MongoDB and QE?
Jobs at SpringField

"Nuclear Technician"  "CEO"
MongoDB database

Document collection

doc_id  document
1  doc1 = {
    "Name": "Homer Simpson",
    "Job": "Nuclear Technician",
    "Address": "742 Evergreen Terrace"
}

2  doc2 = {
    "Name": "Lenny Leonard",
    "Job": "Nuclear Technician",
    "Address": "123 Evergreen Terrace"
}

3  doc3 = {
    "Name": "Charles Montgomery Burns",
    "Job": "CEO",
    "Address": "1000 Mammon Street"
}
MongoDB database

Document collection

(doc_id  document)

1  doc1 = {
    "Name": "Homer Simpson",
    "Job": "Nuclear Technician",
    "Address": "742 Evergreen Terrace"
}

2  doc2 = {
    "Name": "Lenny Leonard",
    "Job": "Nuclear Technician",
    "Address": "123 Evergreen Terrace"
}

3  doc3 = {
    "Name": "Charles Montgomery Burns",
    "Job": "CEO",
    "Address": "1000 Mammon Street"
}

(Inverted) search index

<table>
<thead>
<tr>
<th>field value</th>
<th>doc_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Nuclear Technician&quot;</td>
<td>1, 2</td>
</tr>
<tr>
<td>&quot;CEO&quot;</td>
<td>3</td>
</tr>
</tbody>
</table>

Equality search on the "Job" field

 Linear scan
MongoDB database = QE

Document collection

<table>
<thead>
<tr>
<th>doc_id</th>
<th>document</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>doc1 = {</td>
</tr>
</tbody>
</table>
|        | "Name": "Homer Simpson",
|        | "Job": "Nuclear Technician",
|        | "Address": "742 Evergreen Terrace"
|        | }        |
| 2      | doc2 = {  |
|        | "Name": "Lenny Leonard",
|        | "Job": "Nuclear Technician",
|        | "Address": "123 Evergreen Terrace"
|        | }        |
| 3      | doc3 = {  |
|        | "Name": "Charles Montgomery Burns",
|        | "Job": "CEO",
|        | "Address": "1000 Mammon Street"
|        | }        |

(Inverted) search index

<table>
<thead>
<tr>
<th>field value</th>
<th>doc_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Nuclear Technician&quot;</td>
<td>1, 2</td>
</tr>
<tr>
<td>&quot;CEO&quot;</td>
<td>3</td>
</tr>
</tbody>
</table>

Equality search on the "Job" field
MongoDB and QE

MongoDB claims that QE

2. **Data encrypted throughout its lifecycle:** Queryable Encryption adds another layer of security for your most sensitive data, where data remains secure in-transit, at-rest, in memory, in logs, and in backups. Additionally, Queryable Encryption encrypts data as fully randomized on the server-side.

MongoDB and QE

<table>
<thead>
<tr>
<th>Database Management System</th>
<th>Ranking Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>1,247.52</td>
</tr>
<tr>
<td>MySQL</td>
<td>1,196.45</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>939.09</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>816.5</td>
</tr>
<tr>
<td>MongoDB</td>
<td>612.77</td>
</tr>
<tr>
<td>Redis</td>
<td>173.83</td>
</tr>
<tr>
<td>IBM DB2</td>
<td>142.07</td>
</tr>
<tr>
<td>Elasticsearch</td>
<td>138.6</td>
</tr>
<tr>
<td>SQLlite</td>
<td>132.67</td>
</tr>
<tr>
<td>Microsoft Access</td>
<td>131.03</td>
</tr>
<tr>
<td>Cassandra</td>
<td>115.22</td>
</tr>
<tr>
<td>Snowflake</td>
<td>115.05</td>
</tr>
<tr>
<td>MemSQL</td>
<td>88.81</td>
</tr>
<tr>
<td>Sphinx</td>
<td>87.08</td>
</tr>
<tr>
<td>Amazon DynamoDB</td>
<td>79.69</td>
</tr>
</tbody>
</table>

Business customers:

2: [https://www.mongodb.com/who-uses-mongodb](https://www.mongodb.com/who-uses-mongodb)
Are the security claims valid?

