Hiding in the Cloud: The Perils and Promise of Searchable Encryption

Jason Perry
Lewis University

In collaboration with:

David Cash
Rutgers University

Paul Grubbs
Cornell University

Tom Ristenpart
Cornell Tech

Cloud storage is everywhere and not going away




Advantages:

automatic synchronization

access from any device, any platform, anywhere

BUT….

What if the cloud provider gets hacked?

What if the cloud provider is the bad guy?

My private information and searches can be exposed.

BUT….

What if

What if

My priv

sign in search jobs US edition

theguardian

home › tech election 2016 US world opinion sports soccer arts lifesty all

Dropbox

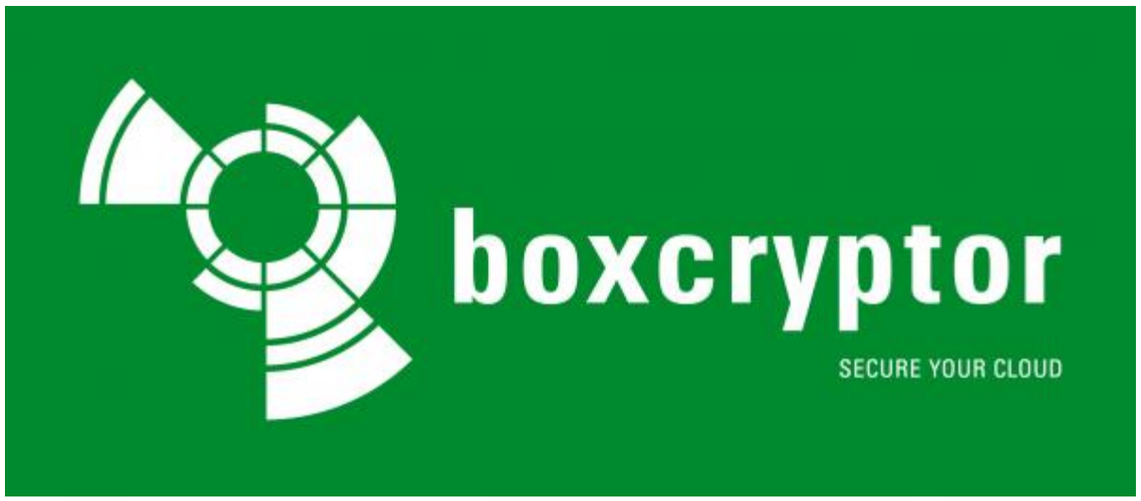
Dropbox hack leads to leaking of 68m user passwords on the internet

Data stolen in 2012 breach, containing encrypted passwords and details of around two-thirds of cloud firm's customers, has been leaked

I Know!

I'll **encrypt** my data with a **client-side key** before I upload it.

"End-to-end encryption"



But this presents major usability problems for files stored remotely, and seems to be a non-starter for interactive services such as email.

I Know!

I'll **encrypt** my data… upload it.

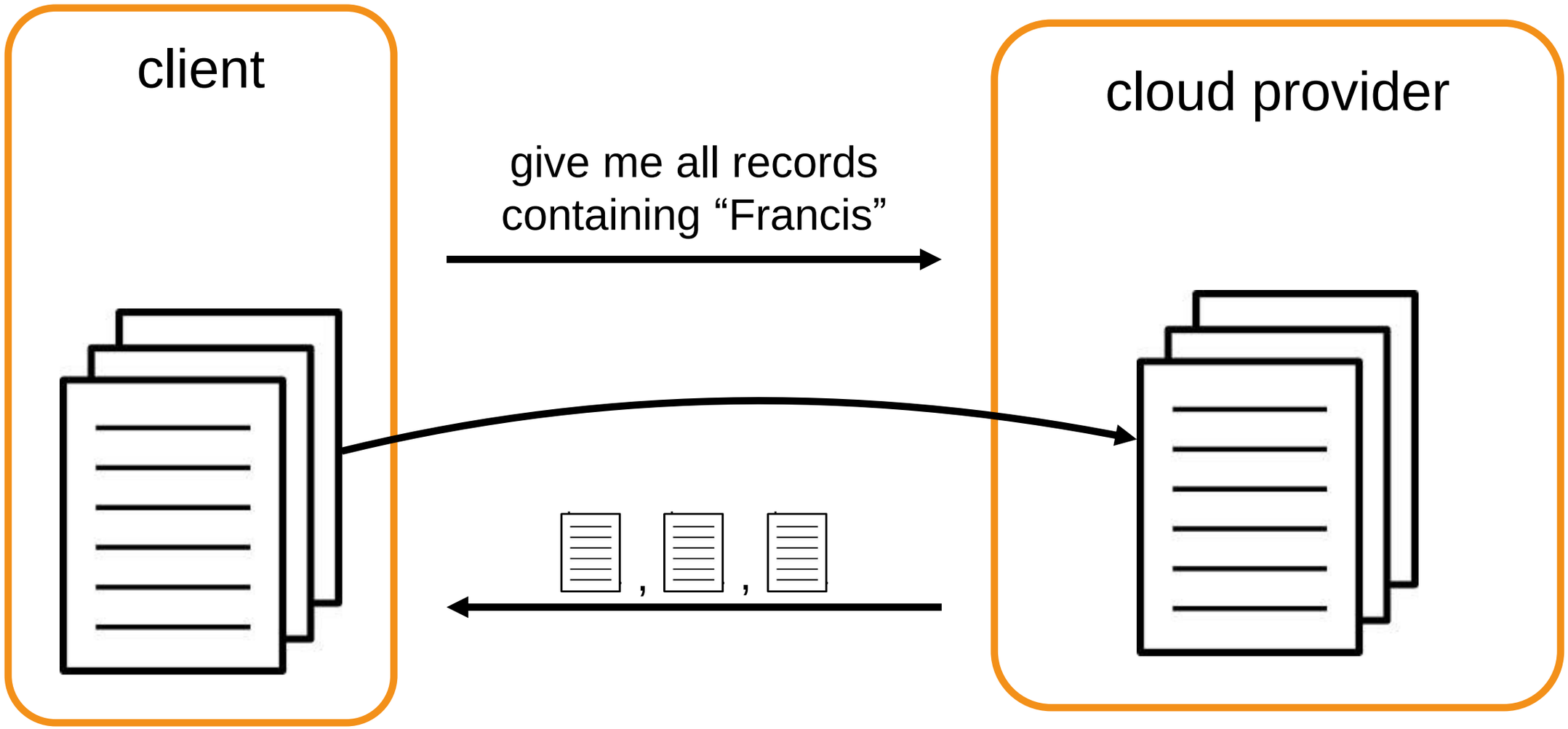
"End-to-e…

We want to **store encrypted data in the cloud** and search it while it's **still in the cloud** and **still encrypted.**

But this presents ma… is for files stored remotely, and seems to be a non-starter f… teractive services such as email.

