Searchable Symmetric Encryption: Optimal Locality in Linear Space via Two-Dimensional Balanced Allocations

Gilad Asharov
Moni Naor
Gil Segev
Ido Shahaf

IBM Research
IBM Research
Weizmann
Hebrew University
Hebrew University

STOC 2016
Cloud Storage

• We are outsourcing more and more of our data to clouds

• We trust these clouds less and less
  • Confidentially of the data from the service provider itself
  • Protect the data from service provider security breaches
Solution: Encrypt your Data!

• But…
  • **Keyword search** is now the primary way we access our data
  • By encrypting the data - this simple operation becomes **extremely expensive**

• How to search on encrypted data??
Possible Solutions

• **Generic tools:** Expensive, great security
  • Functional encryption
  • Fully Homomorphic Encryption
  • Oblivious RAM*

• **More tailored solutions:** practical, security(?)
  • Property-preserving encryption (encryption schemes that supports public tests)
    • Deterministic encryption  [Bellare-Boldyreva-O’Neill06]
    • Oder-preserving encryption  [Agrawal-Kiernan-Srikant-Xu04]
    • Orthogonality preserving encryption  [Pandey-Rouselakis04]
  • Searchable Symmetric Encryption  [Song-Wagner-Perrig01]
Searchable Symmetric Encryption (SSE)
Searchable Symmetric Encryption (SSE)

- **Data:** the database DB consists of:
  - **Keywords:** W={w₁,…,wₙ} (possible keywords)
  - **Documents:** D₁,…,Dₘ (list of documents)
  - DB(wᵢ)={id₁,…,idₙᵢ} (for every keyword wᵢ, list of documents / identifiers in which wᵢ appears)

- **Syntax of SSE:**
  - K ← KeyGen(1ᵏ) (generation of a private key)
  - EDB ← EDBSetup(K,DB) (encrypting the database)
  - (DB(wᵢ),λ) ← Search((K,wᵢ),EDB) (interactive protocol)
### EDBSetup

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searchable</td>
<td>5,14</td>
</tr>
<tr>
<td>Symmetric</td>
<td>5,14,22,45,67</td>
</tr>
<tr>
<td>Encryption</td>
<td>1,2,3,4,5,6,7,8,9,10</td>
</tr>
<tr>
<td>Schemes</td>
<td>22,14</td>
</tr>
</tbody>
</table>

Replace each keyword $w$ with some $\text{PRF}_K(w)$

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>05de23ng</td>
<td>5,14</td>
</tr>
<tr>
<td>91mdik289</td>
<td>5,14,22,45,67</td>
</tr>
<tr>
<td>91sjwimg</td>
<td>1,2,3,4,5,6,7,8,9,10</td>
</tr>
<tr>
<td>oswspl25ma</td>
<td>22,14</td>
</tr>
</tbody>
</table>

**inverted index**

**encrypted index**

---

**Key points:***
- EDBSetup introduces a method for handling keywords in a searchable encrypted data base.
- It involves replacing each keyword with a function $\text{PRF}_K(w)$.
- The table lists keywords and their associated records, with an emphasis on searchable and symmetric encryption schemes.
- The inverted index illustrates the mapping of keywords to records, while the encrypted index shows the encrypted keyword records.
The Challenge…

No leakage on the structure of the lists!

How to map the lists into memory?

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>05de23ng</td>
<td></td>
</tr>
<tr>
<td>91mdik289</td>
<td></td>
</tr>
<tr>
<td>91sjwimg</td>
<td></td>
</tr>
<tr>
<td>oswspl25ma</td>
<td></td>
</tr>
</tbody>
</table>
Functionality - Search
(Allow some Leakage...)

Security Requirement:
The server should not learn anything about the structure of lists that were not queried
Security

• **Good news:** Semantic security for data; no deterministic or order preserving encryption

• But.. for reasonable performance -> leakage for server

• Leakage in the form of access patterns to retrieved data and queries
  • Data is encrypted but server can see intersections b/w query results
    (e.g. identify popular document)

• Additional specific leakage:
  • E.g. we leak |DB(w1)|
  • E.g. the server learns if two documents have the same keyword

• Leads to statistical inference based on side information on data (effect depends on application)
Mapping Lists into Memory

Maybe shuffle the lists?

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>05de23ng</td>
<td></td>
</tr>
<tr>
<td>91mdik289</td>
<td></td>
</tr>
<tr>
<td>91sjwimg</td>
<td></td>
</tr>
<tr>
<td>oswspl25ma</td>
<td></td>
</tr>
</tbody>
</table>
Hiding the Structure of the Lists

Maybe shuffle the lists?
Previous Constructions: Maximal Padding [CK10]

1) Pad each list to maximal size (N?)
2) Store lists in random order
3) Pad with extra lists to hide the number of lists

Size of encrypted DB: \(O(N^2)\)
Previous Constructions
Linked List \cite{CGK+06}
Efficiency Measures

• A variant was implemented in [CJJ+13]
  • Poor performance due to… locality!

• **Space**: The overall size of the encrypted database (Want: O(N))

• **Locality**: number of non-continuous memory locations the server accesses with each query (Want: O(1))

• **Read efficiency**: The ratio between the number of bits the server reads with each query, and the actual size of the answer (Want: O(1))
SSE and Locality [CT14]

Can we construct an SSE scheme that is optimal in **space**, **locality** and **read efficiency**?

