Highly-Scalable
Searchable Symmetric Encryption

Presenter: Jacob Chen
Searchable symmetric encryption

- Queries are only limited to a single keyword
- In order to be practical, a system should support conjunctive search
- Symmetric encryption should be scalable
- All systems leak information
- A practical system must relax absolute privacy
Searchable symmetric encryption (cont’d)

+ Complexity should ideally be proportional to the number of matches of the least frequent term
+ FHE and ORAM are too costly
Contributions

- Developed the first non-generic sub-linear SSE schemes supporting conjunctive keyword search
- Assumes the adaptive adversary model
- Server cannot deduce anything beyond the precisely defined leakage profile
  - Leakage profiles include total size of database
  - Access patterns (intersection of two sets of results and queries)
  - There is never direct exposure of plaintext data or searched values
Contributions (cont’d)

+ Complexity of the search protocols is independent of the number of documents in the database

+ Extends to support answers for Boolean queries
SSE Syntax

- A database has $d$ documents
- Each document has a set of keywords $W_i$
- Output from the SSE protocol are indices, $\text{ind}$ (identifiers)
- Client program uses $\text{ind}$ to retrieve encrypted documents
SSE Syntax (cont’d)

- A SSE Scheme $\pi$ consists of EDBSetup and a Search Protocol
- EDBSetup takes input DB and outputs secret key $K$ along with EDB
- $K$ is given to the client
- EDB is given to server
T-Sets

- A T-set is a tuple set
- T-set associates a list of fixed-size data tuples with each keyword in the database

\[ \Sigma = (\text{TSetSetup}, \text{TSetGetTag}, \text{TSetRetrieve}) \]

\[ (\text{TSet}, K_T) = \text{TSetSetup}(T) \]

\[ s\text{-tag} = \text{TSetGetTag}(K_T, w) \]

\[ \text{TSetRetrieve}(\text{TSet}, s\text{-tag}) \text{ returns } T[w] \]
## T-Sets (cont’d)

<table>
<thead>
<tr>
<th>Index</th>
<th>Strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>$\text{ind}_1$, $\text{ind}_2$, $\text{ind}_3$, ...</td>
</tr>
<tr>
<td>$w_2$</td>
<td>$\text{ind}_4$, $\text{ind}_5$, $\text{ind}_6$, ...</td>
</tr>
<tr>
<td>$w_3$</td>
<td>$\text{ind}_7$, $\text{ind}_8$, $\text{ind}_9$, ...</td>
</tr>
</tbody>
</table>
**Single-Keyword SSE Scheme**

For each $w$ in the database, find its corresponding document $\text{ind}$

**Database** $\rightarrow$ $\langle w_i, \text{ind}_1, \text{ind}_2, \ldots \rangle$

Initialize $T$ to be an empty array

For each $w$ in $W$, build $T[w]$ as follows:

1. Initialize $t$ as an empty list
2. For all $\text{ind}$ in $DB(w)$, in random order
   - $K_s, w \rightarrow F \rightarrow K_e$
   - $\text{Enc}(K_s, \text{ind}) \rightarrow e$
   - $t = [t, e]$

$T[w] = t$

$T \rightarrow \text{TSetSetup} \rightarrow \text{Tset, KT}$

Give $K_S, KT$ to client, and give server TSet

---

**Client Side:**

- $K_S, w \rightarrow F \rightarrow K_e$
- For each $e$ in $t$
  - $\text{Dec} \rightarrow \text{ind}$
SSE Schemes

- The server’s work grows linearly with the number of documents
- Scheme does not support unstructured data
- We want to develop a search protocol that scales with the size of the smallest $DB(w_i)$ among the conjunctive terms $w_i$
- Naive solution is to run a single keyword SSE scheme for each conjunctive word
Basic Cross Tag protocol

For each \( w \) in the database, find its corresponding document ind

\[ \text{Database} \rightarrow \langle w_i, \text{ind}_1, \text{ind}_2, ... \rangle \]

