Security of Searchable Encrypted Cloud Storage

David Cash
Rutgers U

Paul Grubbs
Skyhigh Networks

Jason Perry
Lewis U

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 indexing techniques
End-to-end encryption breaks searching

- Searching incompatible with privacy goals of traditional encryption
Searchable Encryption Research

Usability
• What query types are supported?
• Legacy compatible?

Efficiency
• Space/computation used by server and client

Security
• Minimizing what a dishonest server can learn

This Talk:
• only treating single-keyword queries
• only examining highly efficient constructions
• focus on understanding security

Not treated: More theoretical, highly secure solutions (FHE, MPC, ORAM, …)
Searchable Symmetric Encryption [SWP’00, CGKO’06, …]

Want docs containing word \( w = \text{“simons”} \)

Should not learn docs or queries

Search token: \( T_w \)

C1, C2, C3, …

nCeUKlK7GO5ew6mwpIraODusbskYvBj9GX0F0bNvpuxtwXKuEdbHVuYAd4mEULgyJmzHV03ar8RDPUE16TfEqihoa8WzcEo18U8bQ1BzLK368qufbMMH1GvNsOVqt2xtfZhDUpDig8I0jyWyuOedYOvYq6XPqZc25tDHNCLv2DFJdcD9o4FD
Other SE types deployed (and sold)

Typically lower security than SSE literature solutions, as we will see.
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” \( L \)
- allows server to learn info corresponding to \( L \), but no more

Example \( L \) outputs:

- Size info of records and newly added records
- Query repetition
- Access pattern: Repeated record IDs across searches
- Update information: Some schemes leak when two added records contain the same keyword
What does $L$-secure mean in practice?

Messy question which depends on:

- **The documents**: number, size, type/content
- **The queries**: number, distribution, type/content
- **Data processing**: Stemming, stop word removal, etc
- **The updates**: frequency, size, type
- **Adversary’s knowledge**: of documents and/or queries
- **Adversary’s goal**: What exactly is it trying to do?

Currently almost no guidance in the literature.
Attacking SE: An example

- Consider an encrypted inverted index
- Keywords/data not in the clear, but pattern of access of document IDs is 

<table>
<thead>
<tr>
<th>keyword</th>
<th>records</th>
</tr>
</thead>
<tbody>
<tr>
<td>45e8a</td>
<td>4, 9, 37</td>
</tr>
<tr>
<td>092ff</td>
<td>9, 37, 93, 94, 95</td>
</tr>
<tr>
<td>f61b5</td>
<td>9, 37, 89, 90</td>
</tr>
<tr>
<td>cc562</td>
<td>4, 37, 62, 75</td>
</tr>
</tbody>
</table>

“this keyword is the most common”

“record #37 contains every keyword, and overlaps with record #9 a lot”

- Highly unclear if/when leakage is dangerous
One prior work: Learning queries

**Bad news:** Under certain circumstances, queries can be learned at a high rate (80%) by a curious server who knows all of the records that were encrypted. [Islam-Kuzu-Kantarcioglu]

(sketched later)
This work: Practical Exploitability of SE Leakage

- Many-faceted expansion of [Islam-Kuzu-Kantarcioglu]:
  1. **Different adversary goals**: Document (record) recovery in addition to query recovery
  2. **Different adversary knowledge**: (full, partial, and distributional)
  3. **Active adversaries**: planted documents

- Simple attacks **exploiting only leakage** for query recovery, document recovery, with experiments

- Note: For simplicity, this talk presents attacks on specific implementations.
Datasets for Attack Experiments

Enron Emails

• 30109 Documents from employee sent_mail folders (to focus on intra-company email)
• When considering 5000 keywords, average of 93 keywords/doc.

Apache Emails

• 50582 documents from Lucene project’s java-user mailing list
• With 5000 keywords, average of 291 keywords/doc

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

1. Simpler query recovery

2. Document recovery from partial knowledge

3. Document recovery via active attack
Query recovery using document knowledge

[Islam-Kuzu-Kantarcioglu]

Attack setting:

• Server knows all documents e.g., public financial data
• k random queries issued
• Minimal leakage: Only which records match each query (as SSE)
• Target: Learn the queries

Inverted index (known):

<table>
<thead>
<tr>
<th>keyword</th>
<th>records</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunnyvale</td>
<td>4,37,62,75</td>
</tr>
<tr>
<td>rutgers</td>
<td>9,37,93,94,95</td>
</tr>
<tr>
<td>admissions</td>
<td>4, 9,37</td>
</tr>
<tr>
<td>committee</td>
<td>8,37,89,90</td>
</tr>
</tbody>
</table>

Leakage (unknown queries):

