Encrypted Search: Intro & Basics

Seny Kamara
14,717,618,286*

4%

* since 2013
Why so Few?

"...because it would have hurt Yahoo’s ability to index and search message data..."

– J. Bonforte in NY Times
Q: can we search on encrypted data?
Interdisciplinary
Real-World Problem

• Major companies
  • Microsoft, SAP
  • MongoDB, Cisco
  • Google Research
  • Hitachi, Fujitsu
  • more…

• Funding agencies
  • NSF
  • IARPA
  • DARPA

• Startups
  • Ciphercloud
  • Skyhigh Networks
  • Bitglass
  • Baffle
  • Cossack Labs
  • Strong Salt, Overnest
  • many many more
Encrypted Search (Building Blocks)

- Property-Preserving Encryption (PPE)
- Fully-Homomorphic Encryption (FHE)
- Functional Encryption
- Oblivious RAM (ORAM)
- Structured Encryption (STE)
Efficiency

Functionality

Leakage
What is Search?

• Complexity regimes
  • linear search: \(O(n)\)
  • sub-linear search: \(o(n)\)
• Algorithmic paradigms
  • with pre-processing
  • without pre-processing
• For medium to large data
  • sub-linear search is a *requirement*; not an option

<table>
<thead>
<tr>
<th></th>
<th>Without Pre-Processing</th>
<th>With Pre-Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>sequential scan</td>
<td>not interesting</td>
</tr>
<tr>
<td>Sub-Linear</td>
<td>read sub-set of input</td>
<td>data structures</td>
</tr>
<tr>
<td></td>
<td>(errors)</td>
<td></td>
</tr>
</tbody>
</table>
Background: Data Structures

- Abstract data types
  - capture functionality
  - ex: dictionary
- Data structures
  - instantiate ADTs
  - ex: hash table, binary search tree
- As common in CS
  - we sometimes blur the distinction

- Arrays store values
  - Write: \( A[i] := v_i \)
  - Read: \( A[i] \) returns \( v_i \)
Background: Data Structures

• Dictionaries map labels to values
  
  **DX**
  
  \[
  \begin{array}{ccc}
  \ell_1 & \rightarrow & v_1 \\
  \ell_2 & \rightarrow & v_2 \\
  \ell_3 & \rightarrow & v_3 \\
  \end{array}
  \]

  • Put: \( DX[\ell_2] := v_2 \)
  • Get: \( DX[\ell_2] \) returns \( v_2 \)

• Multi-Maps map labels to tuples
  
  **MM**
  
  \[
  \begin{array}{ccc}
  \ell_1 & \rightarrow & v_1 \rightarrow v_3 \rightarrow v_4 \\
  \ell_2 & \rightarrow & v_3 \\
  \ell_3 & \rightarrow & v_2 \rightarrow v_4 \\
  \end{array}
  \]

  • Put: \( MM[\ell_3] := (v_2,v_4) \)
  • Get: \( MM[\ell_3] \) returns \( (v_2,v_4) \)
Keyword Search in Sub-Linear Time

Setup time

Query time

\[ \text{ans} = (\text{ptr}_1, \ldots, \text{ptr}_n) \]
Database Queries in Sub-Linear Time

Setup time

Query time

\[
\text{ans} = (\text{ptr}_1, \ldots, \text{ptr}_n)
\]
Q: how do we do sub-linear search on encrypted data?
Encrypted Keyword Search in Sub-Linear Time

Setup time

Query time

\[ \text{ans} = (\text{ptr}_1, ..., \text{ptr}_n) \]
**Encrypted Database Queries in Sub-Linear Time**

**Setup time**

```
[Diagram showing setup process]
```

**Query time**

```
ans = (ptr₁, ..., ptrₙ)
```

```
[Diagram showing query process]
```
Q: how do we formalize *encrypted* data structures?
Structured Encryption
[Chase-K.10]

Setup$(1^k, \text{DS}) \rightarrow (K, \text{EDS})$

Token$(K, q) \rightarrow tk$

Query$(\text{EDS, tk}) \rightarrow \text{ans}$
Desiderata

Size of EDS

Setup leakage

EDS

Query leakage

Size of state

Query time

Size of token

ans
Structured Encryption

[Chase-K.10]

