Big Data Analytics over Encrypted Datasets with Seabed

Antonis Papadimitriou ★, Ranjita Bhagwan ☆, Nishanth Chandran ☆, Ramachandran Ramjee ☆, Andreas Haeberlen ★, Harmeet Singh ☆, Abhishek Modi ☆, Saikrishna Badrinarayanan ☆

★ University of Pennsylvania, ☆ Microsoft Research India, ☆ UCLA
Motivation: Big data analytics on sensitive data

<table>
<thead>
<tr>
<th>customer</th>
<th>gender</th>
<th>country</th>
<th>payment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>female</td>
<td>CAN</td>
<td>12</td>
</tr>
<tr>
<td>Bob</td>
<td>male</td>
<td>USA</td>
<td>4</td>
</tr>
<tr>
<td>Charlie</td>
<td>female</td>
<td>USA</td>
<td>1</td>
</tr>
<tr>
<td>Deborah</td>
<td>female</td>
<td>USA</td>
<td>15</td>
</tr>
</tbody>
</table>

- Goal: Outsource big data analytics
  - Store database at a cloud provider
  - Perform analytical queries remotely

- Problem: Rogue cloud admins or hackers could have access to data
  - Sensitive information can be exposed
Prior work: Encrypted databases

- **What can we do?**
  - Use encryption!
  - Examples: CryptDB/Monomi [SOSP11, VLDB13], MS SQL Server [SQL16]
  - These support SQL queries on encrypted data

<table>
<thead>
<tr>
<th>customer</th>
<th>gender</th>
<th>country</th>
<th>payment</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Th6j</td>
<td>h4$89</td>
<td>548yvg</td>
<td>439856</td>
</tr>
<tr>
<td>Fjg893</td>
<td>sfbg43</td>
<td>a3vbt9a</td>
<td>582650</td>
</tr>
<tr>
<td>%gTHR</td>
<td>h4$89</td>
<td>a3vbt9a</td>
<td>143759</td>
</tr>
<tr>
<td>34%^d</td>
<td>h4$89</td>
<td>a3vbt9a</td>
<td>874563</td>
</tr>
</tbody>
</table>
Encrypted databases – Challenges

- **Challenge 1: Performance**
  - Aggregations on encrypted data are slower
  - Ciphertext addition is > 3000x slower than plaintext
  - Adding 100 million values takes 6 minutes instead of 100ms
    - Not good for big data!
Encrypted databases – Challenges

• Challenge 2: Security
  • Encrypted databases use cryptographic schemes with weaker guarantees
  • Example: deterministic encryption (DET) reveals equality
  • Recent attack [CCS15] recovered > 60% from certain DET columns
Our approach

<table>
<thead>
<tr>
<th>customer</th>
<th>gender</th>
<th>revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Th6j&amp;</td>
<td>h4$89h</td>
<td>987242</td>
</tr>
<tr>
<td>Fjg893n</td>
<td>sfbg43q</td>
<td>629459</td>
</tr>
<tr>
<td>%gTHR3</td>
<td>h4$89h</td>
<td>593292</td>
</tr>
<tr>
<td>34%^db</td>
<td>h4$89h</td>
<td>742063</td>
</tr>
</tbody>
</table>

Goal 1: Improve performance
- ASHE – New cryptographic scheme that allows fast aggregation on encrypted data

Goal 2: Improve security
- SPLASHE: DB transformation that enables more queries without using weaker crypto
Seabed: Big data analytics for encrypted datasets

• We built Seabed on top of Spark
  • Seabed leverages ASHE and SPLASHE

• Seabed runs SQL queries on encrypted data
  • Examples: Group-by queries and aggregations (sum, average, variance)

• Seabed is fast enough for big data
  • Up to 100x faster than previous systems
Outline

• Motivation & prior work
• Approach
  • Improving performance
    • ASHE
  • Improving security
    • SPLASHE
• System design
• Evaluation
Why is aggregation slow in encrypted databases?

- We need to sum up encrypted data
  - This is impossible with schemes like AES
- We need an additively homomorphic cryptosystem
  - Example: Paillier encryption [EUROCRYPT99]
  - $\text{Enc}(x_1) \oplus \text{Enc}(x_2) = \text{Enc}(x_1 + x_2)$
Why is aggregation slow in encrypted databases?

- Most homomorphic cryptosystems are expensive!
  - Example: Paillier ciphertexts need to be 2048-bit
  - Homomorphic addition: $Enc(x_1) \oplus Enc(x_2) = Enc(x_1) \times Enc(x_2)$
  - > 3000x slower than plain addition
Can we have faster homomorphic cryptosystems?

