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.

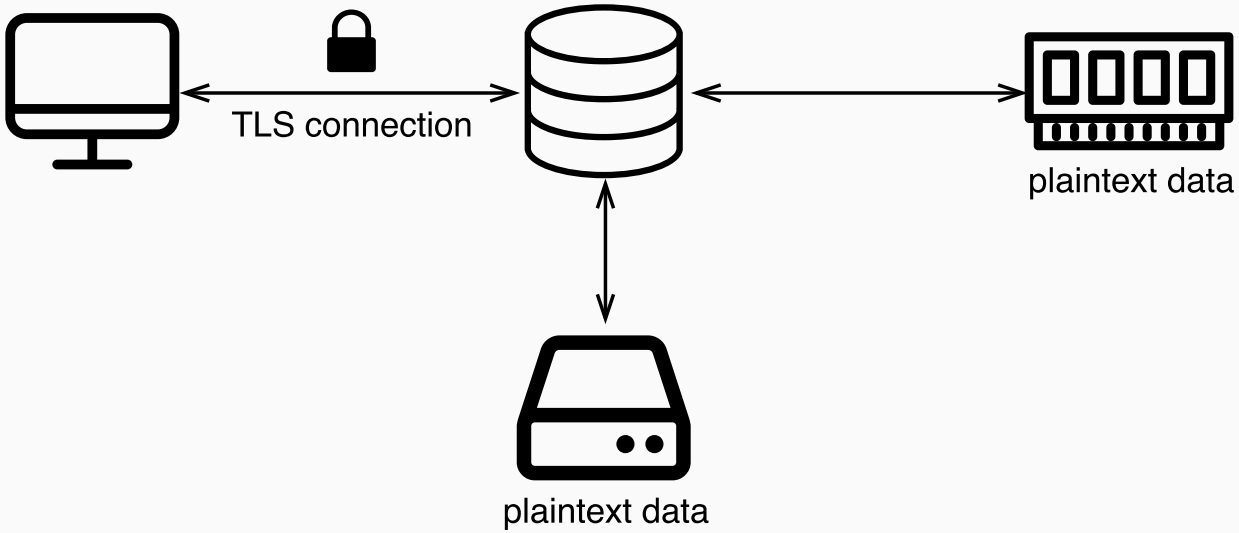
Necessary because:

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

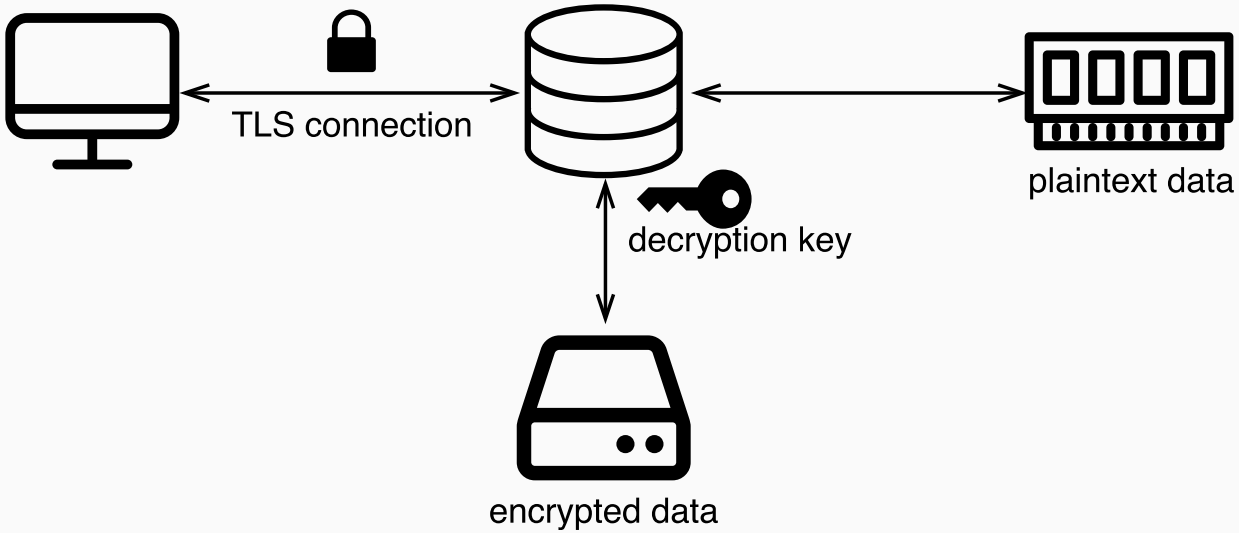
Solution

Encrypt your plain-text data.