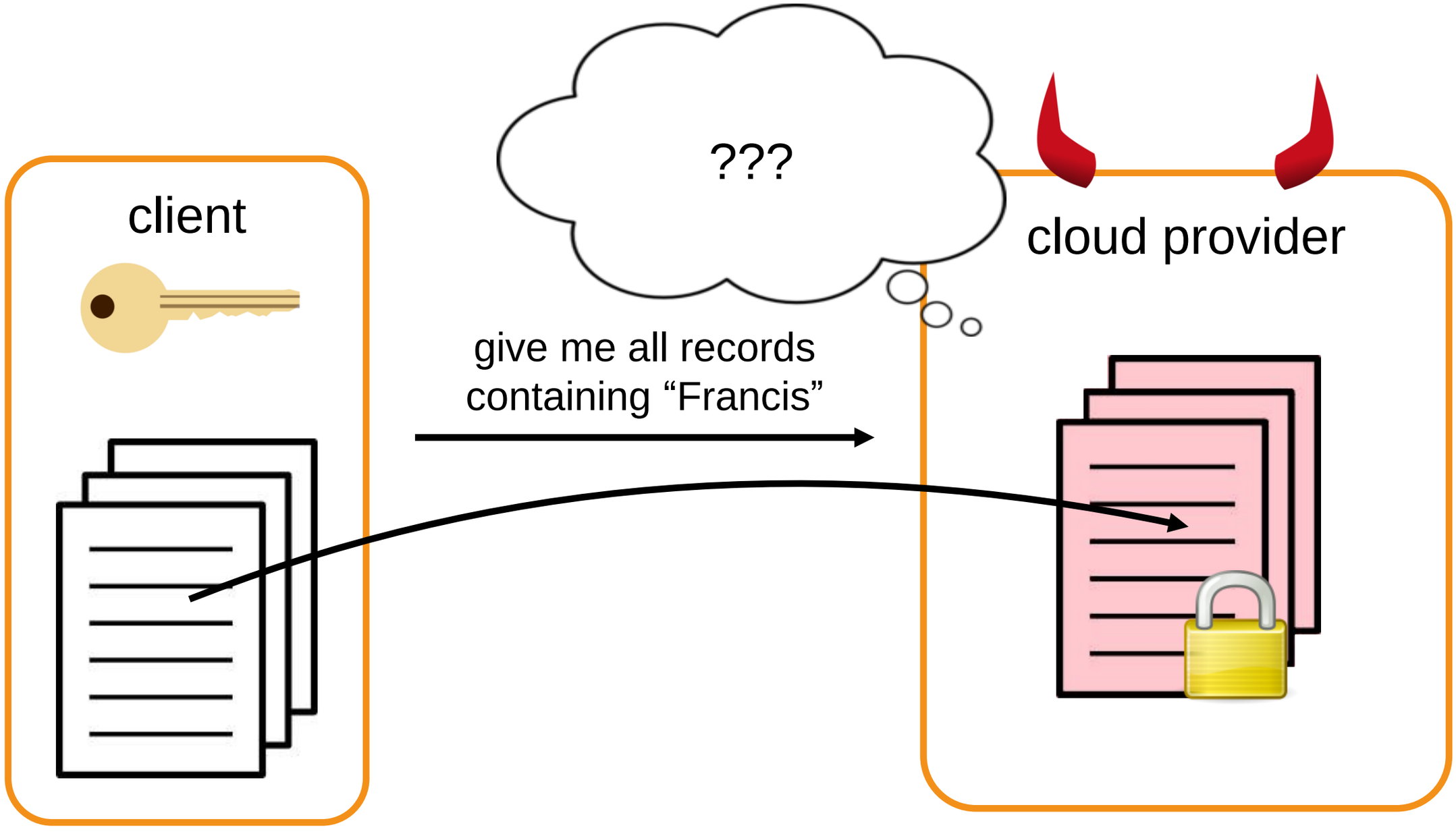
Companies actively promoting products to do this, providing **Searchable Encryption**

