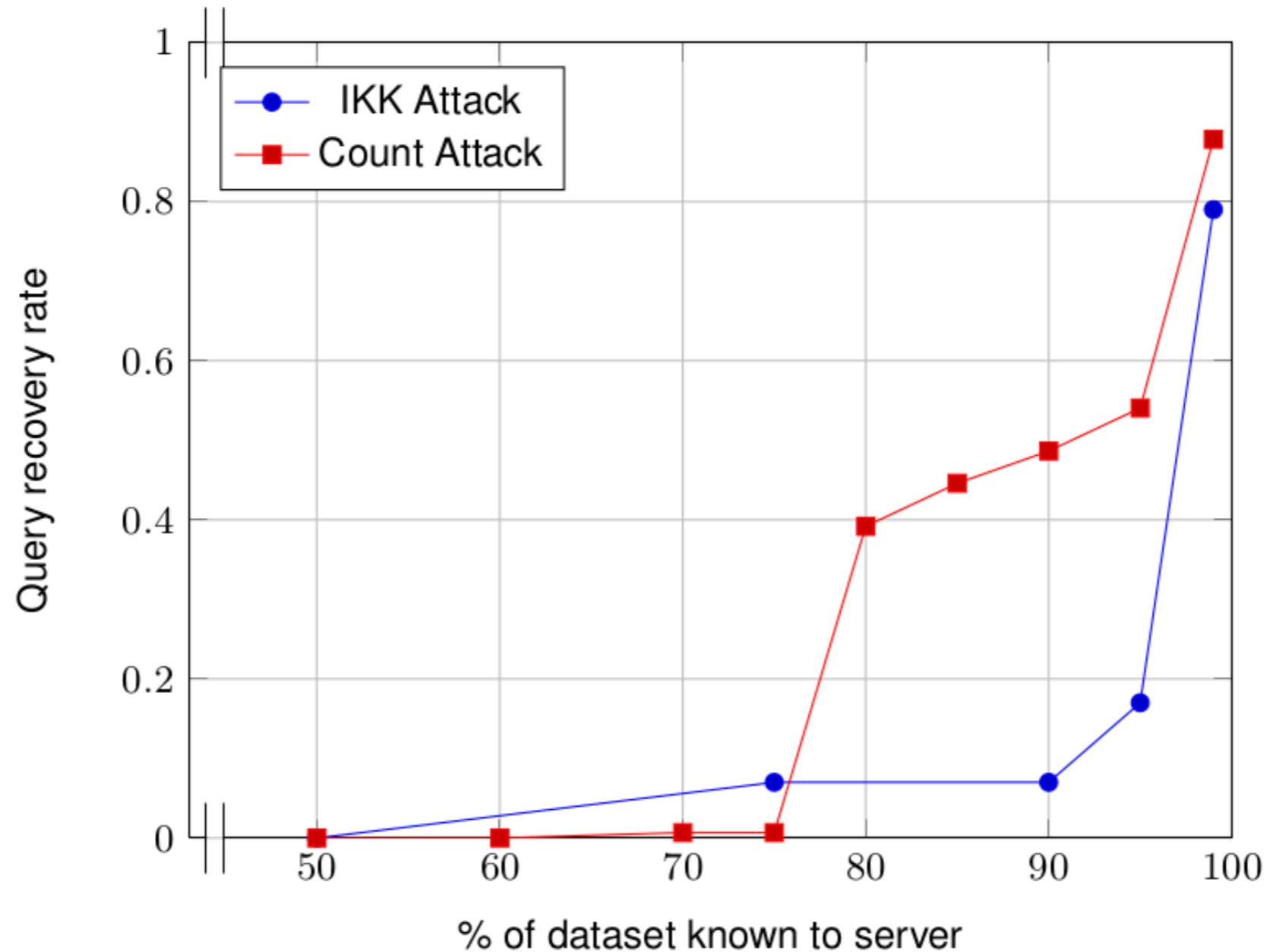


Runs in seconds, not hours

Query Recovery with Partial Knowledge

- What if document set is only partially known to the server?
- We generalized the counting attack to account for partial information, and tested the count and IKK attacks when only $x\%$ of the documents are known

