Cloud Security & Cryptography II

Cloud-oriented Primitives

SENY KAMARA
MICROSOFT RESEARCH
Cloud Cryptography

- Current crypto tools are inappropriate for the cloud
  - Due to assumptions about how tools will be used
  - Results in efficiency loss & insecurity
- We need new tools!
Cloud Cryptography

- Searchable encryption
  - Searching over encrypted data
- Structured encryption
  - Querying encrypted structured data
- Private information retrieval
  - Downloading data privately
- Oblivious RAM
  - Computing privately with untrusted memory
- Proofs of storage
  - Verifying integrity of outsourced data
Searchable Encryption
Cloud Storage
Two Simple Solutions to Search

Large comm. complexity

Q: can we achieve the best of both?
Searchable Symm. Encryption

\[ \text{Enc}_K \]

\[ t_w \]

\[ \text{Enc}_K \]
SSE Parameters

Parameters
- \( n \): number of files in collection
- \(|f|\): size of file collection
- \( m \): number of keywords

Client-side
- Security: CKA1, CKA2, UC
- Token generation & size: \( O(1) \) to \( O(n) \)

Server-side
- Search time: \( \text{OPT}, \text{OPT} \cdot \log(n), O(n), O(|f|) \)
- Index size: \( O(n), O(n \cdot m) \)
Security Definitions

- Security against chosen-keyword attack
  [Goh03, CM05, CGKO06]

  **CKA1**: “Protects files and keywords even if chosen by adversary”

- Security against adaptive chosen-keywords attacks [CGKO06]

  **CKA2**: “Protects files and keywords even if chosen by adversary, and even if chosen as a function of ciphertexts, index, and previous results”
Security Definitions

- UC [KO12]
  - Universal composability [Canetti01]

**UC:** “Remains CKA2-secure even if composed arbitrarily”
# Searchable Symmetric Encryption

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Dynamism</th>
<th>Security</th>
<th>Search</th>
<th>Verifiable</th>
<th>Leakage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWP00</td>
<td>No</td>
<td>CPA</td>
<td>$O(</td>
<td>f</td>
<td>)$</td>
</tr>
<tr>
<td>Goh03</td>
<td>Yes</td>
<td>CKA1</td>
<td>$O(n)$</td>
<td>No</td>
<td>++</td>
</tr>
<tr>
<td>CM05</td>
<td>No</td>
<td>CKA1</td>
<td>$O(n)$</td>
<td>No</td>
<td>++</td>
</tr>
<tr>
<td>CGKO06-1</td>
<td>No</td>
<td>CKA1</td>
<td>OPT</td>
<td>No</td>
<td>++</td>
</tr>
<tr>
<td>CGKO06-2</td>
<td>No</td>
<td>CKA2</td>
<td>OPT</td>
<td>No</td>
<td>++</td>
</tr>
<tr>
<td>CK10</td>
<td>No</td>
<td>CKA2</td>
<td>OPT</td>
<td>No</td>
<td>++</td>
</tr>
<tr>
<td>vLSDHJ10</td>
<td>Yes</td>
<td>CKA2</td>
<td>$O(\log m)$</td>
<td>No</td>
<td>+++</td>
</tr>
<tr>
<td>KO12</td>
<td>No</td>
<td>UC</td>
<td>$O(n)$</td>
<td>No</td>
<td>+++</td>
</tr>
<tr>
<td>KPR12</td>
<td>Yes</td>
<td>CKA2</td>
<td>OPT</td>
<td>No</td>
<td>+++</td>
</tr>
<tr>
<td>KP13</td>
<td>Yes</td>
<td>CKA2</td>
<td>OPT·$\log(n)$</td>
<td>Yes</td>
<td>++</td>
</tr>
<tr>
<td>KPR13</td>
<td>Yes</td>
<td>UC</td>
<td>OPT</td>
<td>Yes</td>
<td>+++</td>
</tr>
</tbody>
</table>
1. Build inverted/reverse index

Posting list

2. Randomly permute array & nodes
2. Randomly permute array & nodes

3. Encrypt nodes
3. Encrypt nodes

4. “Hash” keyword & encrypt pointer
Limitations of SSE-1

- Non-adaptively secure $\Rightarrow$ adaptive security
  - Idea #1 [Chase-K.-10]
    - replace encryption scheme with symmetric non-committing encryption
    - only requires a PRF + XOR
    - 😞: doesn’t work for dynamic data
  - Idea #2
    - Use RO + XOR
Limitations of SSE-1