Outsourced storage and searching



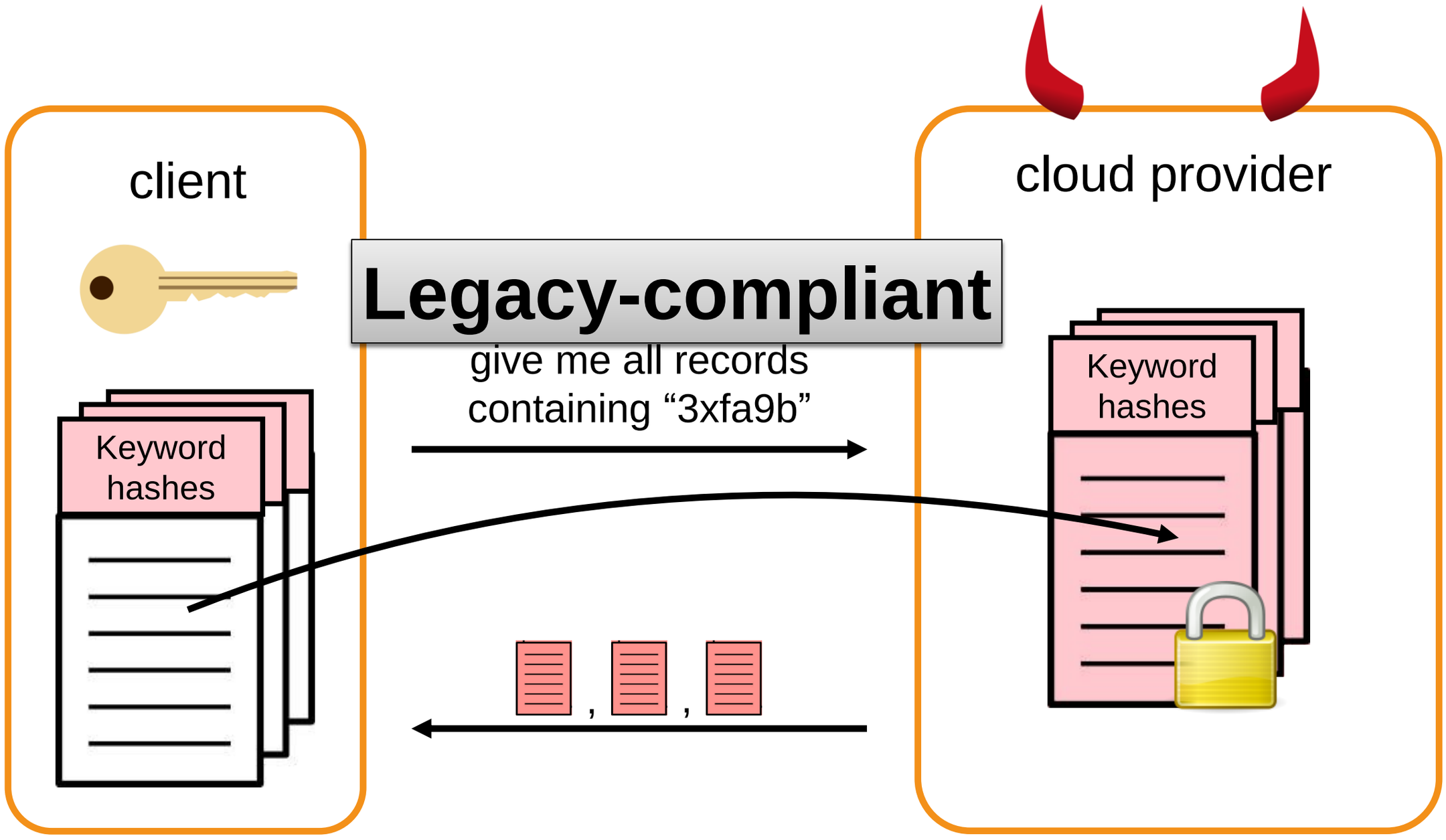
- “Records” could be emails, documents, …
- Searching is performed efficiently in the cloud via standard techniques

End-to-end encryption breaks searching



- Searching incompatible with privacy goals of traditional encryption

Idea 1



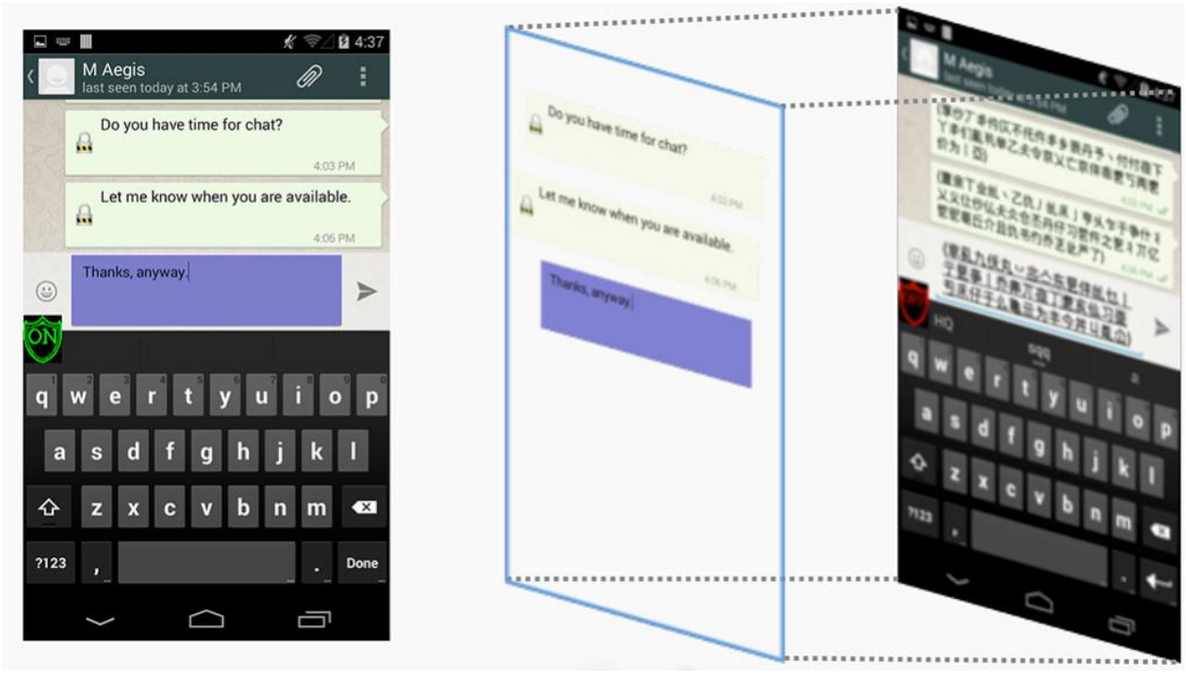
- Enable simple keyword searches by <u>appending keyed hashes of keywords</u> to the documents they match.

A Widely-Used Construction

- This “appended keyword hash” construction can be deployed on an existing service by the client, with no support from the server.
- A client-side program can “overlay” encryption on the existing system
- The service’s own searching capability is exploited to match the hash values

Used by actual products:

- **Mimesis** – overlays UI of android apps
- **Shadowcrypt** – browser extension to encrypt 14 popular web services



What the server saw

Server can immediately see the pattern of which keyword hashes appear in which documents, including words in common.

