FORWARD PRIVATE SEARCHABLE ENCRYPTION

DATE 27/10/2016

ACM CCS - RAPHAEL BOST
Searchable Encryption

Outsource data
Searchable Encryption

Outsource data

* securely
Searchable Encryption

Outsource data

- securely
- keep search functionalities
Searchable Encryption

Outsource data

- securely
- keep search functionalities
- aimed at efficiency
Generic Solutions

Fully Homomorphic Encryption, MPC, ORAM
Generic Solutions

Fully Homomorphic Encryption, MPC, ORAM

✓ Perfect security
Generic Solutions

Fully Homomorphic Encryption, MPC, ORAM

✓ Perfect security

✗ Large overhead (computation, communication)
Ad-hoc Constructions

Can we get more efficient solutions?
Ad-hoc Constructions

Can we get more efficient solutions?

* Yes, but …
Ad-hoc Constructions

Can we get more efficient solutions?

* Yes, but …

* … we have to leak some information
Ad-hoc Constructions

Can we get more efficient solutions?

* Yes, but …

* … we have to leak some information

Security/performance tradeoff
Property Preserving Encryption

Deterministic Encryption, OPE, ORE

✓ Legacy compatible

✓ Very Efficient
Property Preserving Encryption

Deterministic Encryption, OPE, ORE

✓ Legacy compatible

✓ Very Efficient

✗ Not secure in practice (cf. next talks)
Index-Based SE [CGKO’06]
Index-Based SE [CGKO’06]

Structured encryption of the reversed index: search queries allow partial decryption
Index-Based SE \cite{CGKO'06}

Structured encryption of the reversed index: search queries allow partial decryption

- Search leakage:
  - repetition of queries (search pattern)
Index-Based SE [CGKO’06]

Structured encryption of the reversed index: search queries allow partial decryption

- Search leakage:
  - repetition of queries (search pattern)

- Update leakage:
  - updated documents
  - repetition of updated keywords
File Injection Attacks [ZKP’16]

Non-adaptive file injection attacks

- Insert purposely crafted documents in the DB. Use binary search to recover the query

<table>
<thead>
<tr>
<th></th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>$k_2$</td>
<td>$k_3$</td>
<td>$k_4$</td>
</tr>
<tr>
<td>$k_5$</td>
<td>$k_6$</td>
<td>$k_7$</td>
<td>$k_8$</td>
</tr>
</tbody>
</table>
File Injection Attacks [ZKP’16]

Non-adaptive file injection attacks

- Insert purposely crafted documents in the DB. Use binary search to recover the query.

<table>
<thead>
<tr>
<th></th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>k1</td>
<td>k1</td>
<td>k1</td>
</tr>
<tr>
<td></td>
<td>k2</td>
<td>k2</td>
<td>k2</td>
</tr>
<tr>
<td></td>
<td>k3</td>
<td>k3</td>
<td>k3</td>
</tr>
<tr>
<td></td>
<td>k4</td>
<td>k4</td>
<td>k4</td>
</tr>
<tr>
<td></td>
<td>k5</td>
<td>k5</td>
<td>k5</td>
</tr>
<tr>
<td></td>
<td>k6</td>
<td>k6</td>
<td>k6</td>
</tr>
<tr>
<td></td>
<td>k7</td>
<td>k7</td>
<td>k7</td>
</tr>
<tr>
<td></td>
<td>k8</td>
<td>k8</td>
<td>k8</td>
</tr>
</tbody>
</table>
File Injection Attacks \[\text{[ZKP’16]}\]

Non-adaptive file injection attacks

- Insert purposely crafted documents in the DB. Use binary search to recover the query

<table>
<thead>
<tr>
<th></th>
<th>$k_1$</th>
<th>$k_2$</th>
<th>$k_3$</th>
<th>$k_4$</th>
<th>$k_5$</th>
<th>$k_6$</th>
<th>$k_7$</th>
<th>$k_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
File Injection Attacks \[\text{[ZKP'16]}\]

Non-adaptive file injection attacks

- Insert purposely crafted documents in the DB.