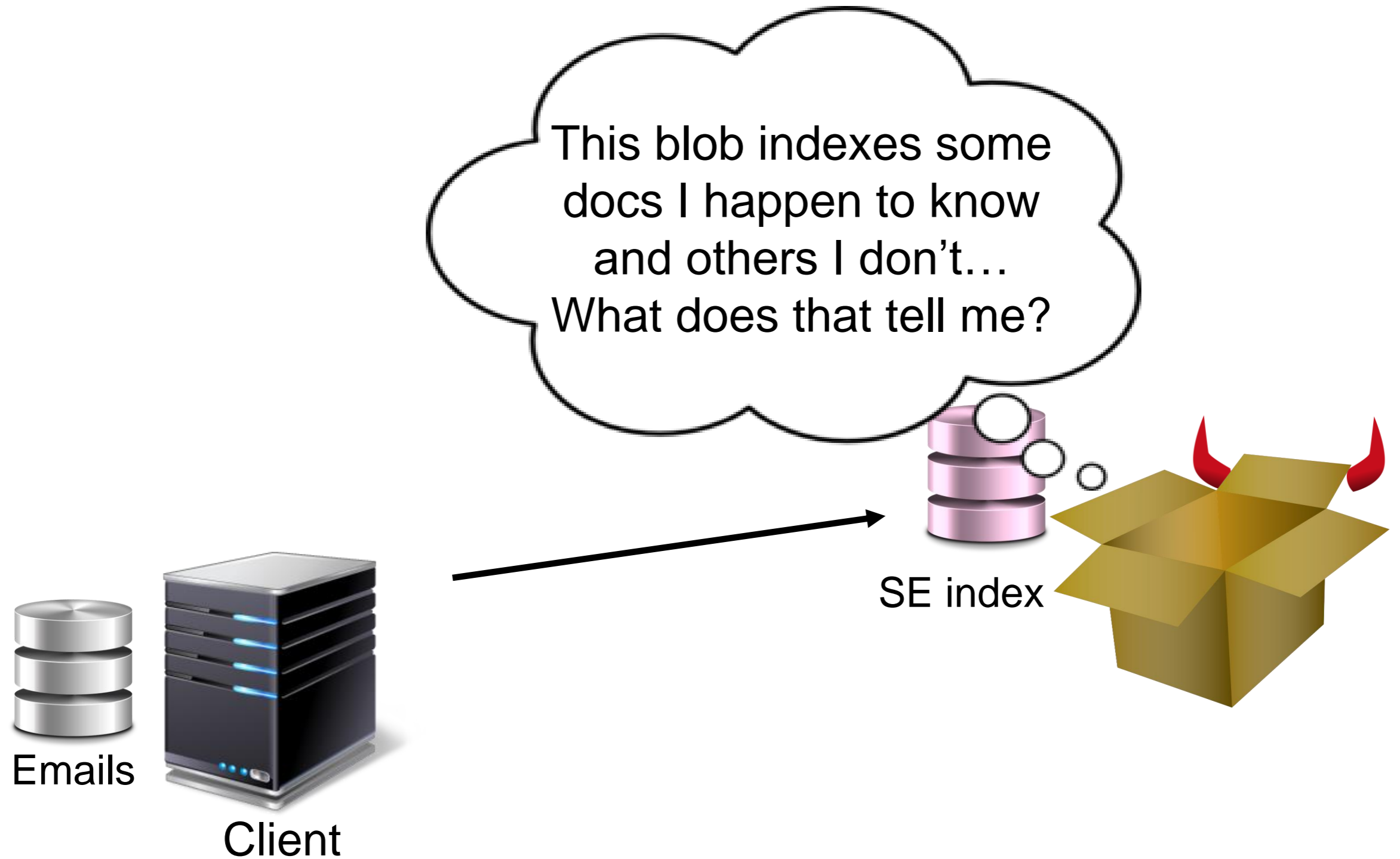
Query Recovery with Partial Knowledge



Enron emails, 500 most frequent keywords indexed (stemmed, non-stopwords), 150 queried at random, 5% of queries initially given to server as hint

