Security and Performance Analysis of Encrypted NoSQL Databases

M.W. Grim BSc., Abe Wiersma BSc.

Supervisor: F. Turkmen PhD

February 6, 2017

University of Amsterdam
Introduction

Problem

Securely storing BigData on NoSQL database systems.
Introduction

Problem

Securely storing BigData on NoSQL database systems.

Necessary because:

- PRISM
- Security vulnerabilities
  1. Ashley Madison
  2. Yahoo
  3. LinkedIn
Introduction

**Problem**

Securely storing BigData on NoSQL database systems.

Necessary because:

- PRISM
- Security vulnerabilities
  1. Ashley Madison
  2. Yahoo
  3. LinkedIn

**Solution**

Encrypt your plain-text data.
Introduction
Plain data
Introduction

Encryption at rest
Introduction

Encryption at rest

- **Application**: Application data is encrypted by application itself before storing it to disk.
- **File system**: Individual files and directories are encrypted by the filesystem when stored to disk.
- **Block manager**: Data blocks are encrypted by the driver before they are written to disk.
- **Hardware**: Encryption of data is fully handled by the firmware of the disk.
Introduction
Research questions

- How is SQL-aware encryption realised in NoSQL database engines?
  - What kind of security does it provide?
  - How does it compare to encryption at rest?
- What is the performance impact of enabling encryption?
  - What limitations are there in terms of functionality?
Computation over encrypted data
Computation over encrypted data

End-to-end encrypted database

- Key stored at client.
- Encryption and decryption by client (end-to-end).
Computation over encrypted data
End-to-end encrypted database

- Key stored at client.
- Encryption and decryption by client (end-to-end).
- Server can’t read data, how to query?
Key stored at client.

- Encryption and decryption by client (end-to-end).
- Server can’t read data, how to query?
- Homomorphic encryption / Order Revealing Encryption
Computation over encrypted data
Paillier

- Partially homomorphic.
- Encrypted addition.

\[ E(m_1) + E(m_2) = E(m_1 + m_2) \]
Computation over encrypted data

ElGamal

- Partially homomorphic.
- Encrypted multiplication.

$$E(m_1) \times E(m_2) = E(m_1 \times m_2)$$
Public compare function on encrypted data.

-1 smaller
0 equal
1 greater

$x > y = -1$
SecureMongo
SecureMongo

- Based on work by Alves et al.
- Python connector wrapper.
- Logic at client side.
- End-to-end encryption with queries on encrypted data.
SecureMongo

- Based on work by Alves et al.
- Python connector wrapper.
- Logic at client side.
- End-to-end encryption with queries on encrypted data.

Our work:

- Sequential inserts.
- Serialized AVL tree.
- Tree balancing at server side.
SecureMongo
AVL tree

*Self-balancing* binary search tree.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Average</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Search</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>
SecureMongo
overview

movies_idx_year

movies

movies_idx_rate

1978
1988
1994
1995
2000
2008
2014

The Shawshank Redemption, 1994, 9
Rain Men, 1988, 8
The Dark Knight, 2008, 9
The Boys from Brazil, 1978, 7
Interstellar, 2014, 9
Memento, 2000, 9
The Usual Suspects 1995, 9

Order Revealing encryption
AES encryption
Order Revealing encryption
SecureMongo

selection

### Client

- **Step 1**
  - 1995
  - `Xel.Qk /3Cid`

- **Step 2**
  - `/GuN kbax2`
  - `Qo@j L=s23`
  - `WGMx 3Aa=
  - `Xel.Qk /3Cid`

- **Step 3**
  - `/6K7LyKXbQd61bToR2 Lpx9cpNSpyH0KaoEeNx`

- AES decrypt

### Movies

- `movies_idx_ year`

- `movies`

The Usual suspects

1995, 9

<table>
<thead>
<tr>
<th>User</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>pUstlVRtxVaY87omBm07S+swUCUldzoCQqSdc</td>
<td>ChTBZAx7Wp0nBFDCSr5fNukiCWMzXEOxXARc</td>
</tr>
<tr>
<td>k32b1rnx29x1XNuud9Ab1L3xuUfgcJQqpl</td>
<td>6tx9SKsR0WwLe1q5M e3yCOAXkBluH0Xmg3</td>
</tr>
<tr>
<td>/Fux+MLok+Kao2TeGcAmxy29u3Azm4QO24+657</td>
<td>Yi/GuQekaM43NdMjmnRNtxTelkuqpeew3V98</td>
</tr>
<tr>
<td>/6K7LyKXbQd61bToR2 Lpx9cpNSpyH0KaoEeNx</td>
<td></td>
</tr>
</tbody>
</table>
SecureMongo

insertion

Client

movies_idx_year

movies

1995

The Usual Suspects
1995, 9

ORE

AES encryption

insert

insert

+/GuN kbax2

QkaEp YChom

Qo@j L=s23

51Se Usti

WGMx 3As=

AeM/ Z4Ae

XeL.Qk /3Cld

pUsti\VWRfxVaY87Qm8r
mo7S+swUCldrzeCKqSCd

ChTbZAozWp0nBPcSR5f
NukiCWMzXE0/xcARc

k32b1xa2/x1XNiud/
9Ab1L/L3xsu/flgcJQp/

6bxQKSaRoWwLelq5M
e3yCOAXWKBiuH0XmG3

/Fux+ML0K+Ka2TeGcAm
xy2hu3Azm4QQ2j+6S7

Yf+/GuQekaM43NdMjhn
RNTnTelifuqsPeRrVb

o6K7lyKXbQd61bTBgR2
LPx9cpNSPyHD0KaoEeNx
Method
Method
Our work

- Studied homomorphic / order revealing encryption
- Improved earlier work by Alves et al.
- Evaluated performance and security
  1. Encryption at rest
  2. End-to-end encryption
Method
Plain vs. encryption at rest

YCSB

Command-line parameters
- DB to use
- Target throughput
- Number of threads
- ...

