Dynamic Searchable Encryption with Small Client Storage

Ioannis Demertzis  
University of Maryland  
yannis@umd.edu

Javad Ghareh Chamani  
HKUST and Sharif University of Technology  
jgc@cse.ust.hk

Dimitrios Papadopoulos  
HKUST  
dipapado@cse.ust.hk

Charalampos Papamathou  
University of Maryland  
cpap@umd.edu
What is Dynamic Searchable Encryption (DSE)?

**Client**

- **\( L_{\text{Query}} \) leakage** --- Search pattern: whether a search query is repeated

**Untrusted Cloud**

- **\( L_{\text{Setup}} \) leakage**: total leakage prior to query execution e.g. size of each encrypted file, size of the encrypted index
- **\( L_{\text{Update}} \) leakage**: leakage during update execution

---

**Search query:**

- **keyword**

---

**Access pattern**: encrypted document ids and files that satisfy the search query

---

**Update query:**

- **keyword**

---

**Security (informal)**: The adversary does not learn anything beyond the above leakages!
High-level idea: The server should not be able to relate an update with a previous operation!

- Time
  - 1: Add (F1, w1)
  - 2: Query (w1)
  - 3: Add (F2, w1)

- Server should not learn that update in timestamp 3 is for the same keyword!

- **Definition:** A DSE scheme is forward private if the update does not reveal any information about the involved keyword, i.e., $\mathcal{L}_{\text{Update}}(w) = \mathcal{L}(\text{op}, \text{id})$
Update Leakage --- Backward Privacy

**High-level idea:** The server should reveal controlled information about deleted files during search.

**Time**

1. Add (F1, \(w_1\))
2. Add (F2, \(w_1\))
3. Del (F1, \(w_1\))
4. Query (\(w_1\))

\[\text{TimeDB}(w_1) = \{(F2, 2)\}\]

\[\text{Updates}(w_1) = \{1, 2, 3\}\]

\[\text{DelHist}(w_1) = \{(1, 3)\}\]
Update Leakage --- Backward Privacy

**High-level idea:** The server should reveal controlled information about deleted files during search.

<table>
<thead>
<tr>
<th>Time</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Add (F1, w1)</td>
</tr>
<tr>
<td>2</td>
<td>Add (F2, w1)</td>
</tr>
<tr>
<td>3</td>
<td>Del (F1, w1)</td>
</tr>
<tr>
<td>4</td>
<td>Query (w1)</td>
</tr>
</tbody>
</table>

Backward Privacy Type – I: \( L^{\text{Search}}(w) = L(\text{TimeDB}(w)) \)

Backward Privacy Type – II: \( L^{\text{Search}}(w) = L(\text{TimeDB}(w), \text{Updates}(w)) \)

Backward Privacy Type – III: \( L^{\text{Search}}(w) = L(\text{TimeDB}(w), \text{Updates}(w), \text{DelHist}(w)) \)
Issues with Prior Forward & Backward DSE schemes

**Require:** The client to store an operation counter for each unique keyword!!!

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>w1</td>
<td>3</td>
</tr>
<tr>
<td>w2</td>
<td>2</td>
</tr>
<tr>
<td>w3</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>wM</td>
<td>5</td>
</tr>
</tbody>
</table>

$O(|W|)$ can be up to $O(N)$, where $N$ is the total DB size

$O(|W|)$, where $|W|$ is the dictionary size

Synchronization issues: if the client wants to access the encrypted DB from multiple devices
Issues with Prior Forward & Backward DSE schemes

**Outsourcing** the operational counters to the server requires the use of Oblivious Indexes ~ extra $O(\log^2 N)$ overhead and $O(\log N)$ rounds of interaction!