- QE is an instance of **searchable encryption (SE)** scheme.
- **No security proof** is available yet.
How does QE work?
Simplified token generation

Figure 1: Simplified QE Token Derivation.  

[GPT23]
### Overview of QE (oversimplified)

Encrypted search index (ESC)

<table>
<thead>
<tr>
<th>_id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1^{esc}$</td>
<td>$\text{PRF}_{K_1}(\text{&quot;Nuclear Technician&quot;} | 1)$</td>
</tr>
<tr>
<td>$N_2^{esc}$</td>
<td>$\text{PRF}_{K_1}(\text{&quot;Nuclear Technician&quot;} | 2)$</td>
</tr>
<tr>
<td>Enc$K_2(1)$</td>
<td>Enc$K_2(2)$</td>
</tr>
</tbody>
</table>

$N$: “Nuclear Technician”

Omit doc3 (“CEO”) for simplification
Overview of QE (oversimplified)

Encrypted search index (ESC)

<table>
<thead>
<tr>
<th>_id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1^{esc}$</td>
<td>$\text{PRF}_{K_1}(\text{“Nuclear Technician”} | 1)$</td>
</tr>
<tr>
<td>$N_2^{esc}$</td>
<td>$\text{PRF}_{K_1}(\text{“Nuclear Technician”} | 2)$</td>
</tr>
</tbody>
</table>

$N$: “Nuclear Technician”

encrypted document

<table>
<thead>
<tr>
<th>_id</th>
<th>edoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{PRF}_{K_3}(1)$</td>
<td>edoc_1 = {</td>
</tr>
</tbody>
</table>
|               |   “Name”: ***,
|               |   “Job”: ***,
|               |   “Address”: ***
|               | }            |
| $\text{PRF}_{K_3}(2)$ | edoc_2 = {   |
|               |   “Name”: ***,
|               |   “Job”: ***,
|               |   “Address”: ***
|               | }            |
ESC size

Insert($doc_4$)
Insert($doc_5$)
Insert($doc_6$)

<table>
<thead>
<tr>
<th>_id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1^{esc}$</td>
<td>$Enc_{K_2}(1)$</td>
</tr>
<tr>
<td>$N_2^{esc}$</td>
<td>$Enc_{K_2}(2)$</td>
</tr>
<tr>
<td>$N_3^{esc}$</td>
<td>$Enc_{K_2}(3)$</td>
</tr>
<tr>
<td>$N_4^{esc}$</td>
<td>$Enc_{K_2}(4)$</td>
</tr>
<tr>
<td>$N_5^{esc}$</td>
<td>$Enc_{K_2}(5)$</td>
</tr>
</tbody>
</table>

$N_1^{esc} \leftarrow \text{PRF}_{K_1}(\text{“Nuclear Technician”}|| 1)$
$N_2^{esc} \leftarrow \text{PRF}_{K_1}(\text{“Nuclear Technician”}|| 2)$
...

Compaction

ESC

<table>
<thead>
<tr>
<th>_id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{1}^{esc}$</td>
<td>$\text{Enc}<em>{K</em>{2}}(1)$</td>
</tr>
<tr>
<td>$N_{2}^{esc}$</td>
<td>$\text{Enc}<em>{K</em>{2}}(2)$</td>
</tr>
</tbody>
</table>

Compact

ESC'

<table>
<thead>
<tr>
<th>_id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{null}^{esc}$</td>
<td>$\text{Enc}<em>{K</em>{2}}(2)$</td>
</tr>
</tbody>
</table>