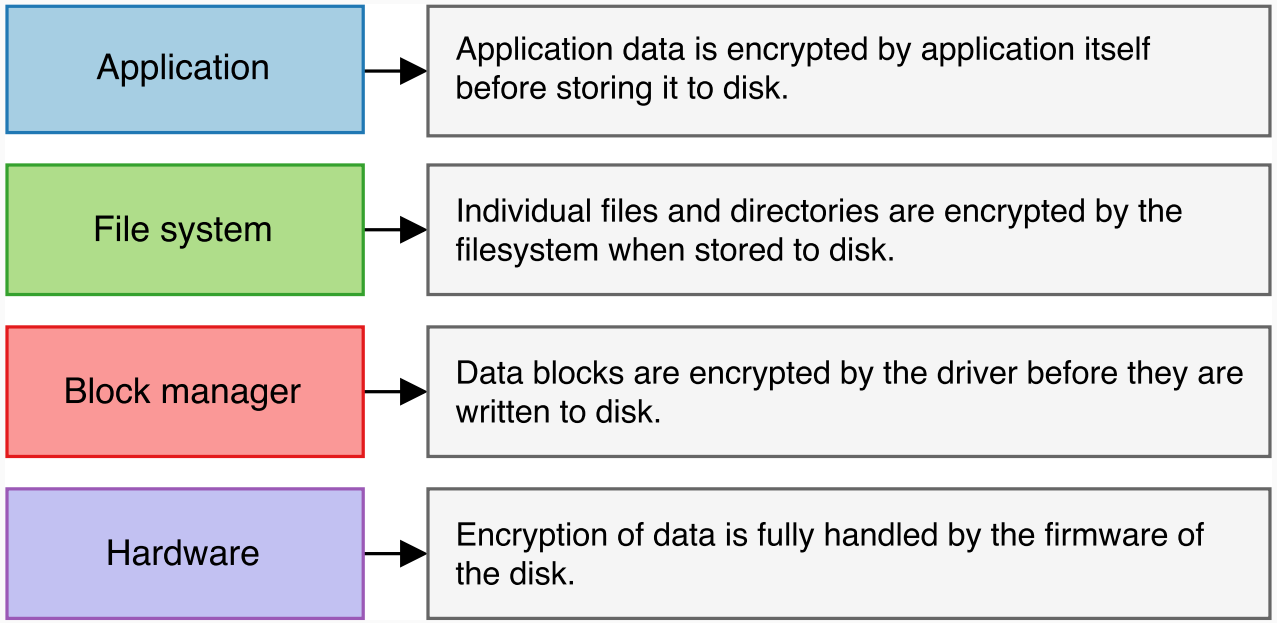
Introduction
Plain data



Introduction
Encryption at rest



Introduction
Encryption at rest

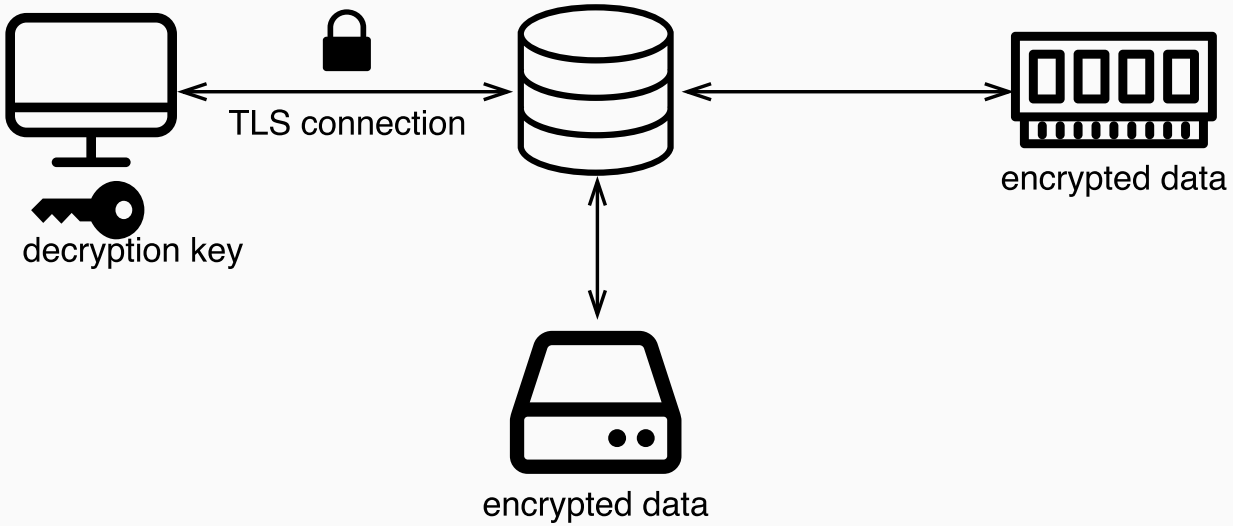


- How is SQL-aware encryption realised in NoSQL database engines?
  - What kind of security does it provide?
  - How does it compare to encryption at rest?
- What is the performance impact of enabling encryption?
  - What limitations are their in terms of functionality?

Computation over encrypted data

Computation over encrypted data

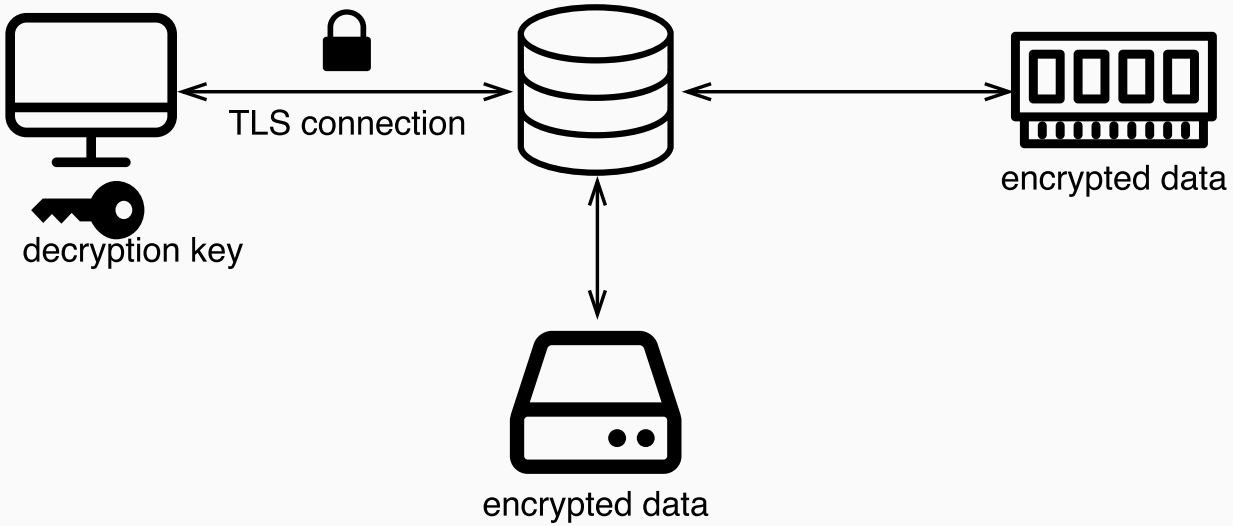
End-to-end encrypted database



- Key stored at client.
- Encryption and decryption by client (end-to-end).