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>w1</td>
<td>3</td>
</tr>
<tr>
<td>w2</td>
<td>2</td>
</tr>
<tr>
<td>w3</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>wM</td>
<td>5</td>
</tr>
</tbody>
</table>

E.g., an Oblivious Hash Map (OMAP)

Untrusted Cloud

Keywords

- w1
- w2
- w3
- wM
### Prior state-of-the-art Works & Our Contributions

<table>
<thead>
<tr>
<th>BP-Type</th>
<th>Search RT</th>
<th>BP-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type-I</td>
<td>2</td>
<td>Type-I</td>
</tr>
<tr>
<td>Type-II</td>
<td></td>
<td>Type-II</td>
</tr>
<tr>
<td>Type-II</td>
<td></td>
<td>Type-II</td>
</tr>
<tr>
<td>Type-II</td>
<td></td>
<td>Type-II</td>
</tr>
<tr>
<td>Type-I</td>
<td>2</td>
<td>Type-I</td>
</tr>
<tr>
<td>Type-III</td>
<td></td>
<td>Type-III</td>
</tr>
<tr>
<td>Type-III</td>
<td></td>
<td>Type-III</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Search</th>
<th>Update</th>
<th>Search RT</th>
<th>BP-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moneta</td>
<td>$\tilde{O}(a_w \log N + \log^3 N)$</td>
<td>$\tilde{O}(\log^2 N)$</td>
<td>2</td>
<td>Type-I</td>
</tr>
<tr>
<td>OMAP+Mitra</td>
<td>$O(a_w + \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td>Type-II</td>
</tr>
<tr>
<td>SDa</td>
<td>$O(a_w + \log N)$</td>
<td>$O(\log N)$ (am.)</td>
<td>1</td>
<td>Type-II</td>
</tr>
<tr>
<td>SDd</td>
<td>$O(a_w + \log N)$</td>
<td>$O(\log^3 N)$</td>
<td>1</td>
<td>Type-II</td>
</tr>
<tr>
<td>ORION</td>
<td>$O(n_w \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td>Type-I</td>
</tr>
<tr>
<td>HORUS</td>
<td>$O(n_w \log d_w \log N + \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td>Type-III</td>
</tr>
<tr>
<td>QOS</td>
<td>$O(n_w \log i_w + \log^2 N)$</td>
<td>$O(\log^3 N)$</td>
<td></td>
<td>Type-III</td>
</tr>
</tbody>
</table>

- $N$: total number of (file, keyword) pairs, $a_w$: #updates for keyword $w$
- $n_w$: #files currently containing keyword $w$, $i_w$: #inserts for keyword $w$
## Prior state-of-the-art Works & Our Contributions

<table>
<thead>
<tr>
<th></th>
<th>Search</th>
<th>Update</th>
<th>Search RT</th>
<th>BP-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moneta</td>
<td>$\tilde{O}(a_w \log N + \log^3 N)$</td>
<td>$\tilde{O}(\log^2 N)$</td>
<td>2</td>
<td>Type-I</td>
</tr>
<tr>
<td>OMAP+Mitra</td>
<td>$O(a_w + \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td>Type-II</td>
</tr>
<tr>
<td>SDa</td>
<td>$O(a_w + \log N)$</td>
<td>$O(\log N)$ (am.)</td>
<td>1</td>
<td>Type-II</td>
</tr>
<tr>
<td>SDd</td>
<td>$O(a_w + \log N)$</td>
<td>$O(\log^3 N)$</td>
<td>1</td>
<td>Type-II</td>
</tr>
<tr>
<td>ORION</td>
<td>$O(n_w \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td>Type-I</td>
</tr>
<tr>
<td>HORUS</td>
<td>$O(n_w \log d_w \log N + \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td>Type-III</td>
</tr>
<tr>
<td>QOS</td>
<td>$O(n_w \log i_w + \log^2 N)$</td>
<td>$O(\log^3 N)$</td>
<td></td>
<td>Type-III</td>
</tr>
</tbody>
</table>

SDa and SDd have **20x-85x** faster search time than MITRA

QOS has **14x-16531x** faster search time than HORUS
### Prior state-of-the-art Works & Our Contributions