“this keyword is the most common”

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

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

This is an example of *leakage*.

How bad is it?

Crypto security definitions usually formalize e.g.:

"***nothing*** is leaked about the input, except size"

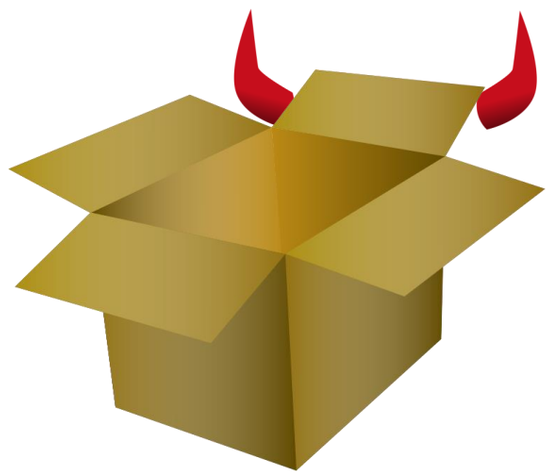
This is definitely worse than that.

But, **what can the cloud provider learn about <u>your</u> <u>data</u> from this leakage?**

Highly unclear if/when this leakage is dangerous

Investigating Leakage

We carried out a series of simulation experiments, using email datasets, to get a sense of the exploitability of leakage by a dishonest server.



The server observes the encrypted records and queries and gathers statistics, then attempts to *reconstruct* the client's documents or queries.

We will outline 3 attack experiments and the results.

Datasets for Attack Experiments

Enron Emails 

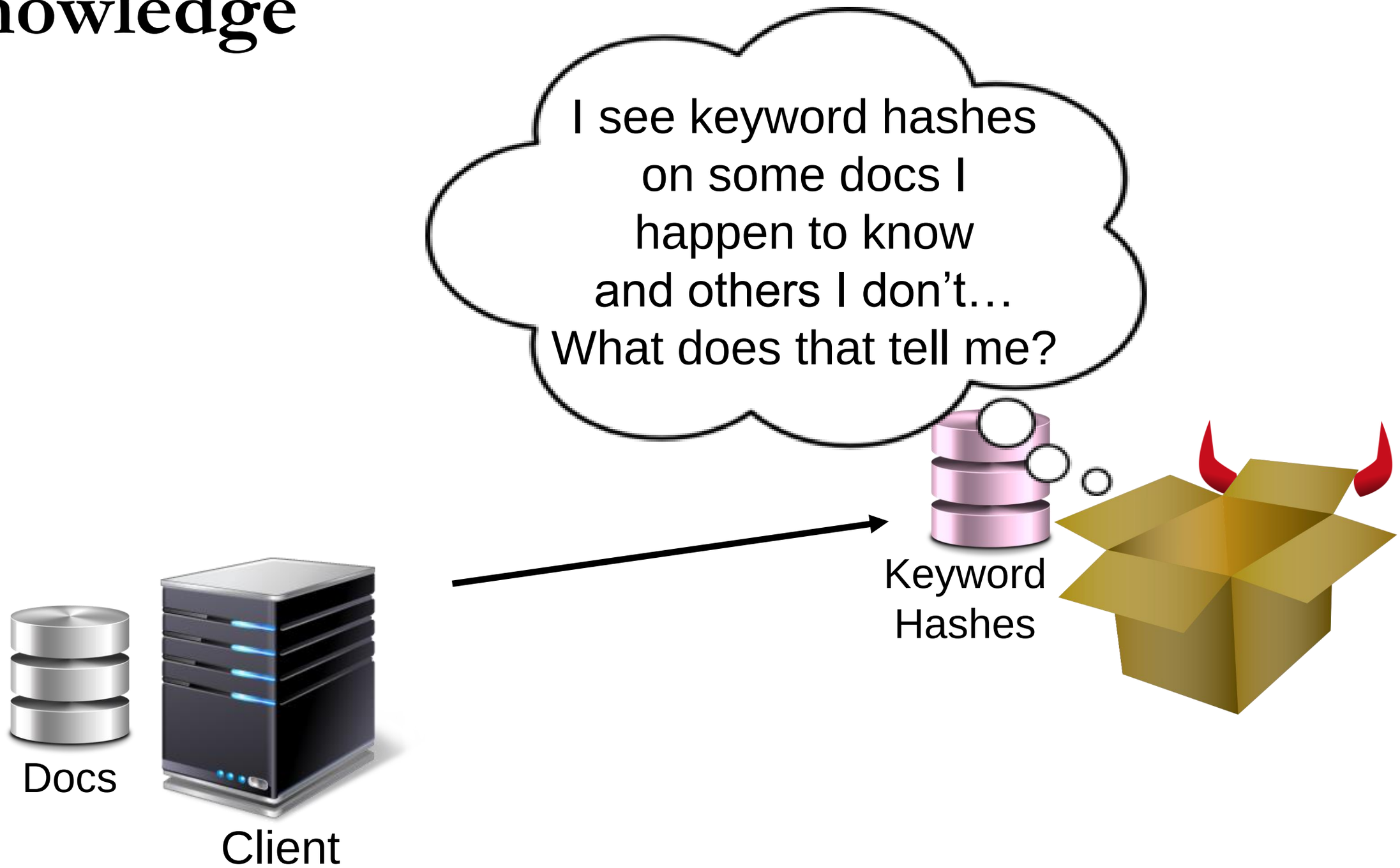
- 30109 documents from employee `sent_mail` folders (to focus on intra-company email)
- When considering 5000 indexed keywords, average of 93 keywords/document

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)