<table>
<thead>
<tr>
<th></th>
<th>rec1</th>
<th>rec2</th>
<th>rec3</th>
<th>rec4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Q4</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Q6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Leakage (unknown queries):

<table>
<thead>
<tr>
<th></th>
<th>rec1</th>
<th>rec2</th>
<th>rec3</th>
<th>rec4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Q4</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Q6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Observes how often each query intersects with other queries
• Uses knowledge of document set to create large optimization problem for finding mapping from queries to keywords
• Solving NP-hard problem, severely limited to small numbers of queries, certain distributions
Observation

The IKK attack requires the server to have virtually perfect knowledge of the document set

If so, then why not just look at the number of documents returned by a query?

When a query term returns a unique number of documents, then it can immediately be guessed
Query Recovery via Counts

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

<table>
<thead>
<tr>
<th></th>
<th>rec1</th>
<th>rec2</th>
<th>rec3</th>
<th>rec4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Q4</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Q5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Leakage:

Q3 matched 3 records, so it must be “rutgers”.

Q2 overlapped with one record containing “rutgers” so it must be “sunnyvale”.
Query Recovery Experiment

Setup:
- Enron email subset
- $k$ most frequent words
- 10% queried at random

- Nearly 100% recovery, scales to large number of keywords, runs in seconds
Query Recovery with Partial Knowledge

• What if document set is only partially known?

• We generalized counting attack to account for imperfect knowledge

• Tested count and IKK attacks when only x% of the document was revealed
Query Recovery with Partial Knowledge

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

1. Simpler query recovery

2. Document recovery from partial knowledge

3. Document recovery via active attack
Document Recovery using Partial Knowledge

This blob indexes some docs I happen to know and others I don’t… What does that tell me?
Passive Document Recovery Attack Setting

- Server knows type of documents (i.e. has training set)
- No queries issued at all
- Some documents become “known”
- **Target:** Recover other document contents
Leakage that we attack

- Stronger SE schemes are immune to document recovery until queries are issued
- So we attack weaker constructions of the form:

<table>
<thead>
<tr>
<th>Record 1:</th>
<th>Record 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>The quick brown fox [...]</td>
<td>The fast red fox [...]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record 1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>zAFDr7ZS99TztuSBIf[...]</td>
</tr>
<tr>
<td>H(K, quick), H(K, brown), H(K, fox), ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hs9gh4vz0GmH32cXK5[...]</td>
</tr>
<tr>
<td>H(K, fast), H(K, red), H(K, fox), ...</td>
</tr>
</tbody>
</table>

Example systems:
- Mimesis [Lau et al’14]
- Shadowcrypt [He et al’14]
- Also: an extremely simple scheme
Simple Observation

If server knows Doc 1, then learns when any word in Doc 1 appears in other docs

Implementation detail: We assume hash values stored in order.

Harder but still possible if hash in random order. (see paper)
Document Recovery with Partial Knowledge

- For each dataset, we ran attack knowing either 2 or 20 random emails
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 gives us 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

Leakage from my crafted email!

Local Proxy

update protocol

SE index
Chosen-Document Attack ⇒ Learn chosen hashes

- Again we attack weaker constructions of the form:

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

H(K,quick), H(K,brown), H(K,fox), ...

New Doc:
contract sell buy

H(K,contract), H(K,buy), H(K,sell), ...

- Hashes in order ⇒ very easy attack
- Hashes not in order ⇒ more difficult (we attack now)
Chosen Document Attack Experiment

- **Goal**: Maximize fraction of keywords learned from a minimum number of chosen documents emails

- **Procedure for generating chosen emails**:
  1. Divided dataset into half training / half test
  2. Based on training set, rank keywords by frequency
  3. Generate chosen emails with k keywords each
  4. Learn unordered hash values of those k keywords
  5. Guess hash → keyword mapping via frequency counts

- Ran with two different training setups:
  1. Training and test sets from same corpus (both Enron or Apache)
  2. Training and test from different corpora (i.e. train on Apache, test on Enron)
Chosen Document Attack Experiment Results

![Graph showing the fraction of keywords against keywords per chosen document. The graph includes lines for recovery rate, error rate, recovery unrelated, and error unrelated.](image-url)
Systematic study of exploitability of multiple SE leakage types reveals serious vulnerabilities.

- Temptation to deploy ad-hoc solutions must be avoided
- If a security proof includes leakage, also need (at least) an empirical characterization of what one can do with the leakage
Future Work and Open Problems

• Many similar directions left to explore

• Relate experiments to real threats, recommend quantitatively how to use SE

• On-going work: Attacks using automatic human language translation against word substitution
Thanks!