Encrypted Search: Leakage Suppression

Seny Kamara
How Should we Handle Leakage?

• **Approach #1:** ORAM simulation
  • Store and simulate data structure with ORAM
  • General-purpose
  • Zero-leakage (if data is transformed appropriately)
  • polylog overhead per read/write on top of simulation

• **Approach #2:** Custom oblivious structures
How Should we Handle Leakage?

• **Approach #3:** Rebuild [K.14]
  • Rebuild encrypted structure after $t$ queries
  • Set $t$ using cryptanalysis
  • Open question: can you rebuild encrypted structures?

• **Approach #4:** Leakage suppression
  • Suppression compilers
  • Suppression transforms
Q: can we reduce leakage?
Leakage Suppression via ORAM

• Common answer is “use ORAM!”
  • usually without any details
  • or experiments
• How exactly do we use ORAM to search?
ORM

**Setup time**

```
| | | | | |
```

**Query time**

```
Read(i)
| | | | | |
```

```
Write(i,v)
| | | | | |
```

```
ORAM.Setup
```

```
ORAM.Read(i)
```

```
ORAM.Write(i,v)
```
Leakage Suppression via ORAM

• ORAM supports read & write operations to an array
  • with polylog(n) cost
  • and leakage profile $\Lambda_{\text{ORAM}} = (\mathcal{L}_S, \mathcal{L}_Q) = (\text{dsize}, \perp)$

• ORAM is a “low-level” primitive
  • designed for read/write operations to an array
  • what if we want to query a more complex structure?

• Need to use ORAM simulation
ORAM Simulation

• Represent DS as an array and store in ORAM
• Client simulates $\text{Query(DS, q)}$ algorithm
  • replaces each $\text{Read}(i)$ with $\text{ORAM.Read}(i)$
  • replaces each $\text{Write}(i, v)$ with $\text{ORAM.Write}(i, v)$
ORAM Simulation

Setup time

```
\text{ORAM.Setup}
```

Query time

```
\text{Query(DS,q)}
```

- \text{Read(3)} → \text{ORAM.Read(3)}
- \text{Write(1,v)} → \text{ORAM.Write(1,v)}
- \text{Read(10)} → \text{ORAM.Read(10)}
ORAM Simulation

• Costs $O(T \cdot \text{polylog}(|DS|))$
  • where $T$ is runtime of $\text{Query}(DS,q)$
• Leakage profile
  • $\Lambda = (\text{dsize}, (\text{runtime, vol}))$
  • $\text{vol}$: size of response (can be suppressed with padding)
• Can we do better?
Q: can we do better than ORAM simulation?
Suppression Compiler

\[ \Lambda = (\mathcal{L}_S, \mathcal{L}_Q) \]
\[ = (\star, (\text{patt}_1, \text{patt}_2)) \]

\[ \Lambda = (\mathcal{L}_S, \mathcal{L}_Q) \]
\[ = (\star, \text{patt}_2) \]
Suppression Compiler for Query Equality

\[ \Lambda = (\mathcal{L}_S, \mathcal{L}_Q) = (\star, \text{qe}q) \]

\[ \Lambda = (\mathcal{L}_S, \mathcal{L}_Q) = (\star, 
\]
Q: Can we build such a thing?
Suppression Compiler for Query Equality

\[ \Lambda = (\mathcal{L}_s, \mathcal{L}_q) \]
\[ = (\star, (qeq, patt)) \]

\[ \Lambda = (\mathcal{L}_s, \mathcal{L}_q) \]
\[ = (\star, \text{nrp}) \]

nrp is the *non-repeating sub-pattern* of patt
Non-Repeating Sub-Patterns

• Leakage patterns can be decomposed into sub-patterns:

\[
patt = \begin{cases} 
patt_1 & \text{if "condition" is true} 
patt_2 & \text{otherwise.}
\end{cases}
\]

• Non-repeating sub-patterns \( \approx \) leakage on non-repeating queries

\[
patt = \begin{cases} 
\text{nrm} & \text{if queries are unique} 
\text{misc} & \text{otherwise.}
\end{cases}
\]
Suppression Compiler for Query Equality

\[ \Lambda = (\mathcal{L}_S, \mathcal{L}_Q) \]
\[ = (\star, (\text{qeq}, \text{patt})) \]
\[ \text{patt} = \begin{cases} 
\text{nrp} & \text{if queries are unique} \\
\text{misc} & \text{otherwise.} 
\end{cases} \]
Cache-based Compiler and Rebuilding

