Cloud Storage
Two Simple Solutions to Search

Q: can we achieve the best of both?
Searchable Symm. Encryption

\[ \text{Enc}_K [\text{Document}] \]
Security Definitions

- Security against chosen-keyword attack
  [Goh03, Chang-Mitzenmacher05, Curtmola-Garay-K.-Ostrovsky06]

  **CKA1:** “Protects files and keywords even if chosen by adversary”

- Security against *adaptive* chosen-keywords attacks
  [Curtmola-Garay-K.-Ostrovsky06]

  **CKA2:** “Protects files and keywords even if chosen by adversary, and even if chosen as a function of ciphertexts, index, and previous results”
Security Definitions

- **UC** [Kurosawa-Ohtaki12]
  - Universal composability [Canetti01]

**UC:** “Remains CKA2-secure even if composed arbitrarily”
Simulation-based definition

```
given the encrypted index, encrypted files and search tokens, no adversary can learn any information about the files and the search keywords other than what can be deduced from the access and search patterns…
```

```
…even if queries are made adaptively
```

access pattern: pointers to (encrypted) files that satisfy search query

search pattern: whether a search query is repeated
SSE Parameters

- **Parameters**
  - \( n \): number of files in collection
  - \(|f|\): size of file collection
  - \( m \): number of keywords

- **Client-side**
  - Security: CKA1, CKA2, UC
  - Token size: \( O(1) \) to \( O(n) \)

- **Server-side**
  - Search time: OPT, \( O(n) \), \( O(|f|) \)
<table>
<thead>
<tr>
<th>Scheme</th>
<th>Dynamism</th>
<th>Security</th>
<th>Search</th>
<th>Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SWP00]</td>
<td>No</td>
<td>CPA</td>
<td>$O(</td>
<td>f</td>
</tr>
<tr>
<td>[Goh03]</td>
<td>Yes</td>
<td>CKA1</td>
<td>$O(n)$</td>
<td>$O(n/p)$</td>
</tr>
<tr>
<td>[CM05]</td>
<td>No</td>
<td>CKA1</td>
<td>$O(n)$</td>
<td>$O(n/p)$</td>
</tr>
<tr>
<td>[CGKO06] #1</td>
<td>No</td>
<td>CKA1</td>
<td>$O(\text{OPT})$</td>
<td>N/A</td>
</tr>
<tr>
<td>[CGKO06] #2</td>
<td>No</td>
<td>CKA2</td>
<td>$O(\text{OPT})$</td>
<td>N/A</td>
</tr>
<tr>
<td>[CK10]</td>
<td>No</td>
<td>CKA2</td>
<td>$O(\text{OPT})$</td>
<td>N/A</td>
</tr>
<tr>
<td>[vLSDHJ10]</td>
<td>Yes</td>
<td>CKA2</td>
<td>$O(\log m)$</td>
<td>N/A</td>
</tr>
<tr>
<td>[KO12]</td>
<td>No</td>
<td>UC</td>
<td>$O(n)$</td>
<td>N/A</td>
</tr>
<tr>
<td>[KPR12]</td>
<td>Yes</td>
<td>CKA2</td>
<td>$O(\text{OPT})$</td>
<td>N/A</td>
</tr>
<tr>
<td>[this work]</td>
<td>Yes</td>
<td>CKA2</td>
<td>$O(\text{OPT}\cdot\log(n))$</td>
<td>$O\left(\frac{\text{OPT}}{p}\cdot\log(n)\right)$</td>
</tr>
</tbody>
</table>
Limitations of Inverted Index Approach

- Static
- Sequential
- [K.-Papamanthou-Roeder12]
  - 😊 Optimal search time
  - 😊 Handles updates
  - 😞 Overly complex
  - 😞 Sequential
A New Approach
Tree-Based Approach

- Advantages
  - Sub-linear
  - Dynamic
  - Parallelizable
  - Simple

- Disadvantages
  - Not optimal
  - Interactive updates
Red-Black Trees

Worst-case
1. Search: $O(\log(n))$
2. Add: $O(\log(n))$
3. Delete: $O(\log(n))$
A New Data Structure

- Keyword Red-Black (KRB) Trees

\[ \mathcal{W} = \{w_1, \ldots, w_t\} \]

Search: \( O(\frac{\text{OPT}}{p} \cdot \log(n)) \)
Add/delete: \( O(\frac{\#f}{p} \cdot \log(n)) \)

If \( p = \omega(\log(n)) \) then search is \( o(\text{OPT}) \)
Encrypting KRB Trees

\[ \text{sk}_3 = \text{Gen}(1^k; F(w_3)) \]

\[ H(N_{id} | P_{K2}(w_3)) \]

\[ \text{Enc}(\text{sk}_3, 0) \ldots \]

\[ \text{Enc}(\text{sk}_3, 1) \ldots \]

1010\ldots1 \xrightarrow{P_{K2}(w_3)} 010\ldots1\ldots1
Encrypting KRB Trees

... Enc(sk₃, 1) ....
... Enc(sk₃, 0) ....
Searching KRB Trees

\[
\text{Token}_K(w_3) = P_{K_2}(w_3), \ sk_3
\]

\[\begin{align*}
\text{Enc}(sk_3, 1) \quad &\text{etc.} \\
\text{Enc}(sk_3, 0) \quad &\text{etc.}
\end{align*}\]

\[H(N_{id} \mid P_{K_2}(w_3))\]
Updating KRB Trees

0110....1

0010....0 010....1 0010....1

+ w_1
Updating Encrypted KRB Trees

\[ \text{Enc}(sk_3, 0) \ldots \]
\[ \text{Enc}(sk_3, 1) \ldots \]
\[ +w_1 \]
Updating Encrypted KRB Trees

File ID

Enc(sk₃, 1) .... Enc(sk₃, 0) ....
Enc(sk₃, 0) .... Enc(sk₃, 1) ....
Enc(sk₃, 1) .... Enc(sk₃, 0) ....
Thanks!