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

Gilad Asharov
Moni Naor
Gil Segev
Ido Shahaf

Cornell-Tech
(Hebrew University)
Weizmann
Hebrew University
Hebrew University
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]
    - Order-preserving encryption [Agrawal-Kiernan-Srikant-Xu04]
    - Orthogonality preserving encryption [Pandey-Rouselakis04]
  - Searchable Symmetric Encryption [Song-Wagner-Perrig01]
### Deterministic and Order Preserving Encryptions

#### “Inference Attacks against Property-Preserving Encrypted Databases”
[Naveed-Kamara-Wright. CCS2015]

<table>
<thead>
<tr>
<th>Name</th>
<th>Lastname</th>
<th>Age</th>
<th>Name</th>
<th>Lastname</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elaine</td>
<td>Samuels</td>
<td>24</td>
<td>Ge5$#u</td>
<td>Q*6sh#</td>
<td>223</td>
</tr>
<tr>
<td>Mary</td>
<td>Stein</td>
<td>37</td>
<td>E89(%y)</td>
<td>2@#3Br</td>
<td>340</td>
</tr>
<tr>
<td>Jim</td>
<td>Stein</td>
<td>81</td>
<td>2Tr^#7</td>
<td>2@#3Br</td>
<td>736</td>
</tr>
<tr>
<td>John</td>
<td>Sommers</td>
<td>3</td>
<td>qM@9*h</td>
<td>gYv6%t</td>
<td>34</td>
</tr>
<tr>
<td>Mary</td>
<td>Williams</td>
<td>17</td>
<td>E89(%y)</td>
<td>X%3oL7</td>
<td>160</td>
</tr>
<tr>
<td>John</td>
<td>Garcia</td>
<td>43</td>
<td>qM@9*h</td>
<td>wnM7#1</td>
<td>308</td>
</tr>
<tr>
<td>John</td>
<td>Gould</td>
<td>37</td>
<td>qM@9*h</td>
<td>8vy8$Z</td>
<td>340</td>
</tr>
</tbody>
</table>
Searchable Symmetric Encryption (SSE)
Searchable Symmetric Encryption (SSE)

- **Data:** the database $DB$ consists of:
  - **Keywords:** $W = \{w_1,\ldots,w_n\}$ (possible keywords)
  - **Documents:** $D_1,\ldots,D_m$ (list of documents)
  - $DB(w_i) = \{id_1,\ldots,id_{n_i}\}$
    (for every keyword $w_i$, list of documents / identifiers in which $w_i$ appears)

- **Syntax of SSE:**
  - $K \leftarrow \text{KeyGen}(1^k)$ (generation of a private key)
  - $EDB \leftarrow EDB\text{Setup}(K, DB)$ (encrypting the database)
  - $(DB(w_i),\lambda) \leftarrow \text{Search}((K,w_i), EDB)$ (interactive protocol)
The Searching Protocol

• \((DB(w), \lambda) \leftarrow Search((K,w), EDB)\) (interactive protocol)

• Usually - one round protocol

\((K,w)\)

\((\tau, \rho) \leftarrow TokGen(K,w)\)

\(DB(w) \leftarrow Resolve(\rho, M)\)

\(M \leftarrow Search(EDB, \tau)\)
Security Requirement

• Two equivalent definitions:
  • Game-based definition
  • Simulation-based definition
Game-Based Definition

- The adversary controls the “cloud”

- Outputs two databases $DB_0, DB_1$ with intersection on $w$ (of the same size, that share some lists $\{DB(w)\}_{w \in w}$ for some set of keywords $w$)

- The client receives $DB_b$ for some randomly chosen $b$

- Runs: $K \leftarrow KeyGen(1^k)$, $EDB \leftarrow EDBSetup(K, DB)$ and $\tau_i = TokGen(k, w)$ for all $w \in w$

- The adversary receives: $(EDB, \{\tau_w\}_{w \in w})$, guesses $b$
Game-Based Definition

**DB₀**

**DB₁**
Game-Based Definition

Need to hide the “structure” of the lists

DB₀

DB₁
Simulation Based Security

• The adversary outputs (DB, w)

  • **REAL world:**
    • The experiment runs *KeyGen, EDBSetup, and TokGen* for every $w \in \mathbb{w}$
    • $EDB$ (the resulting encrypted DB), $\{\tau_w\}_{w \in \mathbb{w}}$ (the resulting tokens)

  • **IDEAL world:**
    • The simulator receives $\mathcal{L}(DB, w)$
      (some leakage on the *queried keywords only*)
    • Outputs $EDB$ (the resulting encrypted DB), $\{\tau_w\}_{w \in \mathbb{w}}$ (the resulting tokens)
    • The adversary receives $EDB, \{\tau_w\}_{w \in \mathbb{w}}$, output REAL/IDEAL
Security

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

• 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)
### 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>

**inverted index**

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>

**encrypted index**
The Challenge…

No leakage on the structure of the lists!