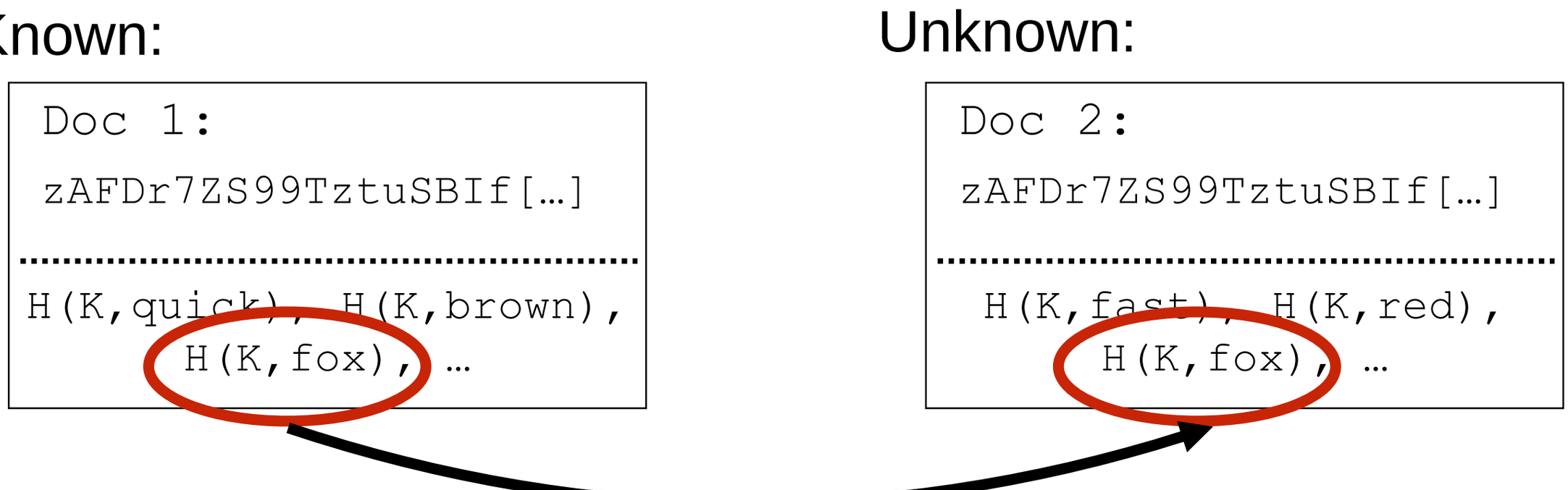
Attack 1: Document Recovery from Partial Knowledge



Passive Document Recovery Attack Setting

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

A Simple Observation



- A server who knows the contents of Doc 1 immediately knows which keywords the hashes are hashes of.
- If hash values are stored in word order of first appearance, server learns the exact correspondence of keywords to hashes.
- Server sees other documents with the same hash, knows the document contains the same word.
- Harder but still possible if hashes are in random order.

Document Recovery from Partial Knowledge

From knowing even a small number of documents, the server learns enough keyword hashes to “piece together” the contents of other documents.