$K_1, K_2$
“Security” of QE

QE satisfies snapshot security of searchable encryption.
“Security” of QE

QE satisfies **snapshot security** of searchable encryption.
How was QE implemented in MongoDB?

```
#ifinclude "mongo/db/fle_crud.h"
#include <string>
#include <utility>

#include "mongo/bson/bsonelement.h"
#include "mongo/bson/bsonmisc.h"
#include "mongo/bson/bsonobj.h"
#include "mongo/bson/bsonobjbuilder.h"
#include "mongo/bson/bsontypes.h"
#include "mongo/crypto/encryption_fields_gen.h"
#include "mongo/crypto/fle_crypto.h"
#include "mongo/db/fle_crud.h"

#include "mongo/db/namespace_string.h"
#include "mongo/db/ops/write_ops_gen.h"
#include "mongo/db/ops/write_ops_parsers.h"
#include "mongo/db/query/find_command_gen.h"
#include "mongo/db/query/fle/server_rewrite.h"
#include "mongo/db/repl/repl_client_info.h"
#include "mongo/db/session/session.h"
#include "mongo/db/session/session_catalog.h"
#include "mongo/db/session/session_catalog_mongod.h"
#include "mongo/db/transaction/transaction_api.h"
#include "mongo/db/transaction/transaction_participant.h"
#include "mongo/db/transaction/transaction_participant_resource_yielder.h"
#include "mongo/executor/network_interface_factory.h"
#include "mongo/executor/thread_pool_task_executor.h"
#include "mongo/idl/idl_parser.h"
#include "mongo/s/grid.h"
#include "mongo/s/transaction_router_resource_yielder.h"
#include "mongo/s/write_ops/batch_write_executor.h"
#include "mongo/util/assert_util.h"
#include "mongo/util/concurrency/thread_pool.h"
```

“fle_crud_mongod.cpp”

Implementing core read and write functionality used by QE

24 out of 29 are native MongoDB library headers
System integration of QE

- QE is built using **native** MongoDB operations.
- QE **interacts** with other MongoDB system components.
- MongoDB has adopted a **cost-effective** approach integrating QE: incurring minimal changes to the existing system.
System integration of QE
System integration of QE
System integration of QE

Grubbs et al. [GRS17]
Logging system

OpLog: data consistency in deployment

QE command

write_ops::FindAndModifyCommandReply processFLEFindAndModify()

uassert(6371800,
    "Encrypted index operations are only supported on replica sets",
    repl::ReplicationCoordinator::get(opCtx->getServiceContext())->getReplicationMode() ==
    repl::ReplicationCoordinator::modeReplSet);

Interacts with MongoDB’s OpLog for replication

"fle_crud_mongod.cpp"
What does the 🍁 look like?
What does the 🌳 say?
What does the leakage say?
How we extract the leakage from OpLog

Insert(doc₁)
Insert(doc₂)
Insert(doc₃)
Compact()

{ "Delete": "ESC"
  "_id": \( N_{1}^{esc} \),
  "txnid": "211" }

\( (N_{1}^{esc}, N_{2}^{esc}) \)

{ "Delete": "ESC"
  "_id": \( N_{2}^{esc} \),
  "txnid": "211" }

{ "Insert": "ESC"
  "_id": \( N_{null}^{esc} \),
  "txnid": "211" }

\( (C_{1}^{esc}) \)

{ "Delete": "ESC"
  "_id": \( C_{1}^{esc} \),
  "txnid": "212" }

{ "Insert": "ESC"
  "_id": \( C_{null}^{esc} \),
  "txnid": "212" }
Inference attack

- The leakage we have extracted corresponds to **frequency** and **correlation** leakage.
- Auxiliary information
- **New inference attack techniques**, based on Gui et al. [GPP21]

\[ \text{edoc}_1, \text{edoc}_2 \text{ contains the same field value} \]
\[ \text{edoc}_3 \text{ has a different one} \]

```
{
    "Name": "Homer Simpson",
    "Job": "Nuclear Technician",
    "Class of Jobs": "Technician",
    "Address": "742 Evergreen Terrace"
}
```

"Nuclear Technician"  "CEO"
Experimental validation

- **Auxiliary information**: ACS (American Community Survey micro-data) 2012
- **Recovery target**: ACS 2013
- Simulated leakage
- Artifact available!
How and when to get this “leaky” OpLog?

- OpLog is stored on the server’s file system.
- After `Compact()`
Takeaways

System integration of searchable encryption schemes is challenging!
Paper & artifact

[GPT23] Zichen Gui, Kenneth G. Paterson, and Tianxin Tang
(tianxin.tang@inf.ethz.ch)
References

- [GRS17] Paul Grubbs, Thomas Ristenpart, Vitaly Shmatikov, Why Your Encrypted Database Is Not Secure
- [GPP21] Zichen Gui, Kenneth G Paterson, Sikhar Patranabis, Rethinking Searchable Symmetric Encryption