- Static data $\implies$ dynamic data

- Problem #1:
  - Given new file $F_N = (AAPL, \ldots, MSFT)$
  - Append node for $F$ to list of every $w_i$ in $F$

1. Over unencrypted index

2. Over encrypted index ???

<table>
<thead>
<tr>
<th>MSFT</th>
<th>$F_2$</th>
<th>$F_{10}$</th>
<th>$F_{11}$</th>
<th>$FN$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOOG</td>
<td>$F_2$</td>
<td>$F_8$</td>
<td>$F_{14}$</td>
<td></td>
</tr>
<tr>
<td>AAPL</td>
<td>$F_1$</td>
<td>$F_2$</td>
<td>$FN$</td>
<td></td>
</tr>
<tr>
<td>IBM</td>
<td>$F_4$</td>
<td>$F_{10}$</td>
<td>$F_{12}$</td>
<td></td>
</tr>
</tbody>
</table>

| $F_K(GOOG)$ | $Enc(\bigcirc)$ |
| $F_K(IBM)$ | $Enc(\bigcirc)$ |
| $F_K(AAPL)$ | $Enc(\bigcirc)$ |
| $F_K(MSFT)$ | $Enc(\bigcirc)$ |
Limitations of SSE-1

- Static data ⇒ dynamic data
  - Problem #2:
    - When deleting a file $F_2 = (AAPL, \ldots, MSFT)$
    - delete all nodes for $F_2$ in every list

1. Over unencrypted index

2. Over encrypted index ???

<table>
<thead>
<tr>
<th>MSFT</th>
<th>F2</th>
<th>F10</th>
<th>F11</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOOG</td>
<td>F2</td>
<td>F8</td>
<td>F14</td>
</tr>
<tr>
<td>AAPL</td>
<td></td>
<td>F1</td>
<td></td>
</tr>
<tr>
<td>IBM</td>
<td>F4</td>
<td>F10</td>
<td>F12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$F_K(GOOG)$</th>
<th>$Enc(\bullet)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_K(IBM)$</td>
<td>$Enc(\bullet)$</td>
</tr>
<tr>
<td>$F_K(AAPL)$</td>
<td>$Enc(\bullet)$</td>
</tr>
<tr>
<td>$F_K(MSFT)$</td>
<td>$Enc(\bullet)$</td>
</tr>
</tbody>
</table>
Dynamic SSE
[K.-Papamanthou-Roeder12]

- Static data ⇒ dynamic data
  - Idea #1
    - Memory management over encrypted data
    - Encrypted free list
  - Idea #2
    - List manipulation over encrypted data
    - Use homomorphic encryption (here just XOR) so that pointers can be updated obliviously
  - Idea #3
    - Deletion is handled using a “dual” SSE scheme
    - Given deletion/search token for $F_2$, returns pointers to $F_2$’s nodes
    - Then add them to the free list homomorphically
Structured Encryption
Structured Encryption

- Searchable encryption
  - Private *keyword search* over encrypted *text data*

- **Q**: can we privately query other types of enc. data?
  - maps
  - image collections
  - social networks
  - web page archives
Graph-structured Data

- Communications
  - email headers, phone logs
- Research papers
  - citations
- Networks
- Social networks
- Web crawlers
- Maps
Structured Encryption

\[ \text{Enc}_K \]

\[ \text{Enc}_K(\square) \quad \text{Enc}_K(\bullet) \]
Structured Encryption

\[ \text{Enc}_K \left( \begin{array}{c} \text{Node} 1 \\ \text{Node} 2 \\ \text{Node} 3 \end{array} \right) \]

\[ \text{Enc}_K \left( \begin{array}{c} \text{Node} 4 \\ \text{Node} 5 \end{array} \right) \]
Structured Data

- Email archive = Index + Email text
Structured Data

- Social network = Graph + Profiles
Structured Encryption

- $\text{Gen}(1^k) \Rightarrow K$
- $\text{Enc}_K(\delta, \overline{m}) \Rightarrow (\gamma, \overline{c})$
- $\text{Token}_K(q) \Rightarrow t$
- $\text{Query}(\gamma, t) \Rightarrow I$
- $\text{Dec}_K(c_i) \Rightarrow m_i$
CQA2-Security

- Simulation-based definition
  - "given the ciphertext no adversary can learn any information about the data and the queries other than what can be deduced from the access and search patterns..."
  - "...even if queries are made adaptively"
  - access pattern: pointers to (encrypted) data items that satisfy query
  - query pattern: whether a query is repeated
- $\Omega(\lambda \cdot \log(n))$ lower bound on token size (in std. model)
  - n: # of data items
  - $\lambda$: # of relevant items
Constructions