Computation over encrypted data

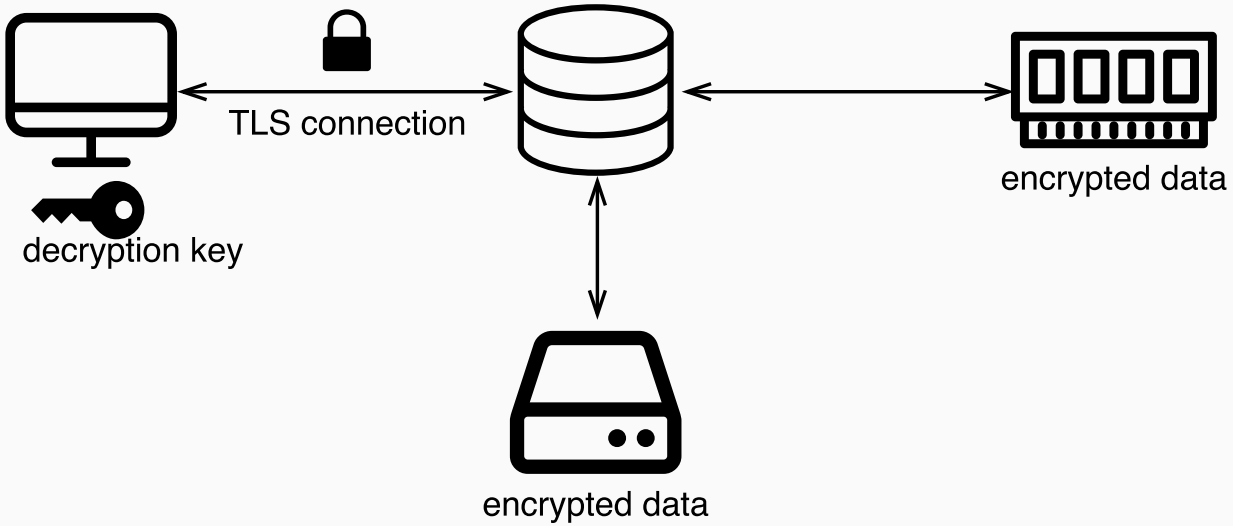
End-to-end encrypted database



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

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?
- 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.

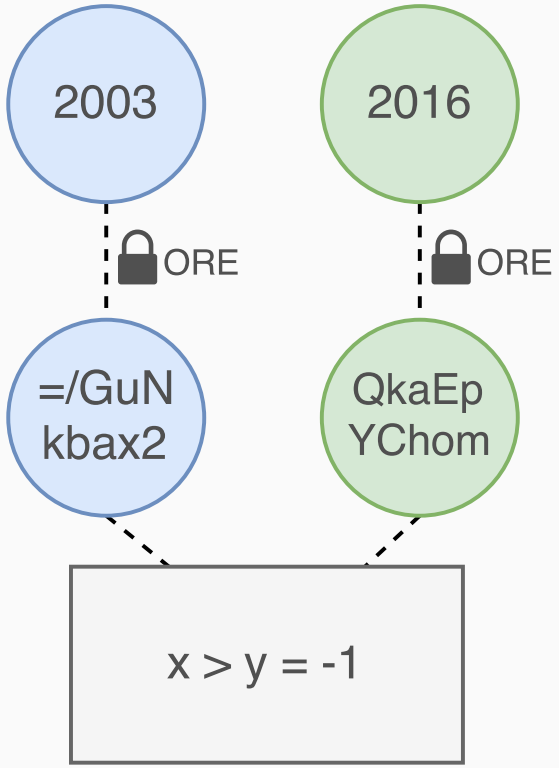
$$E(m_1) * E(m_2) = E(m_1 * m_2)$$

Computation over encrypted data

Order Revealing Encryption

Public compare function on encrypted data.

	-1	smaller
$x > y$	**0**	equal
	1	greater



SecureMongo

SecureMongo

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

SecureMongo

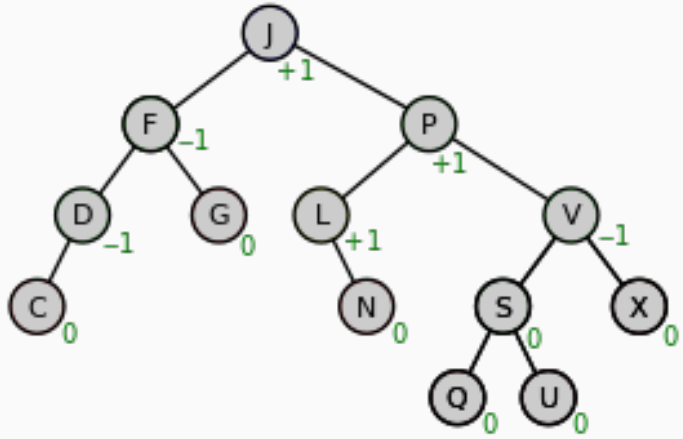
- Based on work by Alves et al.
- Python connector wrapper.
- Logic at client side.
- End-to-end encrytption with queries on encrypted data.

Our work:

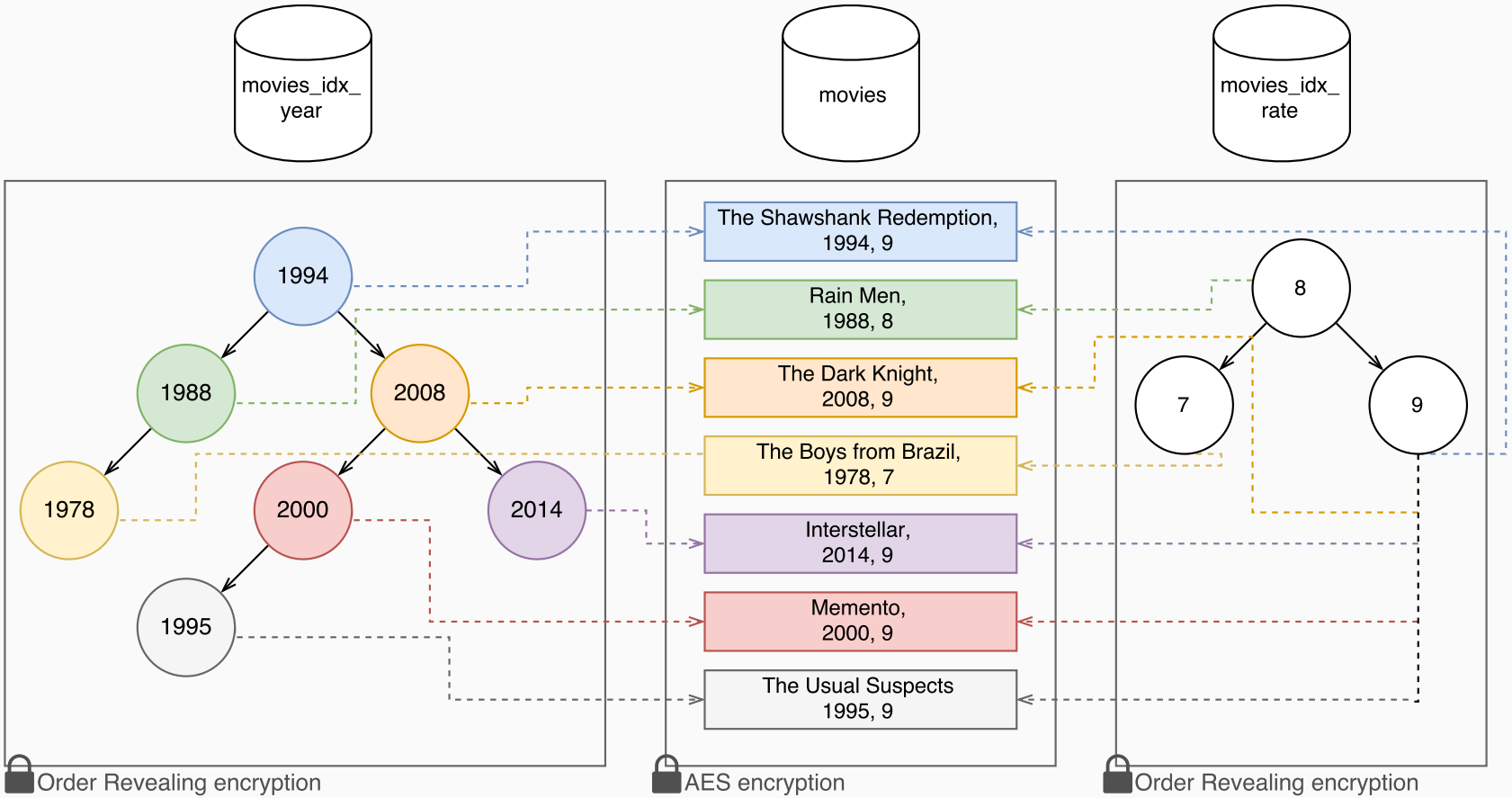
- Sequential inserts.
- Serialized AVL tree.
- Tree balancing at server side.