Use binary search to recover the query

\[
\begin{array}{cccccccc}
D_1 & k_1 & k_2 & k_3 & k_4 & k_5 & k_6 & k_7 & k_8 \\
D_2 & k_1 & k_2 & k_3 & k_4 & k_5 & k_6 & k_7 & k_8 \\
D_3 & k_1 & k_2 & k_3 & k_4 & k_5 & k_6 & k_7 & k_8 \\
\end{array}
\]

log K injected documents
File Injection Attacks [ZKP’16]

* Insert purposely crafted documents. Use binary search to recover the query

⇒ Counter measure: no more than $T$ keywords/doc.

$$(K/T) \cdot \log T \text{ injected documents}$$
File Injection Attacks [ZKP'16]

* Insert purposely crafted documents. Use binary search to recover the query

⇒ Counter measure:
no more than T keywords/doc.

\[(K/T) \cdot \log T \text{ injected documents}\]

* Adaptive version of the attack

\[\log T \text{ using prior knowledge}\]
Adaptive File Injection

- The adaptive attack uses the update leakage:
Adaptive File Injection

- The adaptive attack uses the update leakage:
  - Most SE schemes leak if a newly inserted document matches a previous search query.
Adaptive File Injection

- The adaptive attack uses the update leakage:
  - Most SE schemes leak if a newly inserted document matches a previous search query
  - We need SE schemes with oblivious updates
Adaptive File Injection

- The adaptive attack uses the update leakage:
  - Most SE schemes leak if a newly inserted document matches a previous search query
  - We need SE schemes with oblivious updates

Forward Privacy
Forward Privacy

* Forward private: an update does not leak any information on the updated keywords
Forward Privacy

- Forward private: an update does not leak any information on the updated keywords
- Secure online build of the EDB
Forward Privacy

- Forward private: an update does not leak any information on the updated keywords
- Secure online build of the EDB
- Only one existing scheme so far [SPS’14]
Forward Privacy

- Forward private: an update does not leak any information on the updated keywords
- Secure online build of the EDB
- Only one existing scheme so far [SPS’14]
  ➡ ORAM-like construction
Forward Privacy

- Forward private: an update does not leak any information on the updated keywords
- Secure online build of the EDB
- Only one existing scheme so far [SPS’14]
  - ORAM-like construction
  - Inefficient updates
Forward Privacy

- Forward private: an update does not leak any information on the updated keywords
- Secure online build of the EDB
- Only one existing scheme so far [SPS’14]
  ➔ ORAM-like construction
  ✗ Inefficient updates
  ✗ Large client storage
Σοφος

- Forward private index-based scheme
- Low search and update overhead
- Simpler than [SPS’14]
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)
Add $(\text{ind}_1, \ldots, \text{ind}_c)$ to $w$

Search $w$

$\text{UT}_1(w)$  $\text{UT}_2(w)$  $\ldots$  $\text{UT}_c(w)$
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)

Search \(w\)

\(\text{UT}_1(w)\) \(\text{UT}_2(w)\) \(\ldots\) \(\text{UT}_c(w)\)

\(\text{ST}(w)\)
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)

Search \(w\)

\(\text{UT}_1(w)\) \(\text{UT}_2(w)\) \(\ldots\) \(\text{UT}_c(w)\)

\(\text{ST}(w)\)
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)

Search \(w\)

\[
\begin{align*}
\text{UT}_1(w) & \quad \text{UT}_2(w) & \cdots & \text{UT}_c(w) \\
\end{align*}
\]

\(\text{ST}(w)\)
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)

Search \(w\)

\[ \begin{align*}
\text{UT}_1(w) & \quad \quad \text{UT}_2(w) & \quad \ldots & \quad \text{UT}_c(w) \\
\end{align*} \]

\[ \text{ST}(w) \]
Add $(\text{ind}_1, \ldots, \text{ind}_c)$ to $w$

Search $w$
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)

Search \(w\)

Add \(\text{ind}_{c+1}\) to \(w\)

\[\begin{array}{ccc}
\text{UT}_1(w) & \rightarrow & \text{UT}_2(w) & \rightarrow & \cdots & \rightarrow & \text{UT}_c(w) \\
\text{ST}_1(w) & \leftarrow & \text{ST}_2(w) & \leftarrow & \cdots & \leftarrow & \text{ST}_c(w)
\end{array}\]
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)

Search \(w\)

Add \(\text{ind}_{c+1}\) to \(w\)
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)

Search \(w\)

Add \(\text{ind}_{c+1}\) to \(w\)
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)

Search \(w\)

Add \(\text{ind}_{c+1}\) to \(w\)
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)

Search \(w\)

