Encrypted Search

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

Data Structures

Cryptography

Databases

Graph Algorithms

Information Retrieval

Statistics

Optimization

Distributed Systems
Real-World Problem

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

• Funding agencies
  • NSF
  • IARPA
  • DARPA

• Startups
  • too many to list
Q: what about real-world customers?
Is this Real?

- Banks
- Government agencies (US & Europe)
- Fintech companies
- Tech companies
- Healthcare
- Biotech
- …
Encrypted Search
Encrypted Search

- Sub-field focused on designing
  - *sub-linear* algorithms over encrypted data
  - search engines & databases

- Searchable (symmetric) encryption (SSE)
  - keyword search over collection of encrypted files/documents
  - ElasticSearch, Lucene, …

- Encrypted databases (EDBs)
  - encrypted NoSQL & SQL (relational) databases
  - Postgres, SQL Server, MongoDB, CouchDB, …
Encrypted Search (Building Blocks)

Very leaky

$\Omega(n)$

$O(n)$

Structured Encryption (STE)

$\Omega(n)$

Oblivious RAM (ORAM)
Core Primitive: **Structured Encryption**

- Schemes that
  - encrypt data structures (e.g., multi-maps, dictionaries, ...)
  - support private queries on encrypted structures

- Applications
  - sub-linear searchable encryption (i.e., index-based SSE)
  - encrypted NoSQL & SQL databases
  - encrypted graph algorithms
  - secure multi-party computation
Structured Encryption

[Chase-K.10]

Setup(1^k, DS) → (K, EDS)

Token(K, q) → tk

Query(EDS, tk) → ans
Desiderata

Size of EDS → EDS

tk

ans

Size of state

Size of token

Setup leakage

Query time

Query leakage
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 sends single message called a token
• interactive queries
  • client and server execute multi-round protocol
Background: Data Structures

- Dictionaries map labels to values
  - Put: $\text{DX}[\ell_2] := v_2$
  - Get: $\text{DX}[\ell_2]$ returns $v_2$

- Multi-Maps map labels to tuples
  - Put: $\text{MM}[\ell_3] := (v_2,v_4)$
  - Get: $\text{MM}[\ell_3]$ returns $(v_2,v_4)$
Structured Encryption: Encrypted Dictionary

Setup($k$, DS) $\rightarrow$ (K, EDX)

Token(K, q) $\rightarrow$ tk

Query(EDX, tk) $\rightarrow$ ans
Structured Encryption: Encrypted Multi-Map

[Chase-K.10]

Setup($1^k$, DS) $\rightarrow$ (K, EMM)

Token(K, q) $\rightarrow$ tk

Query(EMM, tk) $\rightarrow$ ans
Adversarial Models

Persistent

View

Snapshot
Persistent (Adaptive) Security
[Curtmola-Garay-K.-Ostrovsky06,Chase-K.10]

• An STE scheme is ($\mathcal{L}_S$, $\mathcal{L}_Q$)-secure vs. a persistent adv. if
  • it reveals no information about the structure beyond $\mathcal{L}_S$
  • it reveals no information about the structure and query beyond $\mathcal{L}_Q$
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 $\mathcal{L}_{\text{Snp}}$. 
Efficiency vs. **Persistent** Security

Query Time

- Fully-homomorphic
- Functional (sk)
- Functional (pk)
- Oblivious
- Structured
- Property-preserving

*Not Scientific!*
Efficiency vs. **Snapshot** Security

- Fully-homomorphic
- Functional (sk)
- Functional (pk)
- Oblivious
- Structured
- Property-preserving

**Not Scientific!**
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
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

- **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
  • \( \Lambda = (\mathcal{L}_S, \mathcal{L}_Q, \mathcal{L}_U) = (\text{dsize}, (\text{qeq}, \text{rid}), \text{usize}) \)

• The “Baseline” leakage profile for response-hiding EMMs
  • \( \Lambda = (\mathcal{L}_S, \mathcal{L}_Q, \mathcal{L}_U) = (\text{dsize}, \text{qeq}, \text{usize}) \)

• 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 $\Delta_{\text{ORAM}} = (L_S, L_Q, L_U) = (\text{dsize, vol, vol})$

- Garbled gates can be viewed as
  - response-revealing 2x2 arrays
  - $\Delta_{\text{GG}} = (L_S, L_Q) = (\text{dsize, qeq})$
How do we Deal with Leakage?

• Our definitions allow us to prove that our schemes
  • achieve a certain leakage profile
  • but doesn’t tell us if a leakage profile is exploitable?

• We need more than proofs
The Methodology

- **Leakage analysis**: what is being leaked?
- **Proof**: prove that scheme leaks no more
- **Cryptanalysis**: can we exploit this leakage?
Leakage Attacks
Leakage Attacks

- **Target**
  - *query recovery*: recovers information about query
  - *data recovery*: recovers information about data

- **Adversarial model**
  - *persistent*: needs EDS and tokens
  - *snapshot*: needs EDS

- **Auxiliary information**
  - *known sample*: needs sample from same distribution
  - *known data*: needs actual data

- **Passive vs. active**
  - *injection*: needs to inject data
Leakage Attacks

• Leakage cryptanalysis is crucial but...

• ...unfortunately much of the attack literature
  • lacks experimental rigor
  • is just plain wrong
  • overhyped

• there is a need for higher standards
Leakage Attacks

• IKK attack
  • highly cited but doesn’t work
  • too few keywords, auxiliary & test data correlated, ...

• Count attack
  • based on strong assumptions
  • adversary needs to know $\geq 75\%$ of client’s data!

• Some target very niche applications & rely on strong assumptions
Leakage Attacks

• Should we discount attacks? Of course not
  • More rigorous
  • Less hyperbolic
  • More upfront about attack limitations & assumptions

• [Blackstone-K.-Moataz’20]: Revisiting Leakage-Abuse Attacks

• [KKMSTY’21]: re-implementation & re-evaluation of most known attacks
How Should we Handle Leakage?

- **Approach #1**: ORAM simulation
  - Store and simulate data structure with ORAM
  - polylog overhead per read/write on top of simulation
  - still leaks information that is exploitable
    - [Kellaris-Kollios-O’neill-Nissim’16, Blackstone-K.-Moataz’20]

- **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?
    • Yes [K.-Moataz-Ohrimenko’18, George-K.-Moataz’21]

• **Approach #4**: Leakage suppression
  • Suppression compilers
  • Suppression transforms
Leakage Suppression

• Techniques to reduce/eliminate leakage

• Suppressing query equality (aka access pattern)
  • general compiler [K.-Moataz-Ohrimenko’18, Geoge-K.-Moataz’21]

• Suppressing co-occurrence (needed by IKK and Count attacks)
  • see appendix in [Blackstone-K.-Moataz19]
Leakage Suppression

• Suppressing volume (aka response size)
  • padding & clustering techniques [Bost-Fouque17]
  • computational techniques [K.-Moataz19, Patel-Persiano-Yeo-Yung’20]

• “General-purpose” suppression
  • worst-case vs. average-case leakage [Agarwal-K.1’9]
  • distributing data [Agarwal-K.’19]
Leakage Suppression

• New tradeoffs to explore
  • leakage vs. correctness [K.-Moataz19]
  • leakage vs. latency [K.-Moataz-Ohrimenko18]
Thanks!