- Adjacency queries on encrypted graphs
  - from lookup queries on encrypted matrices
- Neighbor queries on encrypted graphs
  - from keyword search on encrypted text (i.e., SSE)
- Focused subgraph queries on encrypted web graphs
  - from keyword search on encrypted text
  - from neighbor queries on encrypted graphs
Neighbor Queries on Graphs

\[ \text{Enc}_K \left( \begin{array}{c}
\text{Node}_1 \\
\text{Node}_2 \\
\text{Node}_3
\end{array} \right) + \begin{array}{c}
\text{Document}_1 \\
\text{Document}_2 \\
\text{Document}_3
\end{array} \]

\[ \text{Enc}_K \left( \begin{array}{c}
\text{Document}_4 \\
\text{Document}_5
\end{array} \right) \]
FSQ on Web Graphs

- Web graphs
  - Text data -- pages
  - Graph data --- hyperlinks
- Simple queries on web graphs
  - All pages linked from P
  - All pages that link to P
- Complex queries on web graphs
  - "mix" both text and graph structure
  - search engine algorithms based on link-analysis
    - Kleinberg's HITS [Kleinberg99]
    - SALSA
    - ...

Focused Subgraph Queries

- Search engine algorithms
  - Step 1: compute focused subgraph
  - Step 2: run iterative algorithm on focused subgraph

Crypto
FSQ on Encrypted Graphs

- Encrypt
  - pages with SE-KW
  - graph with SE-NQ
  - does not work!
- Chaining technique
  - combine SE schemes (e.g., SE-KW with SE-NQ)
  - preserves token size of first SE scheme
- Requires associative SE
  - message space: private data items and semi-private information
  - answer: pointers to data items + associated semi-private information
  - [CGKO06]: associative SE-KW but not CQA2-secure!
FSQ on Web Graphs

$\text{Enc}_K$
Private Information Retrieval
Private Information Retrieval
A Simple Solution to PIR

All

Large comm
An Ideal Solution

#$(dws#$

Small comm
Private Information Retrieval
Private Information Retrieval

- **Multi-server PIR** [Chor-Goldreich-Kushilevitz-Sudan95]
  - Servers cannot communicate
  - Information-theoretic security
  - Information theoretic security requires at least two servers

- **Single-server PIR** [Kushilevitz-Ostrovsky97]
  - Homomorphic encryption, phi-hiding
  - Computational security
    - trapdoor permutations, number theory, lattices

- **Keyword PIR** [Chor-Gilboa-Naor97]

- **Hardware-based PIR** [Smith-Safford01, Asonov-Freytag02, …]
  - $O(1)$ communication and $O(n)$ computation [SS01]
  - $O(1)$ communication and $o(n)$ computation [AF02]
$\#3$

$\mathbf{d} = \{4, 6, 9, 2\}$

$q = \{E(0), E(0), E(1), E(0)\}$

$a = 4xE(0) + 6xE(0) + 9xE(1) + 2xE(0)$

$O(n)$
#3

\[ d = \begin{pmatrix} 4 & 6 \\ 9 & 2 \end{pmatrix} \]

\[ q = \begin{pmatrix} E(0), E(1) \end{pmatrix} \]

\[ a = 4xE(0) + 9xE(1) ; 6xE(0) + 2xE(1) \]

\[ O(\sqrt{n}) \]
PIR Connections

- Oblivious Transfer
  - OT = symmetric PIR
  - Privacy for both client and server
- Locally decodable codes [Katz-Trevisan00]
  - Multi-server PIR = locally decodable code
- Collision-resistant hash functions [Ishai-Kushilevitz-Ostrovsky05]
  - Single-server single-round PIR $\Rightarrow$ collision-resistant hash function
Oblivious RAMs
Oblivious RAM

“Allows client to read & write to memory/storage without revealing access pattern”
ORAM vs. PIR

- **Generality**
  - ORAM protects arbitrary computations
  - (traditionally) PIR protects lookups (+ keyword search)

- **Public/Private data**
  - PIR does not hide data from server
  - ORAM hides data from server

- **Server**
  - ORAM “server” does not compute!
  - PIR server computes

- **Computational complexity**
  - ORAM “server” can do \( o(n) \) work
  - PIR server has to do \( O(n) \) work
ORAM vs. PIR

- PIR parameters
  - communication complexity since $\Theta(n)$ computation required
- ORAM parameters
  - round complexity, client storage, server storage
Timeline of Previous Work

1990

[Ostrovsky]

Feb 2008

[Williams-Sion]

Oct 2008

[Williams-Sion-Carbunar]