How to map the lists into memory?
Functionality - Search
(Allow some Leakage...)

Search for keyword: Encryption

(X, w)

Security Requirement:
The server should not learn anything about the structure of lists that were not queried
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 [CGK+06]
Efficiency Measures [CT14]

- 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: $O(1)$)
Efficiency

• Scheme I:
  • **Space**: \( O(N) \)
  • **Locality**: \( O(N) \)
  • **Read efficiency**: \( O(1) \)

• Scheme II:
  • **Space**: \( O(N^2) \)
  • **Locality**: \( O(1) \)
  • **Read efficiency**: \( O(1) \)
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*
Why NO*?

Theorem 1.1 If $\Pi$ is an $\mathcal{L}$-IND-secure SSE scheme with locality $r$ as well as $\alpha$-overlapping reads, then $\Pi$ has $\omega \left( \frac{|\text{BinEnc(DB)}|}{r \cdot (\alpha + 1)} \right)$ server storage.

- Instead of **read efficiency** the theorem captures **“$\alpha$-overlapping reads”**
- Intuitively, any two reads intersect in at most $\alpha$ bits
  - Captures all previous constructions
  - Large $\alpha$ - “waste”

**Intuition for lower bound:**
- Reads do not intersect much (**$\alpha$-overlapping reads**)
- Any list can be placed only in few positions (locality)
- We must pad the lists in order to hide their sizes…
Our Goal:
Constructing a scheme that is nearly optimal?

• Maybe even completely optimal if we do not assume α-overlapping reads? (though, it seems counter-intuitive)
• How do schemes with “large” α look like?
Related Work

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

- Beyond single keyword search
  - Conjunctions, range queries, general boolean expression, wildcards [CashJJJKRS13, JareckiJKRS13, CashJJJKRS14, FaberJKNRS15]
  - Schemes that are not based on inverted index [PappasKVKKMGB14, FischVKKKMB15]

- **Locality** in searchable symmetric encryption [CashTessaro14]

- Dynamic searchable symmetric encryption [……]

- Leakage-abuse attacks [CashGrubbsPerryRistenpart15]
## 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>$O(n_w)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>[CK10]</td>
<td>$O(N^2)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>[CT14]</td>
<td>$O(N\log N)$</td>
<td>$O(\log N)$</td>
<td>$O(1)$</td>
</tr>
<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>

$\tilde{O}(f(N)) = O(f(n) \log f(n))$

*assumes no keyword appears in more than $N^{1-1/\log \log N}$ documents
## Our Schemes

1) Choose for each list “possible ranges” independently

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>05de23ng</td>
<td><img src="image" alt="Blue Range" /></td>
</tr>
<tr>
<td>91mdik289</td>
<td><img src="image" alt="Green Range" /></td>
</tr>
<tr>
<td>91sjwimg</td>
<td><img src="image" alt="Red Range" /></td>
</tr>
<tr>
<td>oswspl25ma</td>
<td><img src="image" alt="Purple Range" /></td>
</tr>
</tbody>
</table>

2) Place the elements of each list in its possible ranges

(is it possible?)
Allocation Algorithms

- We show a general transformation:
  - Allocation algorithm $\Rightarrow$ secure SSE scheme
  - If the allocation algorithm is "efficient" then the SSE is `efficient` (successfully places the lists even though each has few possible "small" possible ranges)

- **Security intuition**: The *possible* locations of each list are completely independent to the *possible* locations of the other lists
  - (But many correlations in the *actual* placement)

- With each query, the server reads *all* possible ranges of the list
  - We never reveal the decisions made for the actual placement

- How to construct efficient Allocation algorithms?
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

$\bullet \times n$

$\text{?}$

$m$
Balls and Bins
(Random Allocation)

• n balls, m bins
  • Choose for each ball one bin uniformly at random
  • \textbf{m=n:} with high probability - there is no bin with
    more than \( \frac{\log n}{\log \log n} \cdot (1 + o(1)) \)

• \textbf{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 **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

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

- **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: $\tilde{O}(\log n)$
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

$w_1$  $w_2$  $w_3$  $w_4$  $w_5$  $w_6$
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: $O(1)$
On the Assumption

• We assume that no keyword appears in more than $n^{1-1/\log\log n}$ documents
  • Keywords with too many occurrences are not indexed by search engines

• **Tightness:**
  • Assume that there are $n^{1/\log\log n}$ lists of size $n^{1-1/\log\log n}$
  • The probability that they all share the same super-bin is noticeable
    • Cannot be placed even using more sophisticated algorithms
  • We generalize this intuition to capture **all** allocation algorithms
Summary

• Novel generalization of classical data structure problem
  • And use it to build a crypto system!
  • The construction seems practical (small constants)

• First constructions of SSE with no bound on the overlapping reads
  • First constructions with \textit{linear} encrypted database size and “good” locality
  • Still, we see limitations of allocation problems (On the size of the maximal list)

• Extending [CT14] lower bound?
## 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!