- Cache-based Compiler
  - needs to rebuild encrypted structure from time to time
- So base STE scheme has to have a Rebuild protocol
- Rebuild protocol must
  - be efficient for server
  - have $O(1)$ client storage
  - be zero-leakage
Our Suppression Pipeline

[K.-Moataz-Ohrimenko18]

\[ \Lambda = (\mathcal{L}_S, \mathcal{L}_Q, \mathcal{L}_A) \]
\[ = (\star, (\text{eq}, \text{patt}), \text{patt}_A) \]

\[ \text{patt} = \begin{cases} 
\text{srlen} & \text{if queries are unique} \\
\text{misc} & \text{otherwise.} 
\end{cases} \]
Q: How does the CBC work?
Square Root ORAM

[Goldreich-Ostrovsky92]

**Setup time**

ORAM.Setup → Main Memory + Dummies → Cache

**Query time**

if item in cache
get dummy
else
get item

Leaks qeq but query never repeated

Zero-leakage
Reinterpreting Square Root ORAM

[K.-Moataz-Ohrimenko18]

**Setup time**

ORAM.Setup

Main Memory + Dummies

Cache

\[ \Lambda_{\text{ERAM}} = (\mathcal{L}_S, \mathcal{L}_Q) \]

\[ = (\star, \text{qeq}) \]

\[ \Lambda_{\text{EDX}} = (\mathcal{L}_S, \mathcal{L}_Q) \]

\[ = (\star, \bot) \]

**Query time**

Main Memory + Dummies

Cache

if item in cache
get dummy
else
get item

\[ \Lambda_{\text{ERAM}} = (\mathcal{L}_S, \mathcal{L}_Q) \]

\[ = (\star, \text{qeq}) \]

\[ \Lambda_{\text{EDX}} = (\mathcal{L}_S, \mathcal{L}_Q) \]

\[ = (\star, \bot) \]
Reinterpreting Square Root ORAM

• Square root ORAM \approx
  • “uses a ZL encrypted dictionary…
  • …to suppress the qeq leakage of an encrypted RAM”

• **Q**: Can we replace the ERAM with another encrypted structure?
  • if yes then no multiplicative **polylog** overhead due to simulation
The Cache-Based Compiler

Setup time

ORAM.Setup

if item in cache
get dummy
else
get item

Query time

Main Memory + Dummies

\( \Lambda_{EDS} = (\mathcal{S}, \mathcal{Q}) \)

\( \Lambda_{EDS} = (\star, \text{qeq}) \)

\( \Lambda_{EDX} = (\mathcal{S}, \mathcal{Q}) \)

\( \Lambda_{EDX} = (\star, \bot) \)

Cache

Main Memory + Dummies

Cache

if item in cache
get dummy
else
get item
The Cache-Based Compiler

• EDS has to satisfy certain properties
  • has to be rebuildable
  • has to be “extendable” ≈ can store dummies
  • has to be “safe” ≈ handles dummies securely

\[ \mathcal{D}_S(\overline{DS}) \leq \mathcal{D}_S(DS) \quad \mathcal{D}_Q(\overline{DS},q) \leq \mathcal{D}_S(DS,q) \]

• has to have “small” non-repeating sub-pattern
The Piggy Back Scheme (PBS)
[K.-Moataz-Ohrimenko18]

• Data structure transformation
  • pad tuples to multiple of $\alpha$ (e.g., $\alpha = 3$)
The Piggy Back Scheme (PBS)

- PBS.Setup(1k, EMM = DX)
  - creates client state that maps labels to number of blocks
  - sends encrypted dictionary EDX to server
Consider sequence \((\ell_1, \ell_3, \ell_2, \ldots)\)

- PBS.Get\((K, \text{state, } Q, \ell_1)\)
  - \(2 := DX[\ell_1]\)
  - Enqueue \(\ell_1|1\) and \(\ell_1|2\) on \(Q\)
  - query := \(Q\).dequeue()
  - send \(EDX.Token(K, \text{query})\)
  - client only gets back

- PBS.Get\((K, \text{state, } Q, \ell_3)\)
  - ...
  - client gets back

The Piggy Back Scheme (PBS)
The Piggy Back Scheme (PBS)

- PBS leverages a **new tradeoff**
  - *security vs. latency*
  - hides response length (volume) but response not immediate
- PBS has leakage profile
  - $\Lambda = (\mathcal{L}_S, \mathcal{L}_Q) = (\star, \text{rqeq}, \star)$
    - where *rqeq* has non-repeating sub-pattern
      - $\perp$ on all but the last query
      - *sr*len on the last query
Thm: If queries and responses are Zipf distributed then under the inverted query hypothesis, latency is $t + \varepsilon \cdot t$ with probability at least

$$1 - \exp \left( -2t \left( \varepsilon \cdot \frac{\alpha}{\mu} \right)^2 \right)$$
Our Suppression Pipeline