Self-balancing binary search tree.

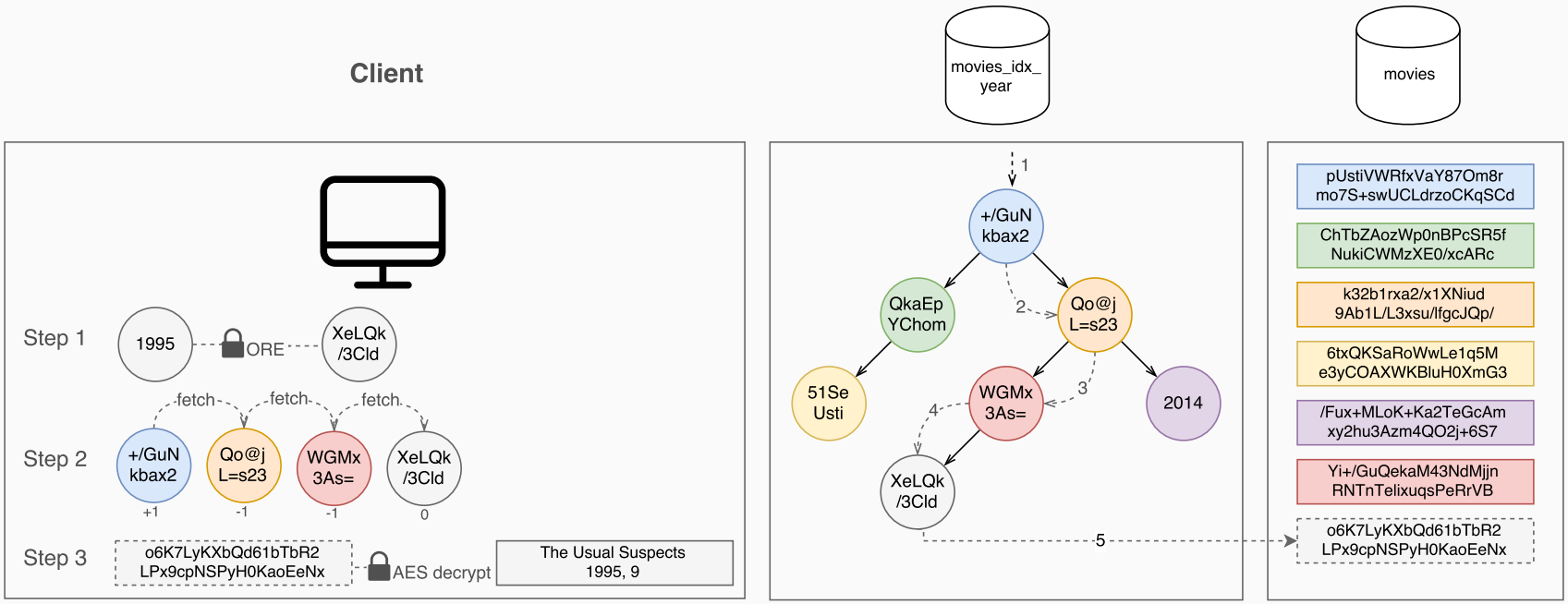
Algorithm	Average	Worst Case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(\log n)$
Insert	$O(\log n)$	$O(\log n)$
Delete	$O(\log n)$	$O(\log n)$