<table>
<thead>
<tr>
<th></th>
<th>Search</th>
<th>Update</th>
<th>Search RT</th>
<th>BP-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Moneta</strong></td>
<td>$\tilde{O}(a_w \log N + \log^3 N)$</td>
<td>$\tilde{O}(\log^2 N)$</td>
<td>2</td>
<td>Type-I</td>
</tr>
<tr>
<td><strong>OMAP+Mitra</strong></td>
<td>$O(a_w + \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td>Type-II</td>
</tr>
<tr>
<td><strong>SDa</strong></td>
<td>$O(a_w + \log N)$</td>
<td>$O(\log N)$ (am.)</td>
<td>1</td>
<td>Type-II</td>
</tr>
<tr>
<td><strong>SDd</strong></td>
<td>$O(a_w + \log N)$</td>
<td>$O(\log^3 N)$</td>
<td>1</td>
<td>Type-II</td>
</tr>
<tr>
<td><strong>ORION</strong></td>
<td>$O(n_w \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td>Type-I</td>
</tr>
<tr>
<td><strong>HORUS</strong></td>
<td>$O(n_w \log d_w \log N + \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td>Type-III</td>
</tr>
<tr>
<td><strong>QOS</strong></td>
<td>$O(n_w \log i_w + \log^2 N)$</td>
<td>$O(\log^3 N)$</td>
<td></td>
<td>Type-III</td>
</tr>
</tbody>
</table>

**QOS** has better search time for deletion ratios between 40%-80%
SDa scheme

**High-level idea:** Organize \(N\) updates in a collection of at most \(\log_2 N\) independent encrypted indexes.
SDa scheme

**High-level idea:** Organize $N$ updates in a collection of at most $\log_2 N$ independent encrypted indexes

Add $(F4, w1)$
High-level idea: Organize $N$ updates in a collection of at most $\log_2 N$ independent encrypted indexes.

If we keep adding the number of encrypted indexes will be $O(N)$.
High-level idea: Organize $N$ updates in a collection of at most $\log_2 N$ independent encrypted indexes.

Whenever two indexes of the same size exist, download them and merge them in a new index.
SDa scheme

High-level idea: Organize $N$ updates in a collection of at most $\log_2 N$ independent encrypted indexes.

Update cost = $O(\log_2 N)$ (amortized)

*Assuming that $N$ is a power of 2
High-level idea: Organize $N$ updates in a collection of at most $\log_2 N$ independent encrypted indexes

Search cost = $O(\log_2 N + a_w)$

Intuition Forward/Backward privacy: Every index is built with a fresh key and the used static SE is response hiding!!!
SDa --- Amortized Update Cost

Cheap Updates $O(1)$

Expensive Updates $O(N)$
SDd scheme

High-level idea: Let’s de-amortize the SDa construction

\[-1, w1)\]

O(\(\log_2 N\))
encrypted indexes
SDd scheme

Client

Untrusted Cloud

OLDEST_1  OLDEST_2  OLDEST_4
OLDER_1    OLDER_2   OLDER_4
OLD_1      OLD_2    OLD_4
NEW_1      NEW_2    NEW_4

1  2  4
Add (F1, w1)

If NEW is full move it to the first empty OLDEST, OLDER or OLD index
SDd scheme

Client

Untrusted Cloud

Add \((F_2, w_1)\)
If $\text{OLDEST}_i$ and $\text{OLDER}_i$ are full start moving the updates to the $\text{NEW}_{i+1}$ index
For achieving **Forward Privacy** we need to use Oblivious MAPs (OMAP).
SDd scheme

Search cost = $O(3\log_2 N + a_w)$

OMAP cost = $O(\log^2 N)$

Update cost = $O(\log^3 N)$
### Prior state-of-the-art Works & Our Contributions

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Search</th>
<th>Update</th>
<th>Search RT</th>
<th>BP-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moneta</td>
<td>$\tilde{O}(a_w \log N + \log^3 N)$</td>
<td>$\tilde{O}(\log^2 N)$</td>
<td>2</td>
<td>Type-I</td>
</tr>
<tr>
<td>OMAP+Mitra</td>
<td>$O(a_w + \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td>$O(\log N)$</td>
<td>Type-II</td>
</tr>
<tr>
<td>SDa</td>
<td>$O(a_w + \log N)$</td>
<td>$O(\log N)$ (am.)</td>
<td>1</td>
<td>Type-II</td>
</tr>
<tr>
<td>SDd</td>
<td>$O(a_w + \log N)$</td>
<td>$O(\log^3 N)$</td>
<td>1</td>
<td>Type-II</td>
</tr>
<tr>
<td>ORION</td>
<td>$O(n_w \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td>$O(\log N)$</td>
<td>Type-I</td>
</tr>
<tr>
<td>HORUS</td>
<td>$O(n_w \log d_w \log N + \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td>$O(\log N)$</td>
<td>Type-III</td>
</tr>
<tr>
<td>QOS</td>
<td>$O(n_w \log i_w + \log^2 N)$</td>
<td>$O(\log^3 N)$</td>
<td>$O(\log N)$</td>
<td>Type-III</td>
</tr>
</tbody>
</table>