[K.-Moataz-Ohrimenko18]

\[ \Lambda = (\mathcal{S}, \mathcal{Q}, \mathcal{A}) \]

\[ = (\star, (\text{qe}, \text{patt}), \text{patt}_A) \]

\[
\begin{align*}
\text{patt} &= \begin{cases} 
\text{srlen} & \text{if queries are unique} \\
\text{misc} & \text{otherwise.}
\end{cases}
\end{align*}
\]
Q: Can we suppress other patterns besides the query equality?
The Volume Pattern

- Volume pattern is the size of a response
  - very common leakage pattern (even ORAM leaks it)
  - hard to suppress without blowup in storage
- [Kellaris-Kollios-Nissim-O’Neill16,…]
  - series of attacks vs. volume pattern of range queries
Suppressing Volume with Naive Padding

- Query complexity \( O\left( \max_{\ell \in \mathbb{L}_{\text{MM}}} \#\text{MM}[\ell] \right) \)

- Storage complexity \( O\left( \#\mathbb{L}_{\text{MM}} \cdot \max_{\ell \in \mathbb{L}_{\text{MM}}} \#\text{MM}[\ell] \right) \)

Can we do better?
Computationally-Secure Leakage

Unbounded Adversary vs. Bounded Adversary
Pseudo-Random Transform (PRT)

• Let $F: \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^\log \mu$ be a PRF
• Let $\lambda \geq 0$ be a parameter (min. response length)
• For each label $\ell$ in $\mathcal{MM}$
  • compute $\text{len}(\ell) = \lambda + F_K(\ell | \#\mathcal{MM}[\ell])$
  • if $\text{len}(\ell) < \#\mathcal{MM}[\ell]$ truncate $\ell$’s tuple to length $\text{len}(\ell)$
  • if $\text{len}(\ell) > \#\mathcal{MM}[\ell]$ pad $\ell$’s tuple to length $\text{len}(\ell)$
Pseudo-Random Transform (PRT)

• Example with $\lambda = 1$ and $\mu = 3$

\[\lambda + F_K(\ell_1 | 4) = 1 + 0 = 1\]
\[\lambda + F_K(\ell_2 | 2) = 1 + 2 = 3\]
\[\lambda + F_K(\ell_3 | 3) = 1 + 1 = 1\]
Pseudo-Random Transform (PRT)

• PRT is a “lossy” transformation
• PRT exploits a new tradeoff
  • lossiness vs. security
• Volume hiding relies on pseudo-randomness of F
• Need to analyze
  • Number of truncations
  • Storage overhead
Zipf-Distributed Multi-Maps

- A MM is Zipf-distributed if the rth tuple has length

$$\frac{1}{r \cdot H_{\#\mathbb{I}_{MM},1}} \cdot \sum_{\ell \in \mathbb{I}_{MM}} \#\mathbb{I}_{MM}[\ell]$$

Response length

Number of labels

Enron dataset
Thm: Let $\frac{1}{2} < \alpha < 1$. If $MM$ is Zipf-distributed, then $MM'$ has size at most

$$\alpha \cdot \#L \cdot \max_{\ell \in L} \#MM[\ell]$$

with probability at least $1 - \exp \left( -\#L \cdot (2\alpha - 1)^2 / 8 \right)$.

Furthermore, it incurs at most

$$\frac{1}{\log(\#L)} \cdot \#L$$

truncations with probability at least $1 - \exp \left( -2 \cdot \#L \cdot \log^2(\#L) \right)$. 
Pseudo-Random Transform (PRT)

- PRT has many advantages
  - easy to use and implement 😊
  - doesn’t impact query and storage complexity too much 😊

- But it is is lossy 😞
  - for keyword search one can rank results
  - so only low-ranked results are lost
Q: Can we design a non-lossy transformation?
Densest Subgraph Transform

[K.-Moataz19]

- Data structure transformation
  - hides volume 😊
  - query complexity ≈ query complexity of naive padding 😞
  - storage complexity ≤ storage complexity of naive padding 😏
  - non-lossy 😊

- How is this possible?
  - New EMM design framework
  - Computational assumptions from average-case complexity
Densest Subgraph Transform

[\text{K.-Moataz19}]

- For each label pick $\mu = \max_\ell \# \text{MM}[\ell]$ bins at random
  - store values in bins
  - if $\# \text{MM}[\ell] < \mu$ don’t store anything in remaining bins
- Pad all bins
Densest Subgraph Transform

[K.-Moataz19]

• Compressing the state
  • instead of choosing edges/bins uniformly at random
  • use a PRF and store key/rand value in state

Client state

\[ \ell_1 \rightarrow B_1 B_3 B_4 \]

\[ \ell_2 \rightarrow B_2 B_3 B_4 \]

\[ \ell_3 \rightarrow B_1 B_2 B_3 \]

\( O(\#L \cdot \max_{\ell \in L} \#MM[\ell]) \)

VS.