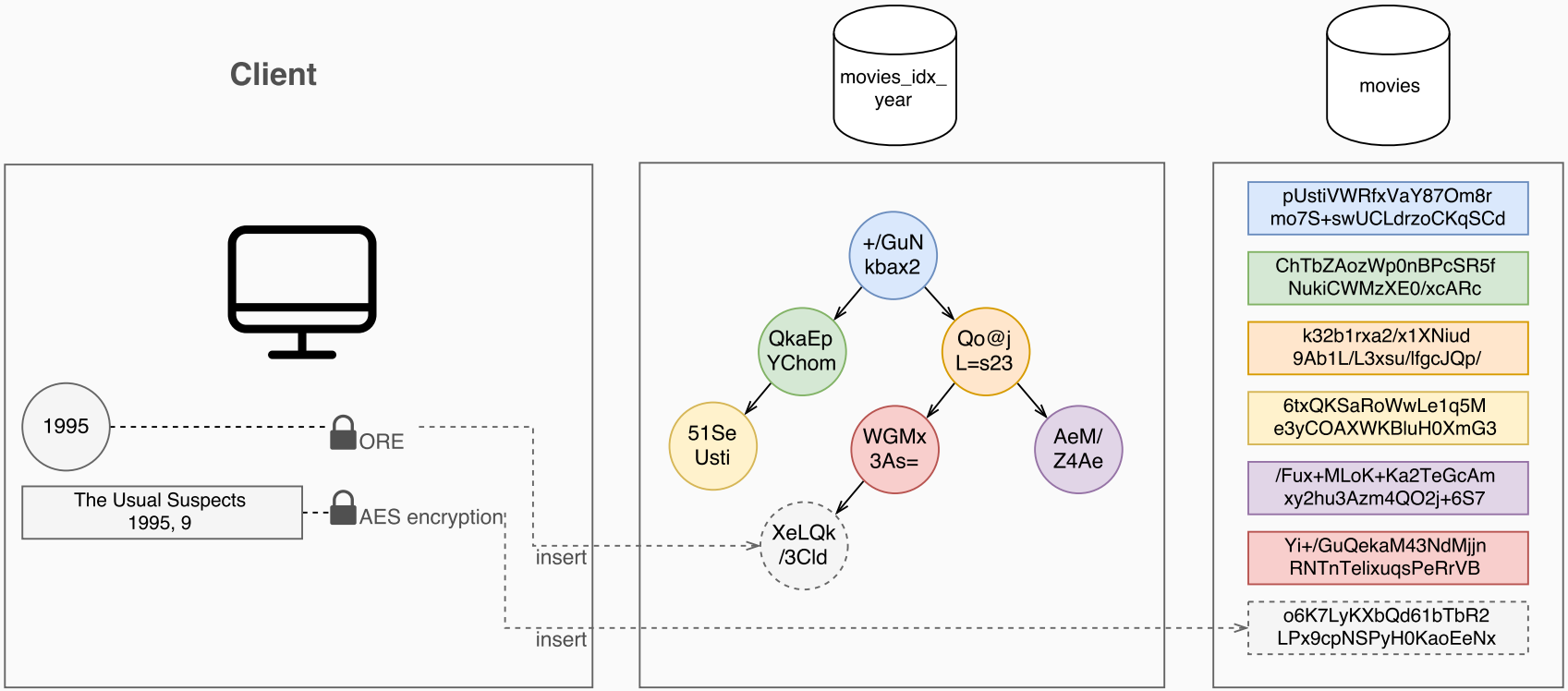
SecureMongo
overview



SecureMongo
selection



SecureMongo
insertion



Method

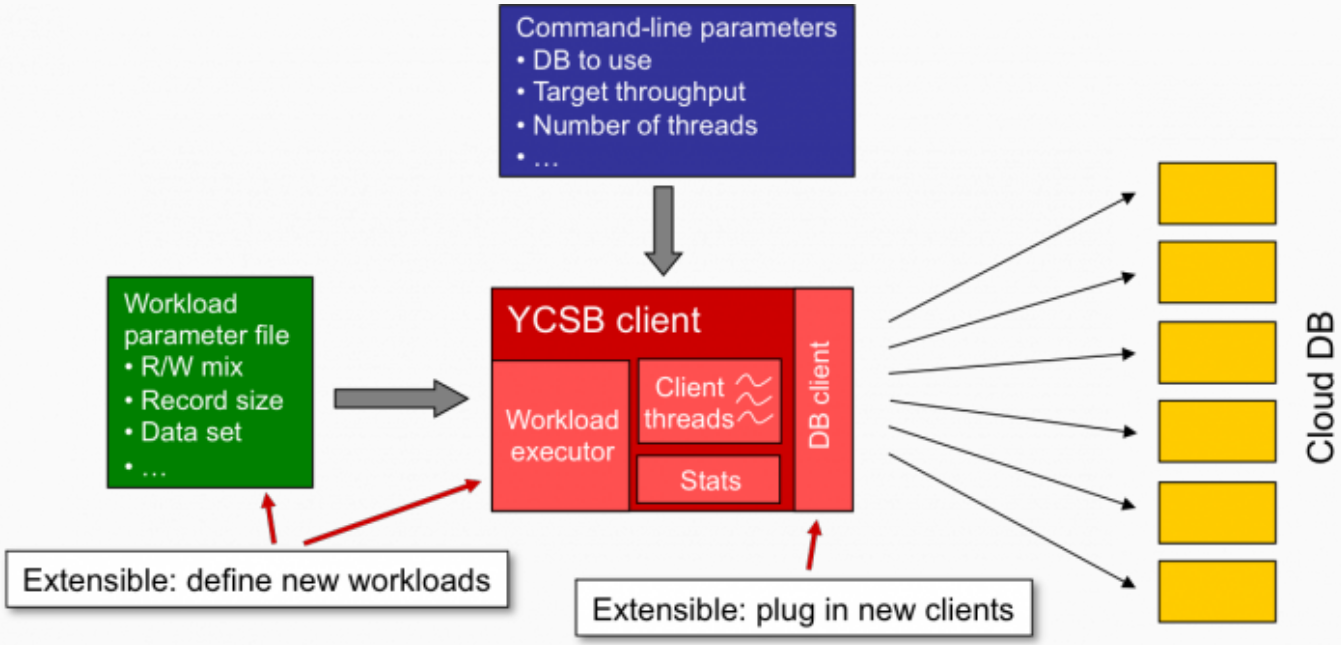
Method
Our work

- 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



Method

Plain vs. encryption at rest

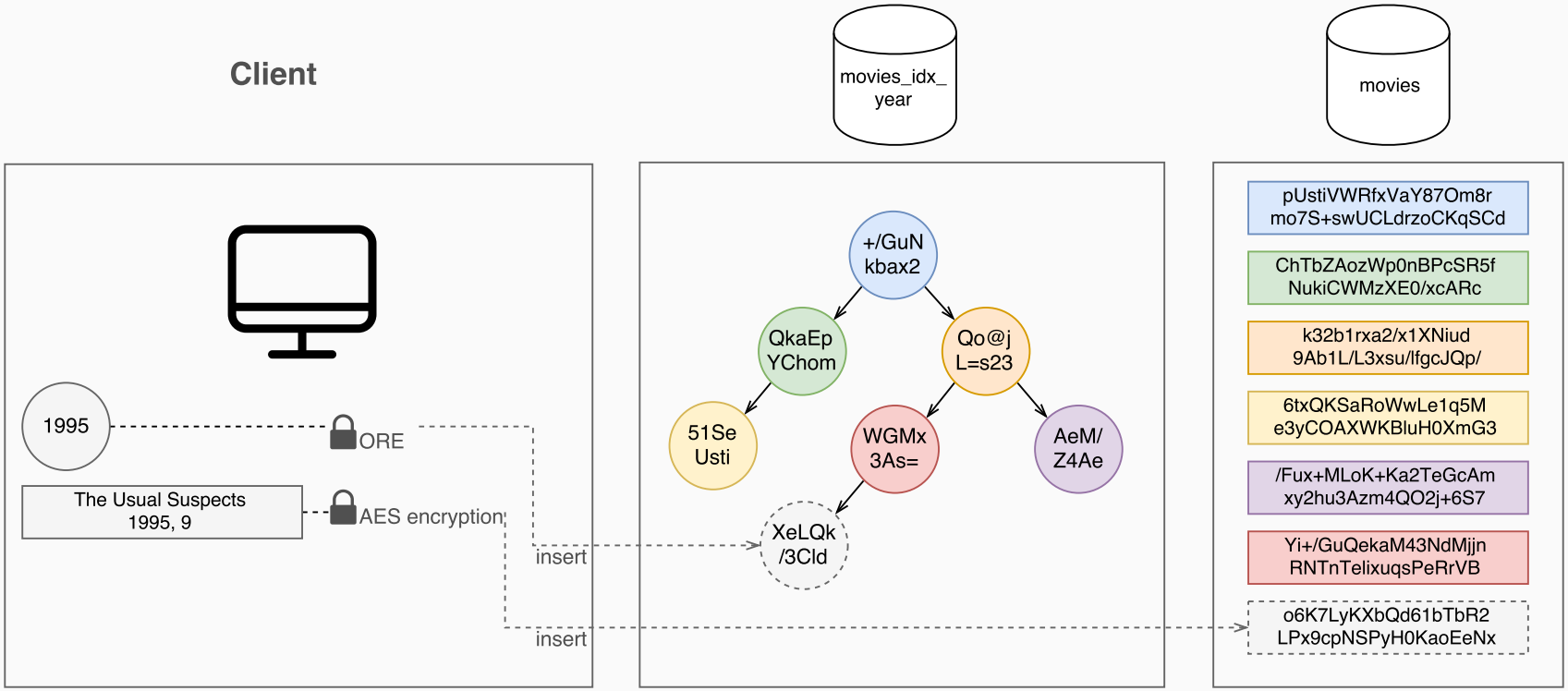
- YCSB default core workload.
- Adjustable with parameters.
- Can extend framework with alternative workloads.