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

For each dataset, server knowing either 2 or 20 randomly chosen emails

Example of in-order document reconstruction

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.
```

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

The effect of one public document

Documents with wide distribution are more likely to become known.

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.

Conclusion 1

Appended-hash constructions are not able to keep the contents of documents secret from a server that observes the hashes.

A Safer Construction

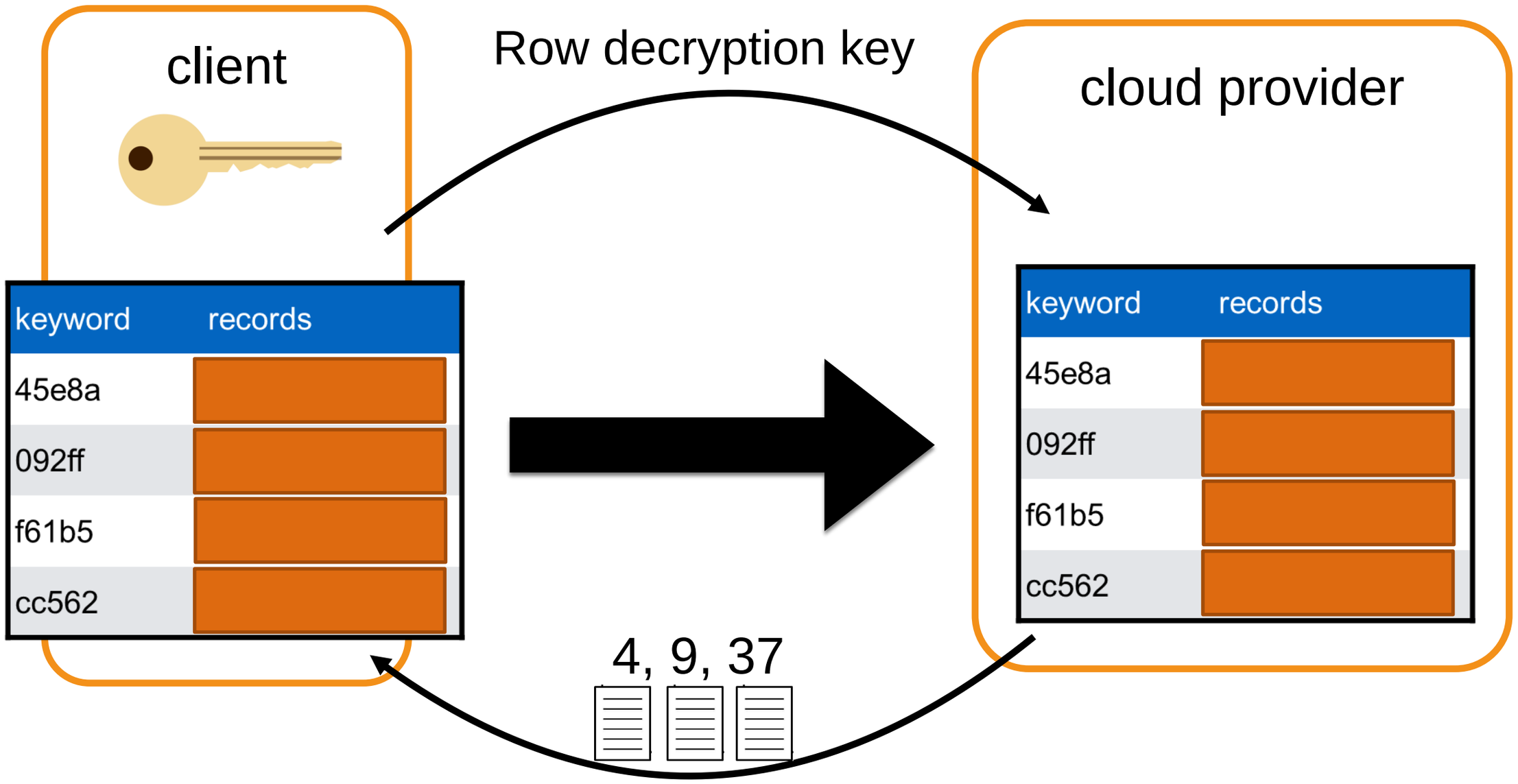
Research in searchable encryption has produced more sophisticated index structures, that keep the keyword-document mapping hidden until the time that a query is issued.

Each query reveals a little more of the keyword appearance pattern.

Searchable Symmetric Encryption

[SWP'00, CGKO'06, …]

Client uploads an *encrypted inverted index*



But there is still leakage!

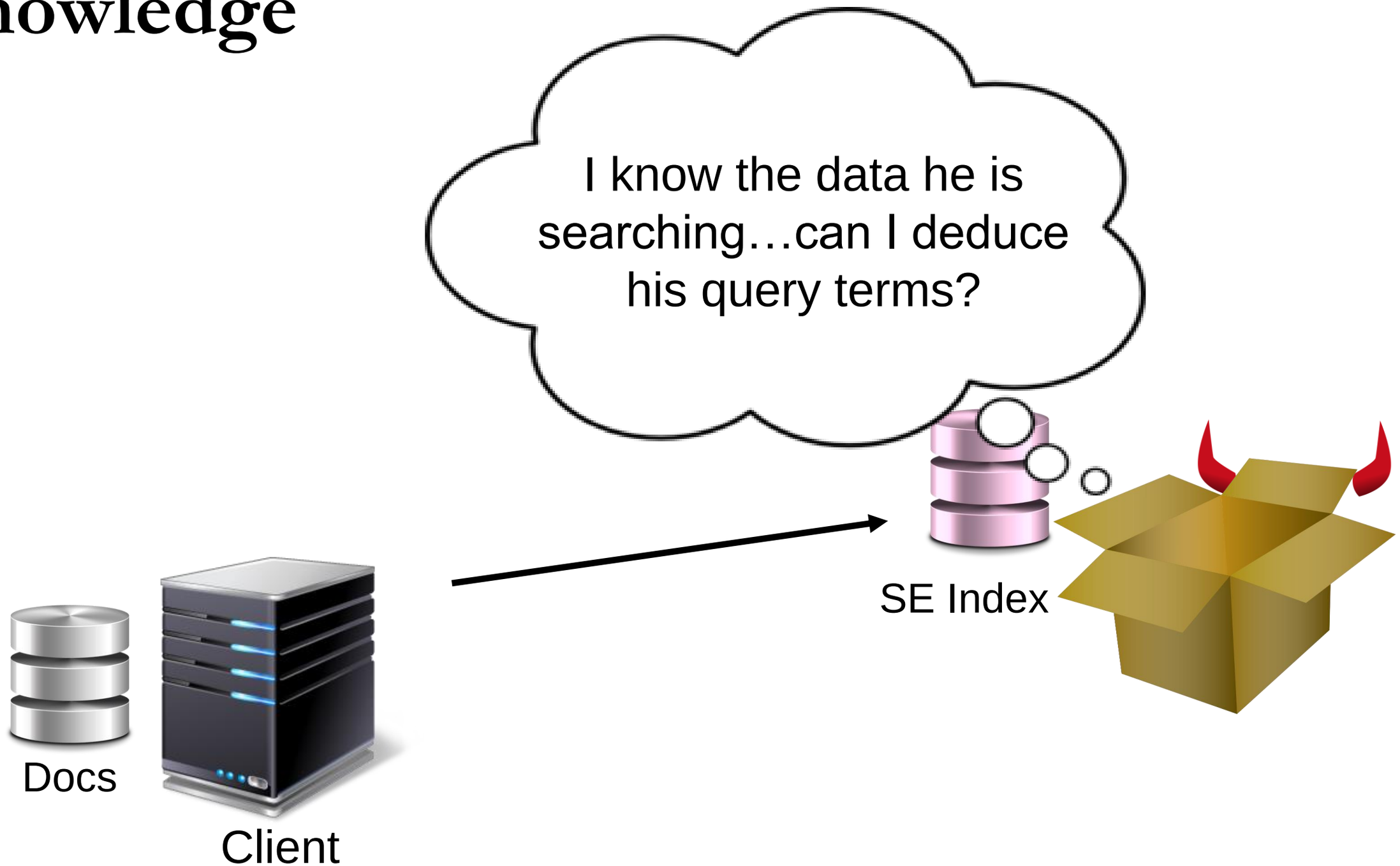
Access pattern is revealed progressively.

Attack experiment where the server already knows the contents of the documents, but attempts to learn what the client queried.