Add \(\text{ind}_{c+1}\) to \(w\)

\[\text{UT}_1(w) \quad \text{UT}_2(w) \quad \ldots \quad \text{UT}_c(w) \quad \text{UT}_{c+1}(w)\]

\[\text{ST}_1(w) \quad \text{ST}_2(w) \quad \ldots \quad \text{ST}_c(w) \quad \text{ST}_{c+1}(w)\]
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)

Search \(w\)

Add \(\text{ind}_{c+1}\) to \(w\)
Naïve solution: $\text{ST}_i(w) = F(K_w, i)$
Naïve solution: $ST_i(w) = F(K_w, i)$

✗ Client needs to send $c$ tokens
Naïve solution: $ST_i(w) = F(K_w, i)$

✗ Client needs to send $c$ tokens
✗ Sending only $K_w$ is not forward private
Naïve solution: \( ST_i(w) = F(K_w, i) \)

- Client needs to send \( c \) tokens
- Sending only \( K_w \) is not forward private

Use a trapdoor permutation
Naïve solution: \( ST_i(w) = F(K_w, i) \)
- Client needs to send \( c \) tokens
- Sending only \( K_w \) is not forward private

Use a trapdoor permutation
Naïve solution: ST_i(w) = F(K_w, i)

✗ Client needs to send c tokens
✗ Sending only K_w is not forward private

* Use a trapdoor permutation
Client stores $W[w] := \text{ST}_c(w)$
Client stores $W[w] := ST_c(w)$

Search $w$: send $ST_c(w)$
Client stores $W[w] := ST_c(w)$

Search $w$: send $ST_c(w)$

Update: $W[w] := \pi^{-1}_{SK}(ST_c(w))$
Search:

- Client: constant
- Server: $O(|DB(w)|)$
Search:
- Client: constant
- Server: $O(|DB(w)|)$

Update:
- Client: constant
- Server: constant
Search:
- Client: constant
- Server: $O(|DB(w)|)$

Update:
- Client: constant
- Server: constant

Optimal
Storage:

- Client: \( O(K) \)
- Server: \( O(|DB|) \)
Σοφος

* TDP π? RSA or Rabin
Σοφος

- TDP π? RSA or Rabin
  - Elements (STs) are large (2048 bits).
Σοφος

- TDP π? RSA or Rabin
  - Elements (STs) are large (2048 bits).
  - Huge client storage.
Σοφος

- TDP π? RSA or Rabin
  - Elements (STs) are large (2048 bits).
  - Huge client storage.
- Client only stores c, pseudo-randomly generates ST₁(w), computes STₖ(w) on the fly
Σοφος

• TDP π? RSA or Rabin
  ❌ Elements (STs) are large (2048 bits).
  ❌ Huge client storage.

• Client only stores \( c \), pseudo-randomly generates \( ST_1(w) \), computes \( ST_c(w) \) on the fly
  ✓ Efficient (non-iterative) using RSA

• Search is embarrassingly parallelizable

\[
x^d \cdot \cdots = x^{(d^c \mod \phi(N))} \mod N
\]
Σοφος - Security

- Update leakage: nothing Forward private!

- Search leakage:
  - search pattern of \( w \)
  - ‘history’ of \( w \): the timestamped list of updates of keyword \( w \)

Adaptive security (ROM)
Σοφος - Evaluation

- C/C++ full fledged implementation
- Server KVS: RocksDB
- Evaluated on a desktop computer
  4 GHz Core i7 CPU (16 cores), 16GB RAM, SSD
Σοφος - Evaluation

2M keywords, 140M entries
5.25GB server storage, 64.2 MB Client storage

![Graph showing search time per matching entry (ms) vs number of matching documents with and without RPC.](image-url)
Σοφος

* Provable forward privacy
Σοφος

- Provable forward privacy
- Very simple
Σοφος

- Provable forward privacy
- Very simple
- Efficient search (IO bounded)
Σοφος

- Provable forward privacy
- Very simple
- Efficient search (IO bounded)
- Asymptotically efficient update (optimal)
Σοφος

- Provable forward privacy
- Very simple
- Efficient search (IO bounded)
- Asymptotically efficient update (optimal)
  - In practice, very low update throughput (4300 e/s - 20x slower than other work)
THANKS!

Paper: http://ia.cr/2016/728

Code: https://gitlab.com/sse/sophos