• But why does Paillier have so large ciphertexts?
  • Because it is an asymmetric scheme based on large integers
  • Encrypt with public key – decrypt with private key

• Do we need asymmetric crypto in outsourced databases?
  • Analysts and data collector usually work for the same organization
  • We could use fast symmetric crypto!
ASHE – Additive Symmetric Homomorphic Encryption

- Encrypt by masking values with random numbers
  - ASHE is semantically secure (IND-CPA)
- No need to remember random numbers
  - Use pseudorandom function F(ID)
- ASHE ciphertexts are 32/64-bit integers
  - Homomorphic addition only takes a few nanoseconds!
ASHE – Optimizations

- **Challenge:** Aggregation and decryption cost depends on ID list length

- **Optimizations:**
  - Optimize encryption so that the randomness cancels out for consecutive IDs
  - Fast evaluation of pseudorandom function via AES-NI
  - Compression techniques to make ID list as small as possible

- **Outcome:** ASHE enables fast aggregation even when the DB is very large
Outline

• Motivation & prior work
• Approach
• Improving performance
  • ASHE
• Improving security
  • SPLASHE
• System design
• Evaluation
Why are encrypted databases vulnerable?

- Some columns use deterministic encryption (DET)
- This leaks the distribution of values
- An adversary with auxiliary information can do a frequency attack [CCS15]
  - In the example, the gender is revealed
How can we avoid deterministic encryption?

SELECT sum (revenue) FROM purchases WHERE gender = “female”

<table>
<thead>
<tr>
<th>customer</th>
<th>gender</th>
<th>payment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>female</td>
<td>12</td>
</tr>
<tr>
<td>Bob</td>
<td>male</td>
<td>4</td>
</tr>
<tr>
<td>Charlie</td>
<td>female</td>
<td>1</td>
</tr>
<tr>
<td>Deborah</td>
<td>female</td>
<td>15</td>
</tr>
</tbody>
</table>

- Support single-table aggregation queries without DET
- SPLASHE: Transform DB schema to avoid DET
  - Answer single-table aggregation queries using additions only
- Some storage overhead
  - Reduced by Enhanced SPLASHE (see paper)
We implemented Seabed on top of unmodified Spark
  • ASHE and SPLASHE implemented in Scala

Seabed’s high-level design is similar to CryptDB’s
  • Accepts SQL queries; transparently answers them on encrypted data
  • Client proxy handles encryption/decryption
Outline

• Motivation & prior work
• Approach
• Improving performance
  • ASHE
• Improving security
  • SPLASHE
• System design
• Evaluation
Evaluation: Questions

- End-to-end latency of aggregation?
- Storage overhead of SPLASHE?
- End-to-end latency in Bing Ads analytics?
- How scalable is aggregation?
- How effective are Seabed’s optimizations?
- Latency of group-by queries?
- Latency of batch queries (Big Data Benchmark)?

Experimental setup:
- Spark with 100 cores
- On MS Azure
- Memory-resident data
How efficient is ASHE aggregation?

- Synthetic data: up to 1.75 billion rows - Query: single column aggregation

- Results
  - Paillier: up to 16.6 minutes
  - No encryption: <1 second
  - How does Seabed do?

- Seabed is 100x faster than Paillier, even in the worst case!
How much storage does SPLASHE need?

- **Dataset**
  - 760M rows, real ad-analytics application from Microsoft
- **We replaced 10 DET columns with SPLASHE, one by one**
- **Measured:** Relative size increase vs. plaintext dataset
- **Results**
  - SPLASHE has substantial storage cost
  - Enhanced SPLASHE reduces this cost by up to 10x
- **With 10x more storage, we avoid DET entirely!**
  - Reduces risk of information leaks
How efficient is Seabed for real-world applications?

- Same ad-analytics application from Microsoft
  - Measured: End-to-end latency of 15 queries

- Results
  - No encryption is about 10x faster than Paillier across all queries
  - Seabed is almost as fast as no encryption (within 15-44%)

- It is possible to do analytics on encrypted big data!
Summary

- Big-data analytics on encrypted data is difficult
  - Key challenges: Performance, security
- We introduce additive symmetric homomorphic encryption (ASHE)
  - Result: much better performance when analyst and data owner trust each other
- We present a schema transformation called SPLASHE
  - Result: Often avoids the need for weaker encryption → better security
- Seabed: an extension of Spark that uses ASHE and SPLASHE
  - Up to 100x faster than previous systems
- Seabed is fast enough for real-world big data applications

Any Questions?
References