1. Simpler query recovery
- 2. Document recovery from partial knowledge**
3. Document recovery via active attack

Document Recovery using Partial Knowledge

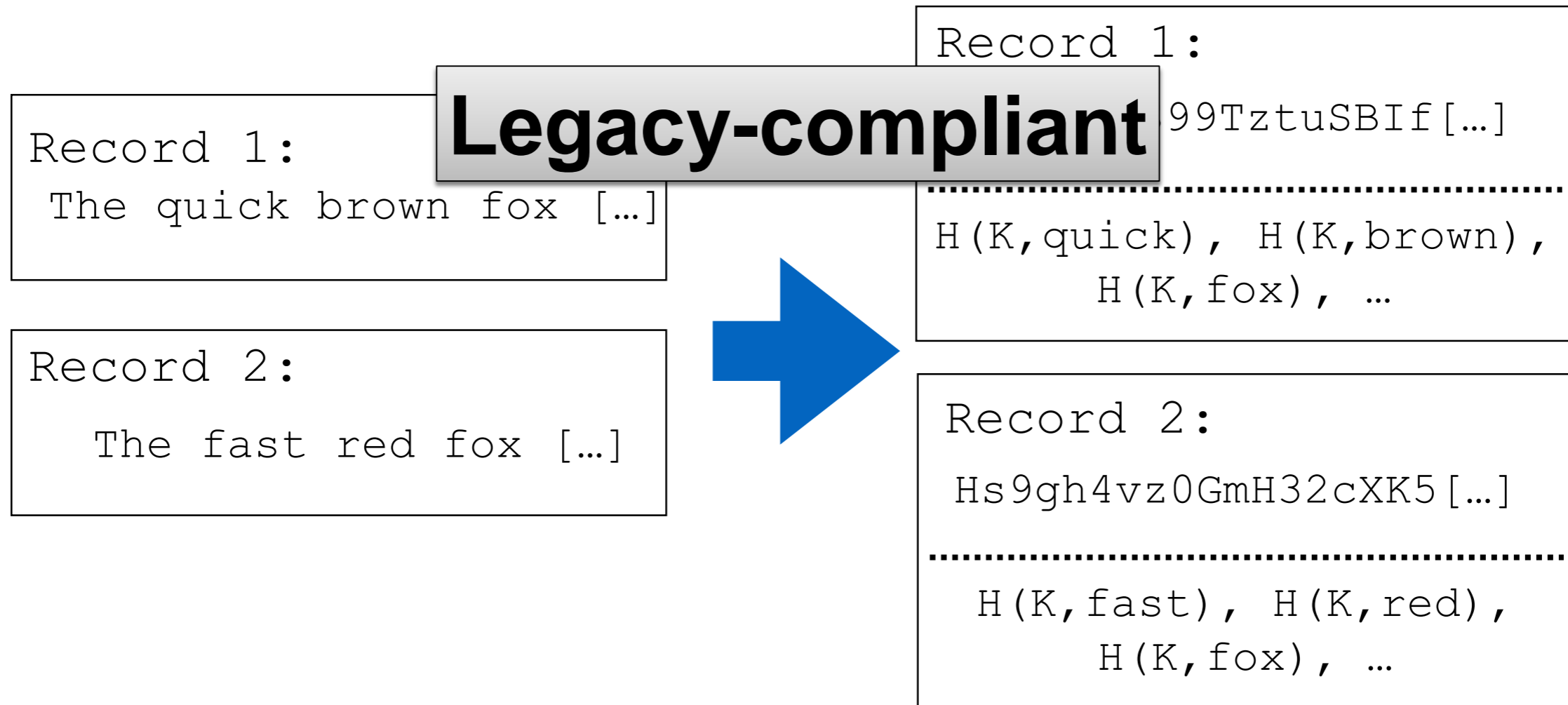


Passive Document Recovery Attack Setting

- No queries issued at all
- Content of some documents becomes known to the server
- **Attacker Goal:** Recover other document contents