**NO!**

- **Lower bound:** any scheme must be sub-optimal in either its **space** overhead, **locality** or **read efficiency**
- Impossible to construct scheme with $O(N)$ **space**, $O(1)$ **locality** and $O(1)$ **read efficiency**

**Our Question:**
can we construct a scheme that is nearly optimal?
Related Work

- A single keyword search
  - Related work [SWP00,Goh03,CGKO06,ChaKam10]

- Beyond single keyword search
  - Conjunctions, range queries, general boolean expression, wildcards [CJJJKRS13, JJKRS13, CJJJJKRS14, FJKNRS15]
  - Schemes that are not based on inverted index [PKVKMCGKB14, FVKKKMB15]

- **Locality** in searchable symmetric encryption [CT14]

- Dynamic searchable symmetric encryption [....]
### Our Results

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Space</th>
<th>Locality</th>
<th>Read Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>[CGK+06, KPR12, CJJ+13]</td>
<td>O(N)</td>
<td>O(n&lt;sub&gt;W&lt;/sub&gt;)</td>
<td>O(1)</td>
</tr>
<tr>
<td>[CK10]</td>
<td>O(N&lt;sup&gt;2&lt;/sup&gt;)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>[CT14]</td>
<td>O(NlogN)</td>
<td>O(logN)</td>
<td>O(1)</td>
</tr>
<tr>
<td>This work I</td>
<td>O(N)</td>
<td>O(1)</td>
<td>Õ(logN)</td>
</tr>
<tr>
<td>This work II*</td>
<td>O(N)</td>
<td>O(1)</td>
<td>Õ(loglogN)</td>
</tr>
<tr>
<td>This work III</td>
<td>O(NlogN)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

<sup>Õ(f(N))=O(f(n) log f(n))</sup>

*assumes no keyword appears in more than N<sup>1-1/loglogN</sup> documents
Our Approach

• We put forward a two-dimensional generalization of the classic balanced allocation problem (“balls and bins”), considering lists of various lengths instead of “balls” (=lists of fixed length)

(1) We construct efficient 2D balanced allocation schemes

(2) Then, we use cryptographic techniques to transform any such scheme into an SSE scheme
Balls and Bins

\( \bullet \times n \)

\( m \)

?
Balls and Bins
(Random Allocation)

- $n$ balls, $m$ bins
  - Choose for each ball one bin uniformly at random
  - $m = n$: with high probability - there is no bin with more than $\frac{\log n}{\log \log n} \cdot (1 + o(1))$
  - $m = n/\log n$: with overwhelming probability, there is no bin with load greater than $\tilde{O}(\log n)$
Two-Dimensional Allocation
Two-Dimensional Allocation
Two-Dimensional Allocation

Place the whole list according to a \textit{single} probabilistic choice!
Two-Dimensional Allocation
Two-Dimensional Allocation
Two-Dimensional Allocation
Two-Dimensional Allocation
Two-Dimensional Allocation
Two-Dimensional Allocation
Two-Dimensional Allocation
Two-Dimensional Allocation

What is the maximal load?
How Do We Search?

Search( )
Our First Scheme: 2D Random Allocation

- **Main Challenge** (compared to 1D case): Heavy dependencies between the elements of the same list

- **This yields an SSE scheme with:**
  - Space: \( \#\text{Bins} \times \text{BinSize} = O(N) \)
  - Locality: \( O(1) \)
  - Read efficiency: \( \tilde{O}(\log n) \)

**Theorem:** Set \( \#\text{Bins} = N/O(\log N \log \log N) \). Then, with an overwhelming probability, the maximal load is \( 3\log N \log \log N \).
The Power of Two Choices

• In the classic “balls and bins” [ABKU99]:
  • If we choose one random bin for each ball, then the maximal load is $O(\log N/ \log\log N)$
  • If we choose two random bins for each ball, and place the ball in the least loaded one, then the maximal load is $O(\log\log N)$
  • Exponential improvement!

• Can we adapt the two-choice paradigm to the 2D case?
2D Two-Choice Allocation
2D Two-Choice Allocation
2D Two-Choice Allocation
2D Two-Choice Allocation
2D Two-Choice Allocation

**Theorem:** Assume all lists are of length at most $N^{1-1/\log\log N}$, and set $\#\text{Bins}=N/(\log\log N (\log\log\log\log N)^2)$. Then, with an overwhelming probability, the maximal load is $O(\log\log N (\log\log\log\log N)^2)$.

- **Main Challenge:** (compared to 1D case):
  - Manny challenges...

- This yields an SSE scheme with:
  - Space: $\#\text{Bins} \times \text{BinSize} = O(N)$
  - Read efficiency: $2\text{BinSize} = \tilde{O}(\log\log N)$
  - Locality: $\tilde{O}(1)$
Summary

- **Our approach**: SSE via two-dimensional balanced allocations

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Space</th>
<th>Locality</th>
<th>Read Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work I</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$\tilde{O}(\log N)$</td>
</tr>
<tr>
<td>This work II*</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$\tilde{O}(\log \log N)$</td>
</tr>
<tr>
<td>This work III</td>
<td>$O(N \log N)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Nice combination between DS and Cryptography

Thank You!