Jun 2010

[Pinkas-Reinman]

Jul 2010

[Goodrich-Mitzenmacher]

from O. Ohrimenko
Timeline of Previous Work

2011

[Goodrich-Mitzenmacher-Ohrimenko-Tamassia]
[Shi-Chan-Stefanov-Li]

from O. Ohrimenko
Square Root Scheme

[Goldreich-Ostrovsky96]

**Setup**

1. Add $\sqrt{n}$ dummy items
2. Add $\sqrt{n}$ cache
3. Randomly permute real & dummy items

---

dummies

from O. Ohrimenko
Square Root Scheme
[Goldreich-Ostrovsky96]

Setup
1. Add $\sqrt{n}$ dummy items
2. Add $\sqrt{n}$ cache
3. Randomly permute real & dummy items
4. Encrypt
5. Send to server

from O. Ohrimenko
Read #1
1. Scan cache
2. If item #1 is not in cache read P(1)
3. Write item to cache

Cannot access location P(1) anymore!

Square Root Scheme

[Goldreich-Ostrovsky96]
Read #i
1. Scan cache
2. If item #i is not in cache read P(i)
3. Write item to cache

Square Root Scheme
[Goldreich-Ostrovsky96]

Cannot access location P(1) anymore!

from O. Ohrimenko
Square Root Scheme

[Goldreich-Ostrovsky96]

Read #1 (again)
1. Scan cache
2. If item #1 is not in cache read P(1)
3. Write item to cache
4. If item is in cache, read dummy item
5. Write dummy to cache

Cannot access location P(1) anymore!

from O. Ohrimenko
Read #1 (again)
1. Scan cache
2. If item #1 is not in cache read P(1)
3. Write item to cache
4. If item is in cache, read dummy item
5. Write dummy to cache

Square Root Scheme
[Goldreich-Ostrovsky96]

main memory
\(n + \sqrt{n}\)
cache
\(\sqrt{n}\)

Read #P(n+1)

from O. Ohrimenko
Square Root Scheme

[Goldreich-Ostrovsky96]

Rebuild
1. Merge all items
2. Clear cache
3. Remove duplicates
4. Permute with new permutation

Cache gets full after $\sqrt{n}$ requests

from O. Ohrimenko
Square Root Scheme
[Goldreich-Ostrovsky96]

- Analysis
  - Each request needs $\sqrt{n} + 1$ accesses
  - Each rebuild needs $n \log^2 n$ accesses every $\sqrt{n}$ requests
  - Total accesses for $r$ requests is $O(r \sqrt{n} + n \log 2n r / \sqrt{n})$
  - Amortized complexity per request is $O(\sqrt{n} \log 2n)$
## Oblivious RAMs

<table>
<thead>
<tr>
<th>Reference</th>
<th>Client Storage</th>
<th>Round-trips</th>
<th>Message size</th>
<th>Server Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GO96]</td>
<td>$O(1)$</td>
<td>$O(\sqrt{n})$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>[SCSL11]</td>
<td>$O(\log^2 n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log^2 n)$</td>
<td>$O(n\log n)$</td>
</tr>
<tr>
<td>[GMOT12]</td>
<td>$O(\sqrt{n})$</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
Proofs of Storage
Simple Solutions

Cloud can just store hash!

Linear comm. complexity
Simple Solutions

Large client storage

\[ F_{K1} \rightarrow F_{K2} \rightarrow F_{K3} \rightarrow K1 \rightarrow \cdots \rightarrow F_{K1} \]

\[ \text{Bounded \# of verifications} \]
Proof of Storage

Ateniese+07, Juels-Kaliski07

\[ O(1) \]

Petabytes

\[ \pi \]

\[ C \]

K
PoS = PoR v PDP

- Proof of retrievability [Juels-Kaliski07]
  - High tampering: detection
  - Low tampering: retrievability
- Proof of data possession [Ateniese+07]
  - Detection
PoS Security

- Completeness

**COMP**: “if Server possesses file, then Client accepts proof”

- Soundness

**SOUND**: “if Client accepts proof, then Server possesses file”
Formalizing Possession

- Knowledge extractor
  - [Feige-Fiat-Shamir88, Feige-Shamir90, Bellare-Goldreich92]
  - Algorithm that extracts information from other algorithms
  - Typically done by rewinding
- Adapted to PoS soundness

**SOUND**: “there exists an expected poly-time extractor $K$ that extracts the file from any poly-time $A$ that outputs valid proofs”
Designing PoS

- Based on sentinels
  - [Juels-Kaliski07]
  - Embed secret blocks in data and verify their integrity
  - 😊 Very efficient encoding
  - 😞 Only works with private data