recordcount	16,000,000
operationcount	100,000
readproportion	0.5
updateproportion	0.5

Method

Plain vs. computation over encrypted data

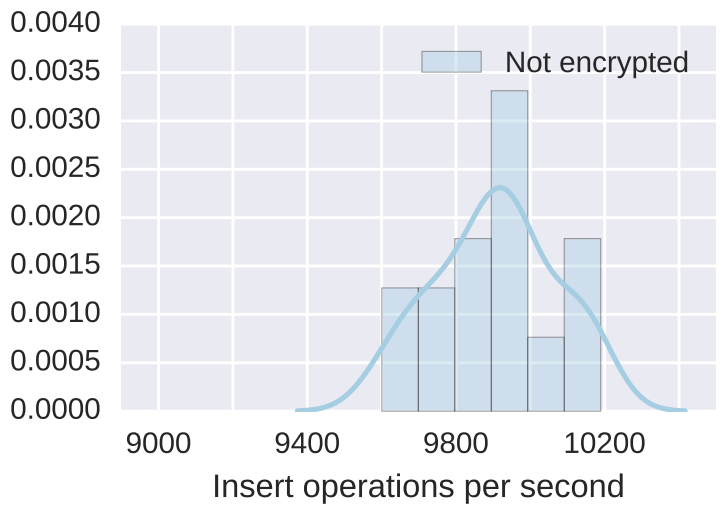
- BenchmarkDB
- Python framework
- IMDB movies