- Many variants of STE
  - response-revealing
    - EDS query reveals answer in plaintext
  - response-hiding
    - EDS query reveals encrypted answer
  - non-interactive queries
    - clients send single message called a token
  - interactive queries
    - client and server execute multi-round protocol
Evolution of Structured Encryption

**Efficiency**
- ‘00: Linear in file length [SWP00]
- ‘03: Linear in #docs [Goh03]
- ‘06: Optimal [CGKO06,CK10]
- ‘12: Optimal Dynamic [KPR12,CJJJKRS14]
- ‘14: I/O efficient [CT14,CJJJKRS14,ANSS16,DP18,ASS18]

**Expressiveness**
- ‘00: Single-keyword SSE [SWP00,Goh03,CGKO06,CJJJKRS14]
- ‘06: Multi-user SSE [CGKO06,JJKRS13,PPY16,HSWW18]
- ‘13: Boolean SSE [CJJJKRS13,PKVK+14,KM17]
- ‘14: Range SSE [PKVK+14,FJKNRS15]
- ‘18: STE-based SQL [KM18]

**Security**
- ‘06: Leakage-parametrized security definitions [CGKO06]
- ‘12: Attacks [IKK12,CGPR15,ZKP16,KMNO16,LMP18,GLMP18]
- ‘14: Forward/Backward Security [SPS14,Bost16,LC17,BMO17,AKM18]
- ‘18: Leakage Supression [KMO18,KM19]
- ‘19: Snapshot [AKM18]
Adversarial Models
Adversarial Models

Persistent

Snapshot

View

EDS₀

q

ans

c

EDS₀

q

ans

c

EDS₀

q

EDS₁

EDS₂
Persistent (Adaptive) Security

[Curtmola-Garay-K.-Ostrovsky06,Chase-K.10]

• An STE scheme is \((\mathcal{P}_S, \mathcal{P}_Q)\)-secure vs. a persistent adv. if
  • it reveals no information about the \textit{structure} beyond \(\mathcal{P}_S\)
  • it reveals no information about the \textit{structure} and \textit{query} beyond \(\mathcal{P}_Q\)
Persistent (Adaptive) Security
[Curtmola-Garay-K.-Ostrovsky06,Chase-K.10]
Forward Privacy
[Stefanov-Papamanthou-Shi14, Bost16]

• Informally [SPS14]
  • “Updates not correlated to previous queries”
• Formally [Bost16]
  • $\mathcal{U}(\text{MM}, (\ell, \mathbf{v})) = \#\mathbf{v}$
We say that an STE scheme is $\mathcal{L}_{\text{Snp}}$-secure vs. a snapshot adv. if it reveals no information about the *structure* beyond $\mathcal{L}_{\text{Snp}}$. 
Snapshot (Adaptive) Security

[Amjad-K.-Moataz19]

Real

\[ D_{S_0} \]

\[ E_{D_{S_0}} \]

\[ q \]

\[ u \]

\[ E_{D_{S_1}} \]

\[ E_{D_{S_2}} \]

Ideal

\[ D_{S_0} \]

\[ L_S(D_{S_0}) \]

\[ E_{D_{S_0}} \]

\[ q \]

\[ L_S(D_{S_1}, q) \]

\[ E_{D_{S_1}} \]

\[ q \]

\[ L_S(D_{S_2}, q) \]

\[ E_{D_{S_2}} \]
Snapshot (Adaptive) Security

[Amjad-K.-Moataz19]

Static Structures

\[ L_{\text{Snp}} = L_S \]

Dynamic Structures

- Forward privacy
- Snapshot security
- Insertion independence (variant of history independence)
- Write-only obliviousness
Q: Why do we parameterize definitions with leakage?
Leakage-Parameterized Definitions
[Curtmola-Garay-K.-Ostrovsky, Chase-K.10]

• This area is about tradeoffs
  • but traditional cryptographic definitions don’t capture tradeoffs
• in 00’s, different approaches were proposed to capture leakage
  • #1: limit adversary’s power in the proof
  • #2: make assumptions on data (e.g., high entropy)
• Original motivations for leakage-parameterized definitions
  • Approaches #1 & #2 are misleading (sweep leakage under the rug)
  • Leakage should be made explicit and not be implicit
    • gives clear target for cryptanalysis
    • makes it (somewhat) easier to compare schemes
Q: How do we model leakage?
Modeling Leakage

