FORWARD PRIVATE SEARCHABLE ENCRYPTION & BEYOND

DATE 22/02/2017

MIT - RAPHAEL BOST
Searchable Encryption

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

Fully Homomorphic Encryption, MPC, ORAM

✓ Perfect security

✗ Large overhead (computation, communication)
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
✗ Not secure in practice (e.g. attacks on CryptDB)
Index-Based SE [CGK0’06]

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
Security-Performance Tradeoff

Security

Performance

FHE

ORAM

[SPS’14]

[KPR12][CJJ+’13]

Legacy compatible

Plain
‘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
File Injection Attacks \cite{ZKP16}

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>

\[ \log K \text{ injected documents} \]
‘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

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

- If the attacker has prior knowledge about the database (e.g. frequency distribution)

\[\log T \text{ injected documents}\]
‘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

- 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
- A lot simpler than [SPS’14]
Add \((\text{ind}_1, \ldots, \text{ind}_c)\) to \(w\)

Search \(w\)

\[ \text{UT}_1(w) \quad \text{UT}_2(w) \quad \cdots \quad \text{UT}_c(w) \]

\[ \text{ST}(w) \]
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 \cdots \quad \text{UT}_c(w) \quad \text{UT}_{c+1}(w)\)

\(\text{ST}_1(w)\) \quad \text{ST}_2(w) \quad \cdots \quad \text{ST}_c(w) \quad \text{ST}_{c+1}(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
• 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: $O(|DB(w)|)$

Update:

- Client: constant
- Server: constant

**Optimal**
Storage:

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

- TDP π? RSA or Rabin
  - Elements (STs) are large (2048 bits).
  - Client storage is impractical

- Client only stores c, pseudo-randomly generates ST₁(w), computes STᶜ(w) on the fly
  - Efficient (non-iterative) using RSA

- Search is embarrassingly parallelizable

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

- Update leakage: nothing
- Search leakage:
  - search pattern
  - ‘history’ of w: the timestamped list of updates of keyword w

Forward private

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

https://gitlab.com/sse/sophos
Σοφος - Evaluation

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

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

FHE
ORAM

[SPS'14]

Σοφος

[KPR'12][CJJ+’13]

Legacy compatible

Plain
BEYOND
FORWARD
PRIVACY

PRACTICAL ISSUES WITH
SEARCHABLE ENCRYPTION
AND OPEN PROBLEMS
Thwarting File Injections

- Σοφος only thwarts the adaptive file injection attacks
- Idea: randomly delay the insertion of entries in the database
- How to define the security of such countermeasures?
Locality

- Σοφος makes 1 random access/match
  - Even with SSDs, random disk accesses are very expensive

- One cannot construct a (static) SE scheme with optimal locality, linear storage, or optimal search complexity [CT’14]

- [ANSS’16] built a scheme with optimal loc., linear storage, and high read efficiency (log log N)
Σοφος - Locality

![Graph showing search time per matching entry (ms) vs. number of matching documents for databases with different entry counts. The graph illustrates the decrease in search time as the number of matching documents increases.](image-url)

- Database with 14e5 entries
- Database with 14e6 entries
- Database with 14e7 entries
Locality and Forward Priv.

- The [ANSS’16] solution is inherently static. What about dynamic schemes?
- Locality goes against forward privacy
  - Locality: put entries with the same kw. close
  - F.P.: entries matching the same kw. are unrelated
- I think there is a (complicated) lower bound involving locality, comm. complexity, DB size and read efficiency

Open Problem
Locality in practice

- Regroup entries matching the same keyword by (large) blocks
- [MM’17] combine this idea with ORAM to save 80% of the IOs during search
- Other proposal: cache search results
Other adversaries

- The literature only focuses on persistent adversaries. Could we have better guarantees against weaker ones?
- Snapshot adversaries, ‘late’ persistent adversaries
- Might be important in practice: e.g. when caching previous queries’ results
Backward Privacy

- Queries should not be executed over deleted documents (*cf.* secure deletion)
- Only interesting against ‘late’ persistent adversaries
- Achieved by ORAM. Looks hard to achieve efficiently (single interaction, low comm. complexity)
THANKS!

Paper: http://ia.cr/2016/728
Code: https://gitlab.com/sse/sophos