Example application: private searches of publicly known data.

A previous attack was able to reveal ~80% of query terms with small datasets, using a computationally intensive annealing algorithm to perform matching. [IKK‘12]

Attack 2: Query Recovery from Document Knowledge



Another simple idea

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.

We call this the "count attack"

Query Recovery via Result Counts

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

Leakage:

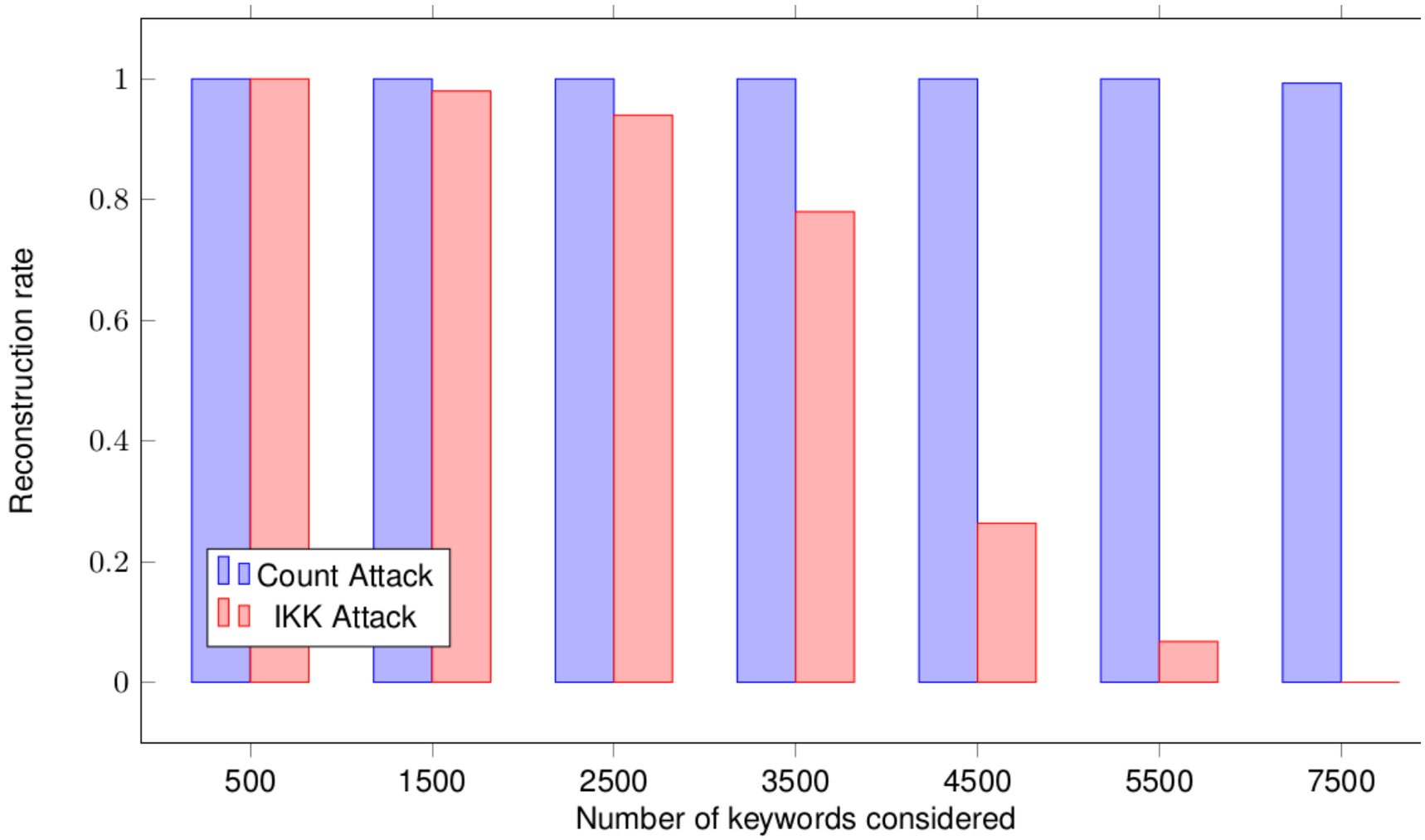
	rec1	rec2	rec3	rec4
Q1	1			
Q2		1		
Q3		1	1	1
Q4	1		1	
Q5	1			1
Q6	1			

Q3 matched 3 records, so it must be "Lewis"

Q2 overlapped w/ one record containing "Lewis" so it must be "Francis"

Query Recovery Experiment

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



Runs in seconds, not hours; scales up to larger dictionaries

Conclusion 2

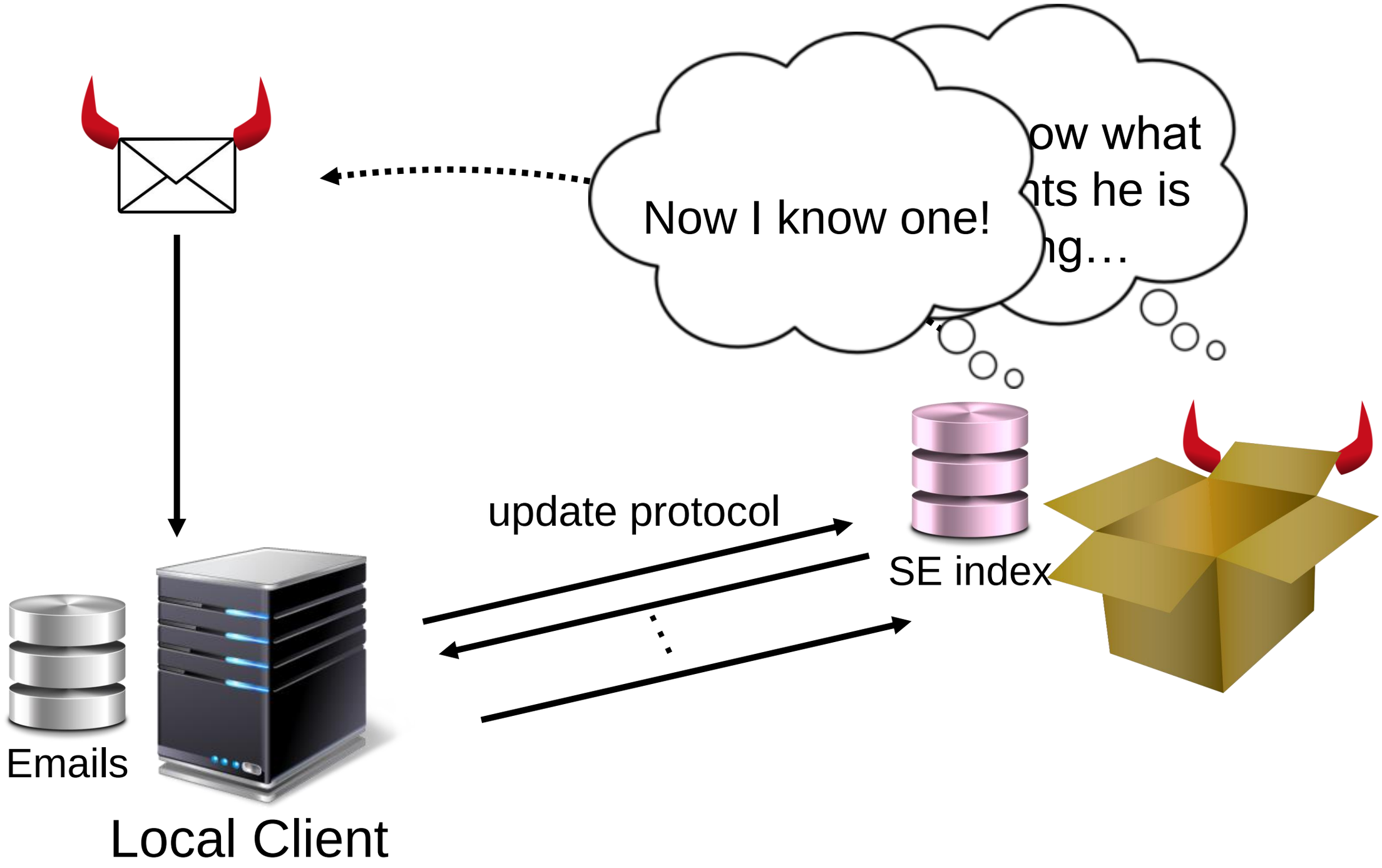
Even the stronger searchable encryption schemes are not safe for hiding queries on known datasets.

This attack gives no definitive answer for other cases.

One more attack trick

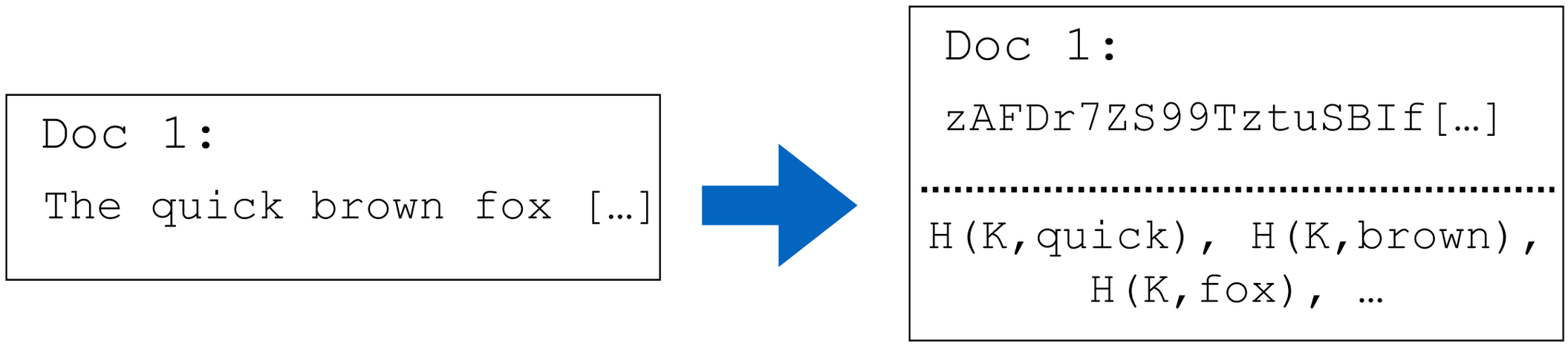