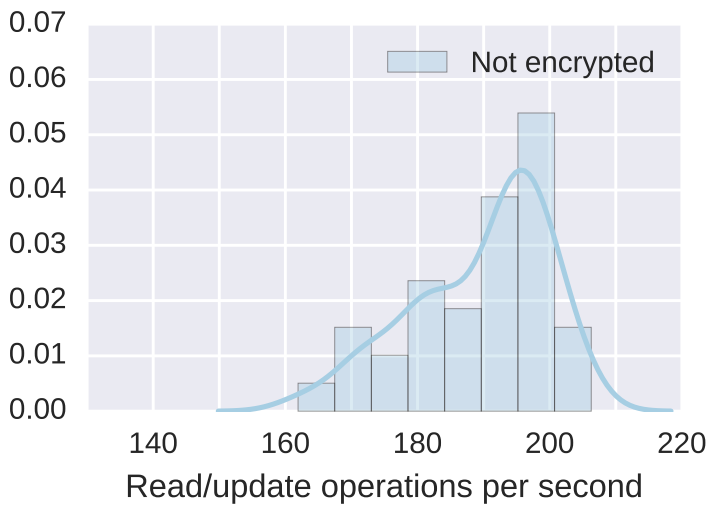


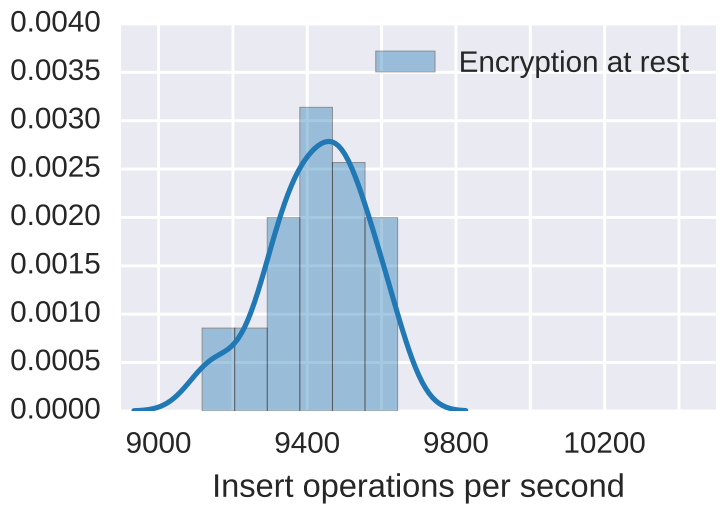
Results encryption at rest

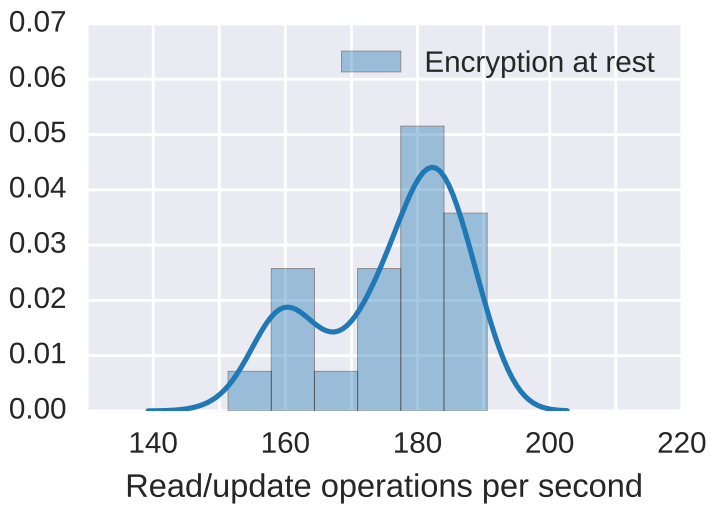
Results

Performance encryption at rest



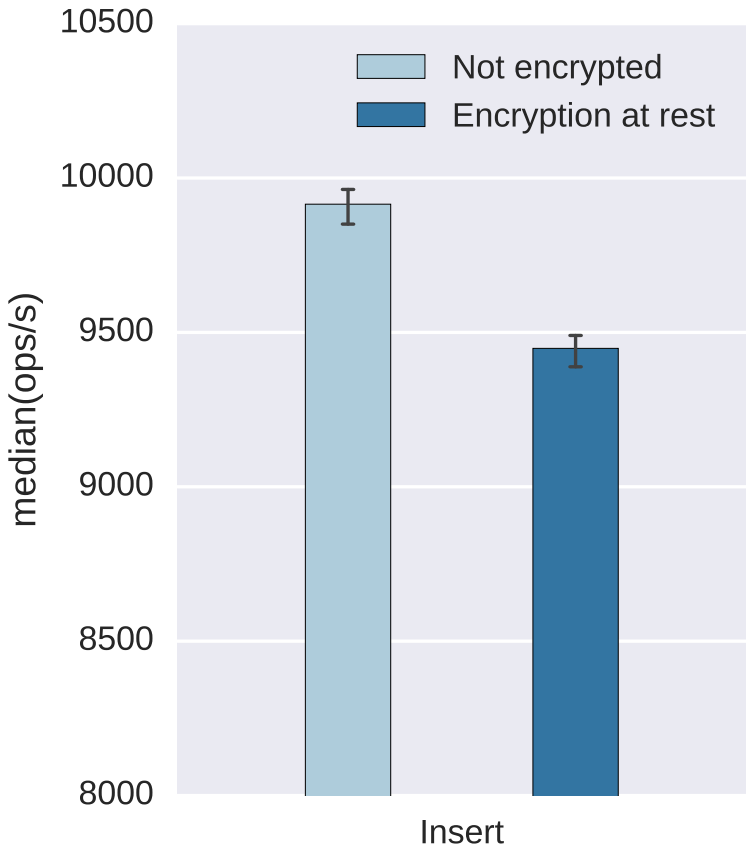


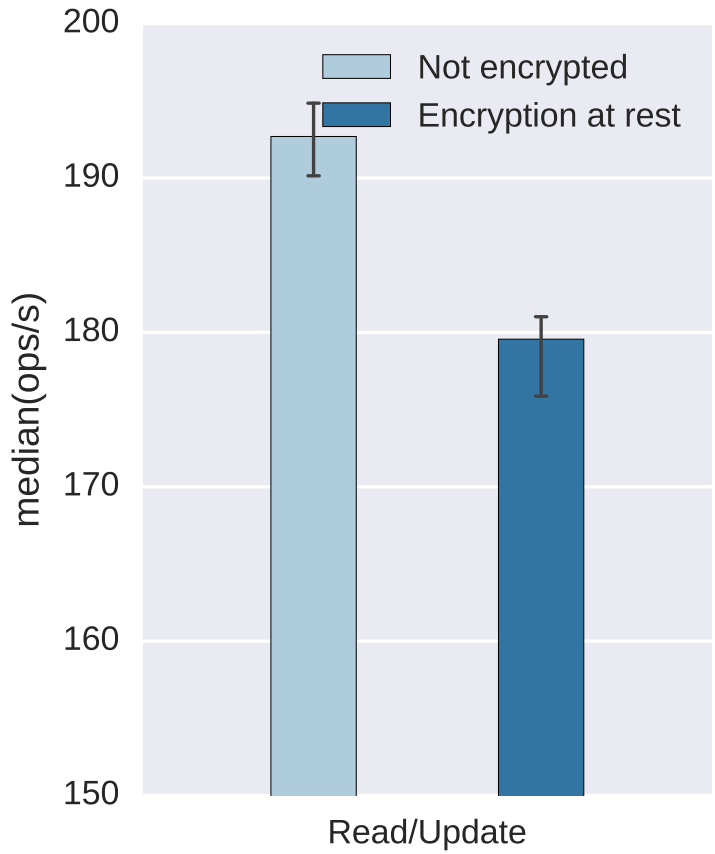




Results

Performance encryption at rest





Insert

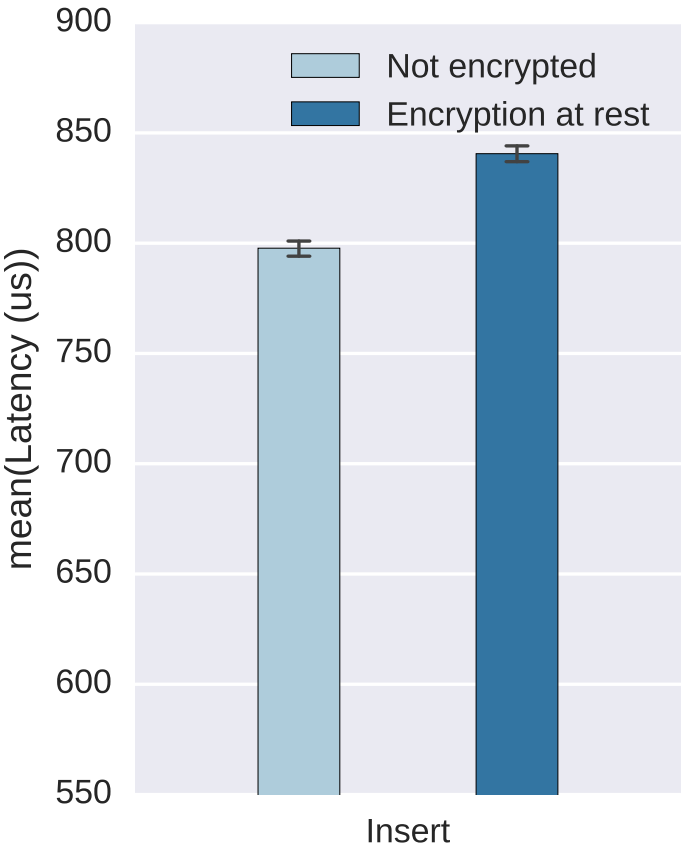