- Based on homomorphic linear authenticators (HLA)
  - [Ateniese+07]
  - Authenticates data with tags that can be aggregated
  - 😊 works with public data
HLA-based PoS

HLA

Semi-compact PDP

PRF

Compact PDP

Erasure code

1 2 3 4

HLA

1 2 3 4 EC EC

Semi-compact PoR

Compact PoR

1 2 3 4 t_1 t_2 t_3 t_4 t_5 t_6

EC

PRF
HLA-based PoS

**SOUND:** “there exists an expected poly-time extractor $K$ that extracts the file from any poly-time $A$ that outputs valid proofs”
Extracting a File

**SOUND:** “there exists an expected poly-time extractor $K$ that extracts the file from any poly-time $A$ that outputs valid proofs”

1. If $c_1$ and $c_2$ are lin. Indep.
2. solve for $f$ using linear algebra

$$f = \text{Extract}$$
Extracting a File

1. If $c_1$ and $c_2$ are linearly independent, solve for $f$ using linear algebra.
2. If $c_1$ and $c_2$ are not linearly independent, just pick them at random.

What if $A$ doesn’t compute inner product? Use HLAs!
HLA

- Syntax
  - $\text{Gen}(t^k) \Rightarrow K$
  - $\text{Tag}(K, f) \Rightarrow (t, st)$
  - $\text{Chall}(1^k) \Rightarrow c$
  - $\text{Auth}(K, f, t, c) \Rightarrow \alpha$
  - $\text{Vrfy}(K, \mu, c, T) \Rightarrow b$

- Security

**UNF:** “given $f$ and $c$, no $\mathcal{A}$ can output a valid $\alpha$ for an element $\mu \neq \langle c, f \rangle$”
Simple HLA [Shacham-Waters08]

\[ t_i = H_K(i) + f_i \cdot w \]

\[ \mu = \langle c, f \rangle \quad \text{and} \quad \alpha = \langle c, t \rangle \]

\[ \alpha = \langle t, (H_K(1), \ldots, H_K(n)) \rangle + \mu \cdot w \]
Simple HLA

**UNF:** “given \( f \) and \( c \), no \( \mathcal{A} \) can output a valid \( \alpha \) for an element \( \mu \neq \langle c, f \rangle \)”

- UNF: \( \alpha \) proves that \( \mu \) is the inner product of \( f \) and \( c \)
- Why is Simple HLA unforgeable?
  - For intuition see [Ateniese-K.-Katz10]
  - Connection to 3-move identification protocols
Simple HLA = Semi-Compact PoS

\[ t_i = H_K(i) + f_i \cdot w \]

\[ \alpha = \langle t, (H_K(1), \ldots, H_K(n)) \rangle + \mu \cdot w \]

\[ \mu = \langle c, f \rangle \text{ and } \alpha = \langle c, t \rangle \]

Everything in \( \mathbb{Z}_p \)

\( O(n)! \)

\( O(1) \)
Compressing Challenges

- Idea #1
  - [Ateniese+07]
  - Send key to a PRF and have server generate challenge vector
  - **Problem:** how do we reduce to PRF security if $A$ knows the PRF key?

- Idea #2
  - [Shacham-Waters08] Use a random oracle

- Idea #3
  - [Dodis-Vadhan-Wichs10] Use an expander-based derandomized sampler

- [Ateniese-K.-Katz10]
  - Idea #1 is secure
  - Security of PRF implies that PRF-generated vectors are linearly independent with high probability
Other Topics

- Order Preserving Encryption
  - [Agrawal+04, Boldyreva-Chenette-O’Neill11,...]
- Private stream search
  - [Ostrovsky-Skeith05,...]
- Verifiable computation
  - [Goldwasser-Kalai-Rothblum08, Gennaro-Gentry-Parno10,...]
The End
Simulation-based definition

``given the encrypted index, encrypted files and search tokens, no adversary can learn any information about the files and the search keywords other than what can be deduced from the access and search patterns...”

“...even if queries are made adaptively”

access pattern: pointers to (encrypted) files that satisfy search query

query pattern: whether a search query is repeated
CKA2-Security

Real World

\[ \text{Enc}_K(\text{document}) \]

q

\[ \top \]

Ideal World

\[ \mathcal{L}_1(\text{document}) \]

\(?$s!l)csd@#C\]

q | \mathcal{L}_1(q)

@#kj^%ks#
Equivocation

Ideal World

$q \mid \mathcal{L}_1(q)$

$q \mid \mathcal{L}_1(q)$