Here is an even nastier thing the server can do.

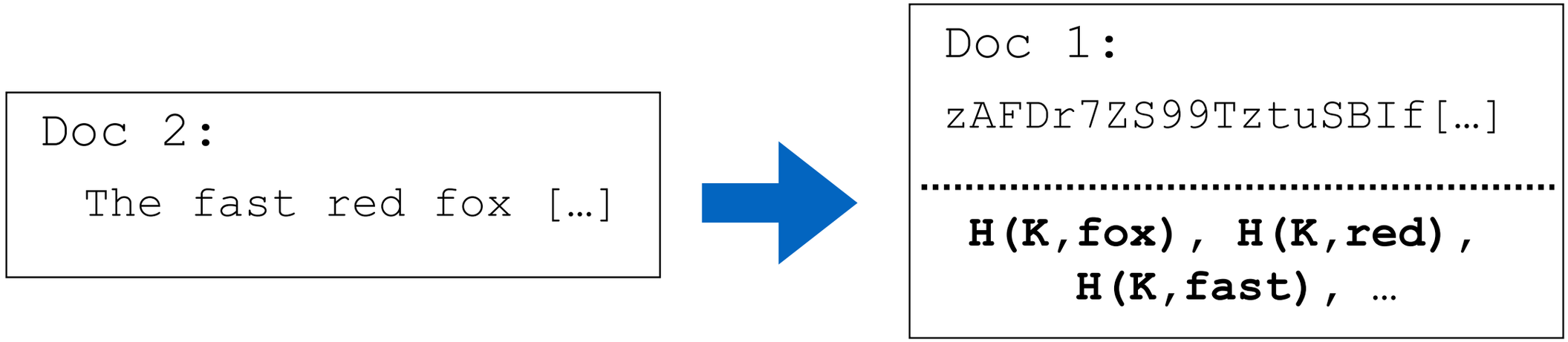
Attack 3: Chosen-Document Insertion



Chosen-Document Attack ⇒ Learn chosen hashes

- Again we attack the appended-keyword-hash constructions





- Server immediately learns keyword hashes of planted doc
- Hashes stored in document order ⇒ very easy attack
- Hashes not in order ⇒ more difficult (our results)

Chosen Document Attack Experiment

Goal: Maximize number of keywords learned from a minimum number of chosen documents (planted 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 keywords with <20% error rate

Potentially more secure constructions

Newer work: **Sophos - Forward Secure Searchable Encryption** [Bost 2016]

https://eprint.iacr.org/2016/728

Claim “optimal point of the security/performance tradeoff for SSE”

Conclusion

Encrypting data securely and the capability to search it seem to be opposing goals.

Current constructions for searchable encryption provide possibilities for negotiating an efficient compromise.

However, claims of security products must be sifted to determine what these compromises are.

We’ve only scratched the surface…

Thank you!