Workload parameter file
- R/W mix
- Record size
- Data set
- ...

Extensible: define new workloads
Extensible: plug in new clients

YCSB client
Workload executor
Client ~ threads ~ Stats

DB client

Cloud DB
Method
Plain vs. encryption at rest

- YCSB default core workload.
- Adjustable with parameters.
- Can extend framework with alternative workloads.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>recordcount</td>
<td>16,000,000</td>
</tr>
<tr>
<td>operationcount</td>
<td>100,000</td>
</tr>
<tr>
<td>readproportion</td>
<td>0.5</td>
</tr>
<tr>
<td>updateproportion</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Method
Plain vs. computation over encrypted data

- BenchmarkDB
- Python framework
- IMDB movies
Results encryption at rest
Results

Performance encryption at rest

- Insert operations per second
  - Not encrypted: 9000, 9400, 9800, 10200
  - Encryption at rest: 140, 160, 180, 200, 220

- Read/update operations per second
  - Not encrypted: 0.0000, 0.0005, 0.0010, 0.0015, 0.0020, 0.0025, 0.0030, 0.0035, 0.0040
  - Encryption at rest: 0.00, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07
Results
Performance encryption at rest

Insert
8000
8500
9000
9500
10000
10500
median(ops/s)
Not encrypted
Encryption at rest

Read/Update
150
160
170
180
190
200
median(ops/s)
Not encrypted
Encryption at rest

Insert
4.9% lower throughput

Read/Update
7.3% lower throughput
Results
Performance encryption at rest

Insert 5.2% slower
Read 7.4% slower
Update 7.5% slower
Results SecureMongo
## Results

### Performance SecureMongo

<table>
<thead>
<tr>
<th>Database size</th>
<th>Average latency</th>
<th>Mongo read</th>
<th>MongoSecure read</th>
<th>Mongo write</th>
<th>MongoSecure write</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td>0.06</td>
<td>0.06</td>
<td>0.06</td>
<td>0.06</td>
<td></td>
</tr>
</tbody>
</table>

![Graph showing performance comparison](image-url)
Results security
**Results**

**Security threat model**

**Threat 1**
Full access to the database server, both logical and physical.

**Threat 2**
The application server and database server are compromised arbitrarily.
Threat 1: plain

**Issue**

The plain-text data is there no elbow grease required for access.
### Threat 1: encrypted at rest

<table>
<thead>
<tr>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key is continuously needed on server.</td>
</tr>
</tbody>
</table>

1. Cold-boot extraction from memory (always).
2. Extract from hard-disk (if key is stored on disk).
3. Retrievable from secondary server by posing as the database-server (can be negated by two factor key retrieval).

The AES used is AES-256CBC which is IND-CPA secure. The AES cryptosystem is run using OpenSSL in accordance with FIPS 140-2.
Threat 1: SecMongo framework

1. **AES** encryption used in AES-128CBC is IND-CPA secure. PyCrypto is used with a randomly generated IV for every encryption.

2. **ORE** proposed by Lewi and WU offers IND-OCPA.

3. **ElGamal** is proven IND-CPA secure.

4. **Paillier** is proven IND-CPA secure.

5. The **AVL-tree** implementation negates inference attack robustness.
Threat 2: plain

<table>
<thead>
<tr>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>The plain set-up is still utterly compromised.</td>
</tr>
</tbody>
</table>
**Results**

**Security threat model**

---

**Threat 2: encrypted at rest**

**Issue**

Key retrieval was already possible using a cold-boot attack, threat expansion means decrypted data can be retrieved by posing as the application.
## Threat 2: SecMongo framework

<table>
<thead>
<tr>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key is continuously needed by the application.</td>
</tr>
</tbody>
</table>
Conclusion
Solution

Encrypt your plain-text data.
**Conclusion**

<table>
<thead>
<tr>
<th><strong>Solution</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt your plain-text data. ✓</td>
</tr>
</tbody>
</table>
### Conclusion

<table>
<thead>
<tr>
<th><strong>Solution</strong></th>
<th>Encrypt your plain-text data. ✓</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TradeOff</strong></td>
<td>Security ↔ Performance</td>
</tr>
</tbody>
</table>


Discussion & Future work
Discussion & Future work

- Native Tree traversal in MongoDB would increase performance for Secure Mongo Framework, iterative tree traversal would be done on the server.
- Although range requests are possible using the ORE encryption, they are not yet implemented.
Questions?