• Each scheme has a leakage profile: $\Lambda = (\mathcal{L}_S, \mathcal{L}_Q, \mathcal{L}_U)$
  • where $\mathcal{L}_S = (\text{patt}_1, \ldots, \text{patt}_n)$ is the Setup leakage
  • $\mathcal{L}_Q = (\text{patt}_1, \ldots, \text{patt}_n)$ is the Query leakage
  • $\mathcal{L}_U = (\text{patt}_1, \ldots, \text{patt}_n)$ is the Update leakage
• Each “operational” leakage is composed of leakage patterns
  • $(\text{patt}_1, \ldots, \text{patt}_n)$
Common Leakage Patterns
[K.-Moataz-Ohrimenko18]

- **qeq**: query equality
  - a.k.a. search pattern
- **rid**: response identity
  - a.k.a. access pattern
- **qlen**: query length
- **trlen**: total resp. length
- **rlen/vol**: response length
  - a.k.a. volume pattern
- **req**: response equality
- **mqlen**: max query length
- **mrlen**: max resp. length
- **srlen**: sequence resp. length
- **dsize**: data size
- **usize**: update size
- **did**: data identity
Example Leakage Profiles

• The “Baseline” leakage profile for response-revealing EMMs
  • $\mathbf{\Lambda} = (L_S, L_Q, L_U) = (\text{dsize}, (\text{qeq, rid}), \text{usize})$

• The “Baseline” leakage profile for response-hiding EMMs
  • $\mathbf{\Lambda} = (L_S, L_Q, L_U) = (\text{dsize, qeq, usize})$

• Several new constructions have better leakage profiles
  • AZL and FZL [K.-Moataz-Ohrimenko18]
  • VLH and AVLH [K.-Moataz19]
Structured Encryption vs. Other Primitives

- Encrypted structures appear implicitly throughout crypto
- Oblivious RAM can be viewed as a
  - response-hiding encrypted array
  - with leakage profile $\Delta_{\text{ORAM}} = (s, q, u) = (\text{dsiz}, \bot)$
- PIR can be viewed as a
  - response-hiding encrypted array
  - with leakage profile $\Delta_{\text{PIR}} = (s, q, u) = (\text{did}, \bot)$
- Garbled gates can be viewed as
  - response-revealing 2x2 arrays
  - $\Delta_{\text{GG}} = (s, q, u) = (\text{dsiz}, \text{qeq})$
Encrypted Multi-Maps
Encrypted Multi-Maps:
The Heart of *Sub-Linear* Encrypted Search

- EMMs are used as building block for sub-linear
  - Single keyword search [Curtmola-Garay-K.-Ostrovsky06,…]
  - Conjunctive keyword search [Cash et al.13,…]
  - Boolean keyword search [Cash et al.13, K.-Moataz17,…]
  - Range queries [Faber et al.14, Demertzis et al. 16,…]
  - Substring, wildcard, [Faber et al.14,…]
  - SQL databases [K.-Moataz18,…]
  - Graph databases [Chase-K.10,…]
Pidyn (Modified)

[Cash et al. 14]

EMM.Setup\(l, k\),

\[ K_{\ell_i} = F_K(w_i | 1) \]
Pidyn (Modified)
[Cash et al.14]

\[
\begin{align*}
\text{EMM.Get} & \quad , \quad K_{\ell_1} \\
\text{Hist. Ind. DX} & \quad \begin{cases}
F(K_{\ell_1}, 1) & \rightarrow V_1 \\
F(K_{\ell_1}, 2) & \rightarrow V_3 \\
F(K_{\ell_1}, 3) & \rightarrow V_4 \\
F(K_{\ell_2}, 1) & \rightarrow V_3 \\
F(K_{\ell_3}, 1) & \rightarrow V_2 \\
F(K_{\ell_3}, 2) & \rightarrow V_4 \\
\end{cases}
\end{align*}
\]

\[
\begin{align*}
1. \text{DX.Get} & \quad \begin{cases}
\text{DX} & , \quad F(K_{\ell_1}, 1) \\
\text{DX} & , \quad F(K_{\ell_1}, 2) \\
\text{DX} & , \quad F(K_{\ell_1}, 3) \\
\text{DX} & , \quad F(K_{\ell_1}, 3) \\
\end{cases} \\
\quad \downarrow \\
\end{align*}
\]
Pidyn (Modified)  
[Cash et al. 14]
Pidyn (Modified)  
[Cash et al. 14]
Pidyn (Modified)
[Cash et al. 14]

\[ K_{\ell_1} = F_K(\ell_1 | 1) = K_{\ell_1} \]