A Widely-Used Construction

- Attack works on the leakage from “appended keyword hash” constructions:



Actual systems:

- Mimesis [Lau et al'14]
- Shadowcrypt [He et al'14]

Simple Observation

Known:

```
Doc 1:  
zAFDr7ZS99TztuSBIf [...]  
-----  
H(K, quick), H(K, brown),  
H(K, fox), ...
```

Unknown:

```
Doc 2:  
zAFDr7ZS99TztuSBIf [...]  
-----  
H(K, fast), H(K, red),  
H(K, fox), ...
```

- If server knows Doc 1, then learns when any word in Doc 1 appears in other docs
- More revealing case: Hash values stored in word order of first appearance.
- Harder but still possible if hash in random order. (see paper)

Document Recovery from Partial Knowledge

For each dataset, server knowing either 2 or 20 random emails

Dataset, # Known Docs	Average Keywords Recovered / Doc
Enron, 2	16.3%
Enron, 20	56.0%
Apache, 2	50.7%
Apache, 20	68.4%

Anecdotal Example

Original email:

The attached contract is ready for signature. Please print 2 documents and have Atmos execute both and return same to my attention. I will return an original for their records after ENA has signed. Or if you prefer, please provide me with the name / phone # / address of your customer and I will Fed X the Agreement.

Reconstructed in-order stems:

attach contract ___ signatur pleas print 2
document have ___ execut both ___ same ___
will origin ___ ena sign prefer provid name
___ ___ ___ ___ ___ agreement

- From Enron with 20 random known documents
- Note effect of stemming, stopword removal, and revealing each word once