Client state

\[ \ell_1 \rightarrow \text{rand}_1 \]

\[ \ell_2 \rightarrow \text{rand}_2 \]

\[ \ell_3 \rightarrow \text{rand}_3 \]

\( O(\#L) \)

Some PRF seeds can lead to collisions so just pick again until no collisions
Densest Subgraph Transform

[K.-Moataz19]

- Store bins in a dictionary DX and encrypt DX
Densest Subgraph Transform

[Moataz19]

• To get $\ell_2$,
  • retrieve $\text{rand}_2$ from state
  • compute bin identifiers
    • $2 := F(\text{rand}_2, 1)$,
    • $3 := F(\text{rand}_2, 2)$,
    • $4 := F(\text{rand}_3, 3)$
  • retrieve bins
**Thm:** The load of a bin is at most

\[
\frac{N}{n} + \frac{\ln(1/\varepsilon)}{3} \left(1 + \sqrt{1 + \frac{18N}{n \cdot \ln(1/\varepsilon)}}\right)
\]

with probability at least 1 - \(\varepsilon\), where

\[
N = \sum_{\ell \in \text{MM}} \#\text{MM}[\ell]
\]
Densest Subgraph Transform

[K.-Moataz19]

- Alternative construction for concentrated MMs
  - $\mathbf{v}_2$ and $\mathbf{v}_4$ are duplicated so store them only once
  - Pick bi-partite clique at random
    - store duplicated items in clique
  - Pick remaining edges at random
Densest Subgraph Transform

[K.-Moataz19]

**Thm:** The load of a bin is at most

\[ \frac{N - N_{DS}}{n} + \frac{\ln(1/\varepsilon)}{3} \left( 1 + \sqrt{1 + \frac{18(N - N_{DS})}{n \cdot \ln(1/\varepsilon)}} \right) \]

with probability at least 1 - \( \varepsilon \), where \( N_{DS} \) is the size of concentrated part
Densest Subgraph Assumption

[Applebaum-Barak-Wigderson10]

Erdös-Rényi graph

Erdös-Rényi graph with planted dense subgraph
Densest Subgraph Assumption

[Applebaum-Barak-Wigderson10]

- Variant of the planted clique problem
  - central problem in average-case hardness
- Evidence for hardness
  - studied since the mid-70’s in CS & statistical physics
  - failure of powerful algorithmic techniques
  - restricted lower bounds
    - Sum-of-squares
    - Statistical query
Conclusions
Encrypted Search: 2000-2019

• A large and vibrant area of research
• Many interesting and hard problems
• Many fundamental questions
  • how do we model leakage?
  • how do we quantify leakage?
  • how do we suppress leakage?
• are the tradeoffs we observe inherent? (i.e., lower bounds)
Encrypted Search: 2000-2019

• Many connections
  • algorithms & data structures
  • database theory & systems
  • statistical learning theory
  • optimization
  • graph theory
  • distributed systems
Encrypted Search: 2000-2019

- Many interesting leakage attacks to study
- But many new techniques to bypass leakage attacks
  - padding & clustering techniques [Bost-Fouque17]
  - response-hiding schemes [Blackstone-K.-Moataz19]
  - suppression compilers [K.-Moataz-Ohrimenko18]
  - suppression transforms [K.-Moataz19]
  - worst-case vs. average-case leakage [Agarwal-K.19]
  - distributing data [Agarwal-K.19]
Encrypted Search: 2000-2019

• New tradeoffs to explore
  • leakage vs. correctness \([\text{K.-Moataz19}]\)
  • leakage vs. latency \([\text{K.-Moataz-Ohrimenko18}]\)
Encrypted Search: 2000-2019

• Real-world impact
  • Microsoft SQL Server
  • MongoDB Field Level Encryption
  • Cisco WebEx
  • Ionic
  • more coming…
Thanks to…

Archita Agarwal  Ghaus Amjad  Hajar Alturki  Laura Blackstone

Marilyn George  Tarik Moataz  Olya Ohrimenko  Sam Zhao
The End