FORWARD PRIVATE
SEARCHABLE ENCRYPTION
Searchable Encryption

- Outsource data ...
- ... securely
- ... keep search functionalities
Generic Solutions

We can use generic tools to solve this problem:

- Fully Homomorphic encryption
  - Run all computations on the server
    Complexity linear in the DB size
- Oblivious RAM
  - Hide access pattern but…
    ORAM lower bound (logarithmic)
Ad-hoc Constructions

Can we get more efficient solutions?

- Yes, but …
- … we have to leak some information

Security/performance tradeoff
Security of SE
Security of SE

Update(+, w, ind)
Security of SE

Everything the server learns, he can compute from the leakage
The search protocol can be simulated from the leakage
Common Leakage

- **Search leakage:**
  - repetition of queries (aka. search pattern)
  - results

- **Update leakage:**
  - updated documents
  - repetition of updated keywords
  - ...

Previous Results

- First constructions [SWP00]
- Formalization of the security model [CGKO06]
- Efficient dynamic constructions [KPR12]
- Boolean queries & scalability [CJJKRS13]
  - various extensions (dynamisms, wildcards, range queries, …)
- Reduced update leakage [SPS14]
- …
Security-Performance Tradeoff

- FHE
- ORAM
- [SPS'14]
- [KPR12][CJJ+'13]

Legacy compatible
Plain
Leakage-Abuse Attacks

- ‘Everything the server learns, he can compute from the leakage’
  ➞ What can be computed from the leakage?
- Recover the queried keywords from the leakage
‘Passive’ Attacks

- [IKK’12]: Using a co-occurrence probability matrix, the attacker can recover from 100% to 65% of the queries

- [CGPR’15]: Improvement of the IKK attack, 100% recovery
  - Use padding as a countermeasure
‘Active’ 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>k₁</th>
<th>k₂</th>
<th>k₃</th>
<th>k₄</th>
<th>k₅</th>
<th>k₆</th>
<th>k₇</th>
<th>k₈</th>
</tr>
</thead>
<tbody>
<tr>
<td>D₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₂</td>
<td>k₁</td>
<td>k₂</td>
<td>k₃</td>
<td>k₄</td>
<td>k₅</td>
<td>k₆</td>
<td>k₇</td>
<td>k₈</td>
</tr>
<tr>
<td>D₃</td>
<td>k₁</td>
<td>k₂</td>
<td>k₃</td>
<td>k₄</td>
<td>k₅</td>
<td>k₆</td>
<td>k₇</td>
<td>k₈</td>
</tr>
</tbody>
</table>

log K injected documents
‘Active’ Attacks

* [ZKP’16]: Non-adaptive file injection attacks
  * Insert purposely crafted documents in the DB. Use binary search to recover the query
  * Counter measure: no more than $T$ kw./doc.
    
    $$(K/T) \cdot \log T \text{ injected documents}$$
  
* Adaptive version of the attack
  
    $$(K/T) + \log T \text{ injected documents}$$
‘Active’ Adaptive Attacks

- [ZKP’16]: File injection attacks
  - Adaptive version of the attack
    \[(K/T) + \log T\] injected documents
  - If the attacker has prior knowledge about the database (e.g. frequency distribution)
    \[\log T\] injected documents
‘Active’ Adaptive Attacks

- All these adaptive attacks use the update leakage:
  - For an update, most SE schemes leak if the inserted document matches a previous query
  - We need SE schemes with oblivious updates

Forward Privacy
Forward Privacy

- An SE scheme is forward private if its update protocol does not leak any information about the updated keywords

\[ L(op,w,ind) = L'(op, ind) \]

- Important feature: secure online build of the EDB
- Only one existing scheme so far [SPS’14]
  ➡ Very close to ORAM (logarithmic updates)
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)

Search \(w\)

\[
\begin{align*}
\text{UT}_1(w) & \quad \text{UT}_2(w) & \quad \ldots & \quad \text{UT}_c(w) \\
\text{ST}(w) & \quad \ldots & \quad \text{ST}(w)
\end{align*}
\]
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)  
Search \(w\)  
Add \(\text{ind}_{c+1}\) to \(w\)
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
Σοφος - Complexity

- Search(w):
  Client: $O(1)$
  Server: $O(|DB(w)|)$

- Update(+, w, ind):
  Client: $O(1)$
  Server: $O(1)$

- Storage:
  Client: $O(K)$
  Server: $O(N)$

Optimal
Σοφος

- TDP π? RSA or Rabin
  - Elements (STs) are large (2048 bits).
  - Client storage is impractical
- Pseudo-randomly generate \( ST_0(w) \), and compute \( ST_c(w) \) on the fly (only \( c \) is stored by the client)
  - Efficient (non-iterative) using RSA
- Search is embarrassingly parallelizable

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

- Update leakage: nothing

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

Adaptive security (ROM)
**Σοφος - Evaluation**

- C/C++ full fledged implementation
- Server KVS: RockDB
- Evaluated on a desktop computer
  4 GHz Core i7 CPU, 8GB RAM, SSD

[https://gitlab.com/sse/sophos](https://gitlab.com/sse/sophos)
Σοφος - Evaluation

![Graph showing search time per matching entry vs. number of matching documents for different database sizes.]

- Database with 14e5 entries
- Database with 14e6 entries
- Database with 14e7 entries

The graph illustrates the decrease in search time as the number of matching documents increases for databases of different sizes.
Provable forward privacy

Efficient search

Asymptotically efficient update (optimal)

In practice, very low update throughput (4300 p/s - 20x slower than other work)
Security-Performance Tradeoff

- FHE
- ORAM
- Σοφος
- [SPS’14]
- [KPR12][CJJ+’13]

Legacy compatible
- Plain
Ongoing/future work

- Improve the update throughput (get rid of RSA)
- Dynamic padding
- Thwart (non-adaptive) file injection attacks
THANKS!