**Definition:** A DSE scheme has optimal (resp. quasi-optimal) search time, if the asymptotic complexity of Search is $O(n_w)$ (resp. $O(n_w \text{polylog}(N))$).
Motivation for Optimal/Quasi-optimal Search

Client

Add (F1, \( w_1 \))
Add (F2, \( w_1 \))
\vdots
Add (FN, \( w_1 \))
Del (F1, \( w_1 \))
\vdots
Del (FN-1, \( w_1 \))
Query(\( w_1 \))

Untrusted Cloud

SDd search cost = \( O(\log_2 N + a_{w}) \)

\( w_1 \) is contained only in File N, but the result has size \( O(a_w) \sim O(N) \)
**QOS --- Main Idea**

**Idea:** For each keyword $w$ create a data structure that helps us avoid the deleted regions

Add $(F1, w_1)$
Add $(F2, w_1)$
Add $(F5, w_1)$
Add $(F10, w_1)$
Add $(F35, w_1)$

Next empty leaf for $w_1$

Tree for keyword $w_1$ (N=8)
**QOS --- Main Idea**

**Idea:** For each keyword \( w \) create a data structure that helps us avoid the deleted regions

\[
\text{Del}(F1, w1) \rightarrow \text{OMAP} \rightarrow \text{Returns the leaf that contains } F1, w1
\]

Tree for keyword \( w1 \) (\( N=8 \))
**QOS --- Main Idea**

**Idea:** For each keyword $w$ create a data structure that helps us avoid the deleted regions.

Del ($F_1, w_1$)

Del ($F_2, w_1$)

Returns the leaf that contains $F_2, w_2$

Tree for keyword $w_1$ (N=8)
**QOS --- Main Idea**

**Idea:** For each keyword \( w \) create a data structure that helps us avoid the deleted regions.

- \( \text{Del (F1, w1)} \)
- \( \text{Del (F2, w1)} \)

Returns the leaf that contains \( F2, w2 \)

Tree for keyword \( w1 \) (\( N=8 \))
**QOS --- Main Idea**

**Idea:** For each keyword \( w \) create a data structure that helps us avoid the deleted regions.

- \( \text{Del (F1, } w_1) \)
- \( \text{Del (F2, } w_1) \)
- \( \text{Del (F10, } w_1) \)

OMAP

Returns the leaf that contains \( F_{10}, w_2 \)

Tree for keyword \( w_1 \) (N=8)
**QOS --- Main Idea**

**Idea:** For each keyword $w$ create a data structure that helps us avoid the deleted regions.

- $\text{Del (F1, w1)}$
- $\text{Del (F2, w1)}$
- $\text{Del (F10, w1)}$

Returns the leaf that contains $F10, w2$

Tree for keyword $w1$ (N=8)
QOS --- Main Idea

**Idea:** For each keyword $w$ create a data structure that helps us avoid the deleted regions

Query($w_1$) returns $i_w$, the number of inserts for $w_1$

Search cost = $O(\log^2 N + n_w \log i_w)$

Computes the **Best Range Cover** of $[1,i_w]$
Experimental Evaluation

- We implemented $\text{SDa}$, $\text{SDd}$ and $\text{QOS}$ in C++
  - OpenSSL for cryptographic operations
  - AES-NI enabled

- We compare our schemes with the previous state-of-the-art DSE schemes
  - HORUS and MITRA

- We measured search time, update time, and communication size
  - Synthetic dataset of 100 to 100M records
  - Real dataset of 6M crime incidents in Chicago

- Experiments using r5.8xlarge AWS machines
  - 32-core Intel Xeon 8259CL 2.5GHz processor
  - Running Ubuntu 16.04 LTS, with 256GB RAM, 100GB SSD (GP2), and AES-NI enabled.