The effect of one public document

Case study: A single email from the Enron corpus, sent to 500 employees

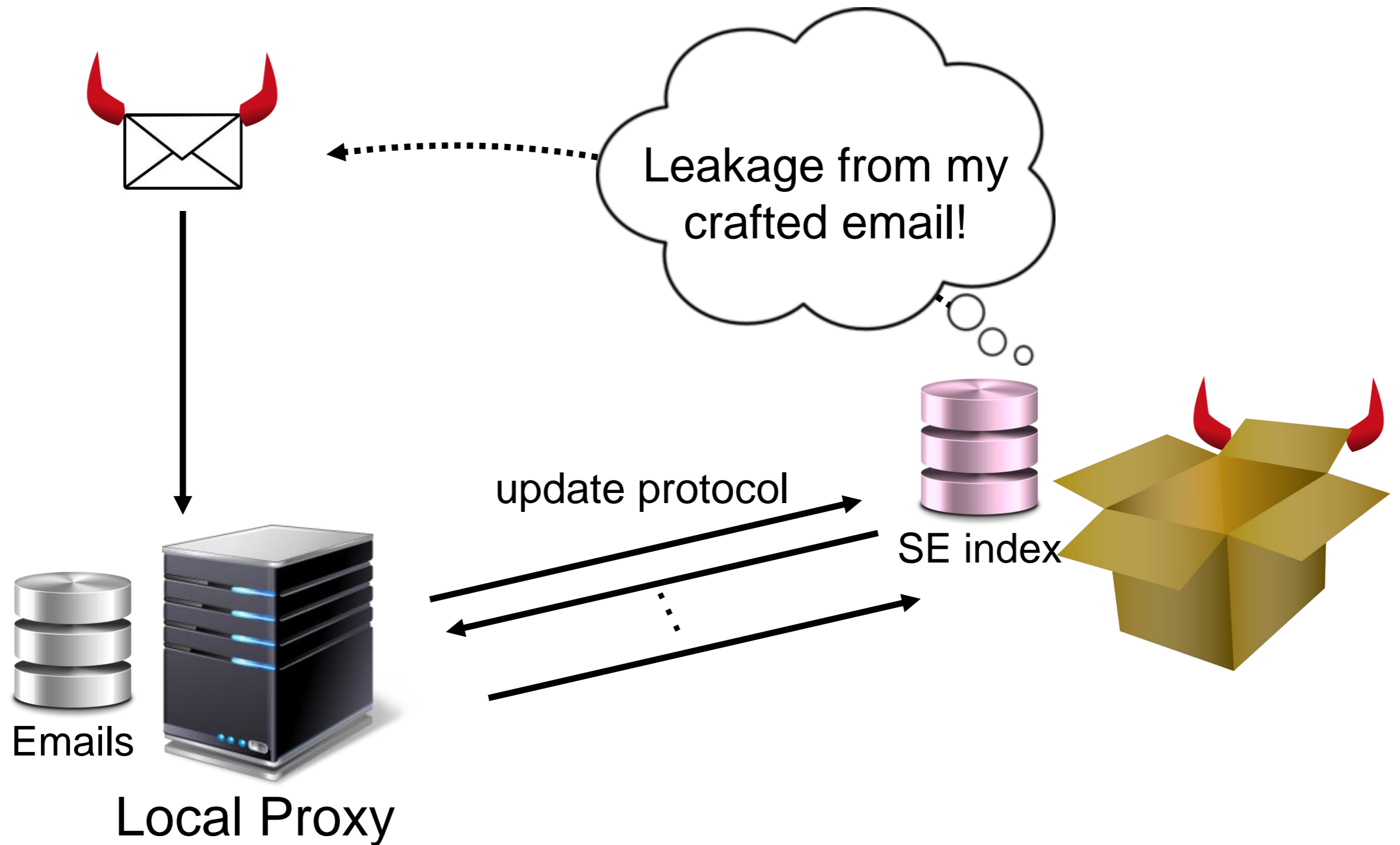
- 832 Unique Keywords
- Topic: an upcoming survey of the division by an outside consulting group.

The vocabulary of this single document would let a server recover on average 35% of the words in every document, not counting stopwords.

Outline

1. Simpler query recovery
2. Document recovery from partial knowledge
- 3. Document recovery via active attack**

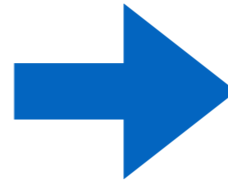
Chosen-Document-Addition Attacks



Chosen-Document Attack \Rightarrow Learn chosen hashes

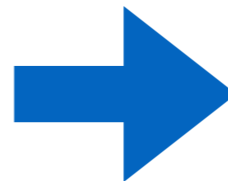
- Again we attack the appended-keyword-hash constructions

Doc 1:
The quick brown fox [...]



Doc 1:
zAFDr7ZS99TztuSBIf [...]
.....
H(K, quick), H(K, brown),
H(K, fox), ...

Doc 1:
The quick brown fox [...]



Doc 1:
zAFDr7ZS99TztuSBIf [...]
.....
H(K, fox), H(K, quick),
H(K, brown), ...

- Hashes stored in document order \Rightarrow very easy attack
- Hashes not in order \Rightarrow more difficult (we attack now)

Chosen Document Attack Experiment

Goal: Maximize number of keywords learned from a minimum number of chosen documents (emails)

Attack Sketch:

1. Malicious server constructs a series of emails, each with a small number of chosen keywords
2. Server observes insertion of planted emails into its index, seeing the appended hashes
3. Uses frequencies of a related corpus to deduce which hashes correspond to which keywords

Experimental results:

Server who plants emails with up to 10 keywords can identify hashes with <20% error rate

Conclusion

Systematic study of exploitability of multiple SE leakage types

- Servers have high capability to deduce query and document contents by observing client interactions
- Product claims must be sifted for potential compromises
- Need framework + experiments for understanding what one can do with leakage
- We've only scratched the surface...
 - Info retrieval and natural language methods!

Thank you!