Initialize \( T \) to be an empty array,
Initialize \( \text{Xset} \) to an empty set

For each \( w \) in \( W \), build \( T[w] \) and \( \text{Xset} \) as follows:

- Initialize \( t \) as an empty list
  \[ K_s, w \rightarrow F \rightarrow K_e \]
  \[ K_x, w \rightarrow F \rightarrow \text{XTrap} \]

- For all \( \text{ind} \) in \( \text{DB}(w) \), in random order
  \[ \text{Enc}(K_s, \text{ind}) \rightarrow e \]
  \[ t = [t, e] \]
  \[ \text{XTrap, ind} \rightarrow \text{f} \rightarrow \text{XTag} \]

\( X\text{set} = [X\text{set}, X\text{Tag}] \)

\( T[w] = t \)

Give \( K_s, K_x, K_t \) to client, and give server \( T\text{set}, X\text{set} \)

Client takes as input \( (K_s, K_t, K_x) \) and keywords \( w_1, w_2, ... w_n \) to query

\( K_t, w_1 \rightarrow \text{TSetGetTag} \rightarrow s\text{-tag} \)

\( K_s, w_1 \rightarrow F \rightarrow \text{K_e} \)

For each \( i = 2, 3, ..., n \)

\( K_x, w_i \rightarrow F \rightarrow \text{XTrap } i \)

Server

\( K_e \rightarrow \text{TSet} \rightarrow s\text{-tag} \)

For each ciphertext \( e \) in \( t \), server computes:

\[ \text{Dec} \rightarrow \text{ind} \]

\[ X\text{set} \rightarrow \text{f} \rightarrow \text{XTrap } i \]

Client
Basic Cross Tag protocol cont’d

- The server’s work in BXT scales with $n \times |DB(w_1)|$
- This protocol improves substantially security wise in terms of leakage
- Vulnerable to a simple attack: server can use xtrap and save it for later use
- Performance and privacy improves with lighter $s$-terms
Oblivious Cross-tags Protocol

**Client:**
- Retrieve e from server
- \( K_s, w_s \) to client

**Server:**
- If \( x_{token}[c], y \) belongs to Xset
  - \( e \) to client

**Protocol Description:**
- **PRF**: Generates \( K_s \) and \( PRFp \)
- **K_s, W**: For each \( w \) in the database, find its corresponding document ind
- **Initialize T to be an empty array**, **Xset to an empty set**
- **For each \( w \) in W**, build **T[w]** and **Xset** as follows:
  - Initialize \( t \) as an empty list
  - \( K_s, W \rightarrow F \rightarrow K_e \)
  - \( K_x, W \rightarrow F \rightarrow XTrap \)
  - For all \( ind \) in DB(w), in random order, init a counter \( c = 0 \)
  - \( K_s, ind \rightarrow PRFp \rightarrow xind \)
  - \( K_z, w[c] \rightarrow PRFp \rightarrow Z \)
  - \( xind \rightarrow Y \rightarrow 1 \)
  - \( Enc(K_s, ind) \rightarrow e \)

- \( t = [t, (e, y)] \)
- \( Xind, K_x, w \rightarrow g \rightarrow XTag \)
- **Xset = [Xset, XTag]**
- **T[w] = t**
- **TsetSetup**
- **Tset, K_r**

**Client Actions:**
- Give \( K_s, K_x, K_r \) to client, and give server Tset, Xset

**Server Actions:**
- **TSetSetup**
- **Tset, K_r**
- For \( c = 1, 2, ..., n \) until server says stop
- \( K_z, w[c], K_x, w \rightarrow G^* \rightarrow x_{token}[c] \)
- \( x_{token}[c] \)

**Client Algorithm:**
- **TSetGetTag**
- **s-tag**
- For \( c = 1, 2, ..., n \)
- **G^***
- \( x_{token}[c] \)
Oblivious Cross-tags Protocol

- Single round of communication
- Achieves better security