Our code is available here: https://github.com/jgharehchamani/Small-Client-SSE
Search Time with 10% Deletions

- **QOS** is faster $14x-16531$ than **HORUS**

- **SDa** and **SDd** are $85x$ and $20x$ faster than **MITRA**

- **Synthetic Dataset** $1M$ records

- **Result size** $100$ records
Search Time with 0-90% Deletion Percentage

- Synthetic Dataset $1M$ records and $i_w = 20K$
Update time

MITRA is up to 21x faster than SDd

HORUS is up to 2x faster than QOS

- Synthetic Dataset 100-100M records
## Conclusion

<table>
<thead>
<tr>
<th></th>
<th>Search</th>
<th>Update</th>
<th>Search RT</th>
<th>BP-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moneta</td>
<td>$\tilde{O}(a_w \log N + \log^3 N)$</td>
<td>$\tilde{O}(\log^2 N)$</td>
<td>2</td>
<td>Type-I</td>
</tr>
<tr>
<td>OMAP+Mitra</td>
<td>$O(a_w + \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDa</td>
<td>$O(a_w + \log N)$</td>
<td>$O(\log N)$ (am.)</td>
<td>1</td>
<td>Type-II</td>
</tr>
<tr>
<td>SDd</td>
<td>$O(a_w + \log N)$</td>
<td>$O(\log^3 N)$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ORION</td>
<td>$O(n_w \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HORUS</td>
<td>$O(n_w \log d_w \log N + \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td>Type-III</td>
</tr>
<tr>
<td>QOS</td>
<td>$O(n_w \log i_w + \log^2 N)$</td>
<td>$O(\log^3 N)$</td>
<td></td>
<td>Type-III</td>
</tr>
</tbody>
</table>

### SDa and SDd
- **20x-85x** faster search time than MITRA
- Minimal client storage & non-interactive search

### QOS (for delete intensive query workloads)
- is the state-of-the-art DSE with **quasi-optimal** search time
- **14x-16531x** faster search time than HORUS
- better search time than MITRA and SDd after different deletion ratios between **40%-80%**
### Thank You! Questions?

<table>
<thead>
<tr>
<th></th>
<th>Search</th>
<th>Update</th>
<th>Search RT</th>
<th>BP-Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moneta</td>
<td>$\tilde{O}(a_w \log N + \log^3 N)$</td>
<td>$\tilde{O}(\log^2 N)$</td>
<td>2</td>
<td>Type-I</td>
</tr>
<tr>
<td>OMAP+Mitra</td>
<td>$O(a_w + \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDa</td>
<td>$O(a_w + \log N)$</td>
<td>$O(\log N)$ (am.)</td>
<td>1</td>
<td>Type-II</td>
</tr>
<tr>
<td>SDd</td>
<td>$O(a_w + \log N)$</td>
<td>$O(\log^3 N)$</td>
<td>1</td>
<td>Type-II</td>
</tr>
<tr>
<td>ORION</td>
<td>$O(n_w \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td>Type-I</td>
</tr>
<tr>
<td>HORUS</td>
<td>$O(n_w \log d_w \log N + \log^2 N)$</td>
<td>$O(\log^2 N)$</td>
<td></td>
<td>Type-III</td>
</tr>
<tr>
<td>QOS</td>
<td>$O(n_w \log i_w + \log^2 N)$</td>
<td>$O(\log^3 N)$</td>
<td></td>
<td>Type-III</td>
</tr>
</tbody>
</table>

### SDa and SDd
- **20x-85x** faster search time than MITRA
- Minimal client storage & non-interactive search

### QOS (for delete intensive query workloads)
- is the state-of-the-art DSE with **quasi-optimal** search time
- **14x-16531x** faster search time than HORUS
- better search time than MITRA and SDd after different deletion ratios between **40%-80%**