Query complexity:
\[ O(#MM[\ell] + dels_0(\ell)) \]

Storage complexity:
\[ O(\sum_{\ell} #MM[\ell] + dels_0(\ell)) \]
I/O Efficiency & Locality

[Cash et al.14]

• The problem with large data
  • if data is very large it gets stored on disk
• Disk seeks are very slow
  • minimize locality: # of non-contiguous accesses
  • minimize read efficiency: how much additional data is read
    • reading contiguous data is OK but not too long
• Pidyn has poor locality
  • $\text{Get}(\ell)$ needs $\#MM[\ell]$ non-contiguous accesses
I/O Efficiency & Locality

[Cash et al. 14]

• Introduce several schemes with improved locality
  • Pipack: packs values in a single ciphertext
  • Piptr: packs pointers to values in a single ciphertext
    • this tradeoffs EMM locality for standard memory locality
  • 2Lev: combines both techniques
Local SSE Schemes

- [Cash-Tessaro14]
  - lower bounds for “non-overlapping” schemes (improved by Asharov et al.)
- [Asharov-Segev-Shahaf18]
  - lower bound for “pad-and-split” schemes
  - $L(N)$ locality & $O(1)$ read efficiency $\implies \Omega(N \log N / \log L)$ space
  - matched by [Demertzis-Papamanthou17]
- [Asharov-Naor-Segev-Shahaf18]
  - lower bound for “statistically-ind.” schemes
  - $O(1)$ locality & $O(N)$ space $\implies \omega(1) \cdot \epsilon(n)^{-1}$ read efficiency
  - matched by [Asharov-Segev-Shahaf18]
Limitations of **Pidyn, Pipack, Piptr, 2Lev**

- Not forward private
  - update tokens can be linked to previous search tokens
  - can be exploited using adaptive file injection attacks
- Query and storage complexity depend on total # of deletes
State-of-the-Art EMMs

<table>
<thead>
<tr>
<th></th>
<th>Search</th>
<th>Client Storage</th>
<th>Forward Privacy</th>
<th>Snapshot</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPS’14</td>
<td>$O(#\text{MM}[\ell] \cdot \text{polylog}(#\text{MM}[\ell]))$</td>
<td>$O(#\text{L})$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>B’16</td>
<td>$O(#\text{MM}[\ell] + \text{dels}_0(w))$</td>
<td>$O(#\text{L})$</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>BMO’17</td>
<td>$O(#\text{MM}[\ell] + \text{dels}_0(w))$</td>
<td>$O(#\text{L})$</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>EKPE’17</td>
<td>$O(#\text{MM}[\ell] + \text{dels}_s(w))$</td>
<td>$O(#\text{L})$</td>
<td>Yes for adds</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No for dels</td>
<td></td>
</tr>
<tr>
<td>AKM19</td>
<td>$O(#\text{MM}[\ell] + \text{dels}_r(w))$</td>
<td>$O(#\text{L} + \text{ML})$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
[AKM19] Client State

- EDB w/ 83 million pairs (11GB)
- state is 210MB
Single Keyword Search from EMMs

K

DX (state)
Sub-Linear Constructions from Black-Box EMMs

• Searchable symmetric encryption [Curtmola-Garay-K.-Ostrovsky06, …]
• Graph queries [Chase-K.10, …]
• Conjunctive & disjunctive keyword search [Cash et al. 13,]
• Worst-case sub-linear disjunctive & Boolean search [Pappas et al. 14, K.-Moataz17]
• Wildcard & substring search [Faber et al. 15]
• Range search [Faber et al. 15, Demertzis et al. 16, Podar-Boelter-Popa16]
• SQL queries on relational DBs [K.-Moataz18]
Sub-Linear Constructions from Black-Box EMMs

• Why constructions based on black-box EMMs?
• Modularity
  • easy to design, understand and analyze
  • benefit from improvements in EMM efficiency
  • benefit from improvements in EMM security/leakage