4.9% lower throughput

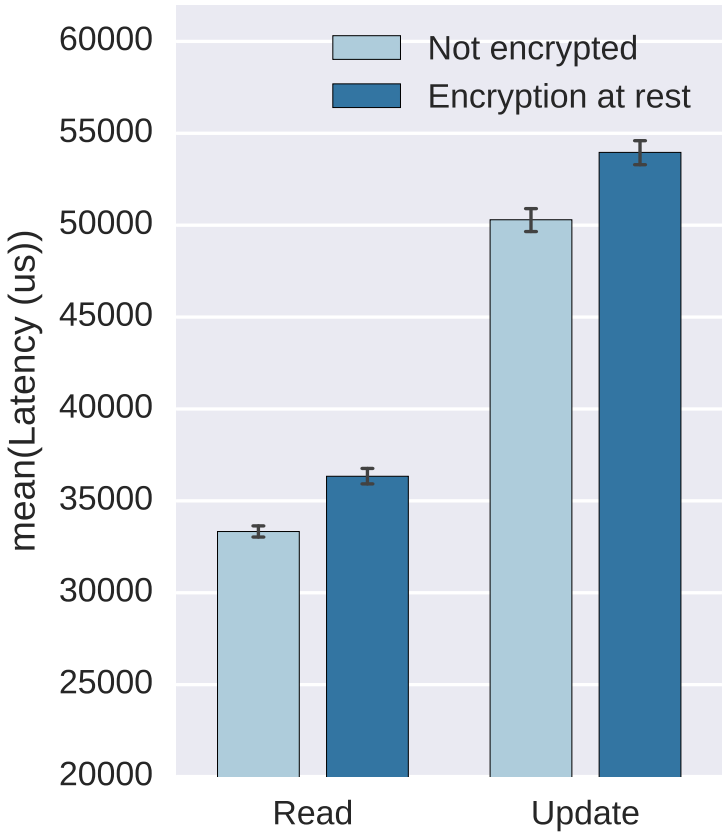
Read/Update

7.3% lower throughput

Results

Performance encryption at rest



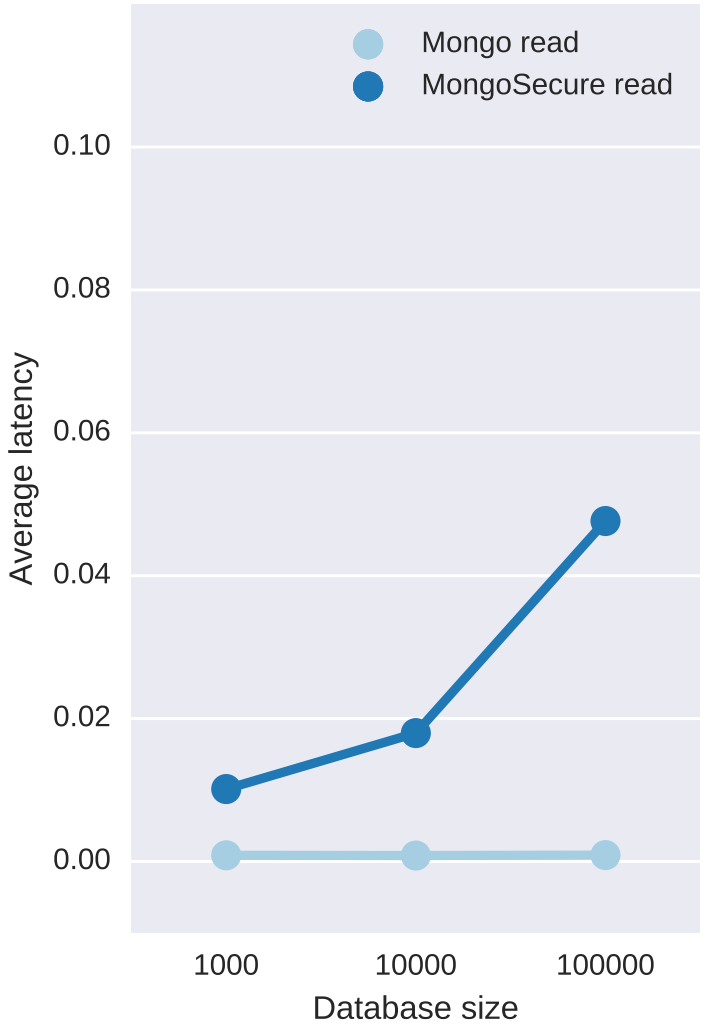


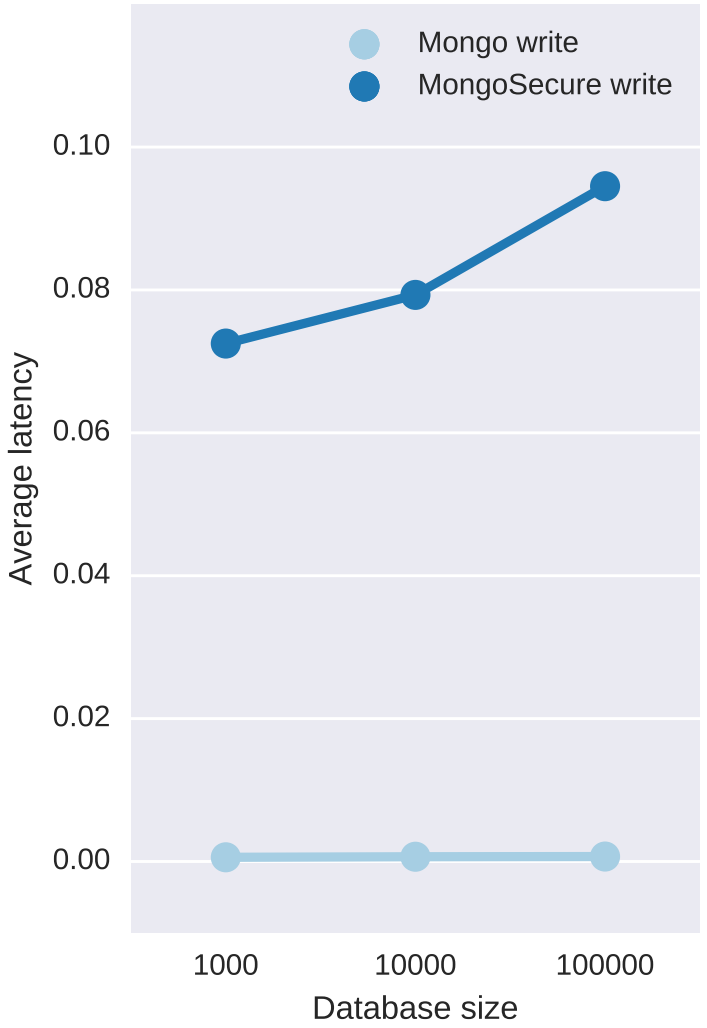
Insert	**Read**	**Update**
5.2% slower	7.4% slower	7.5% slower

Results SecureMongo

Results

Performance SecureMongo

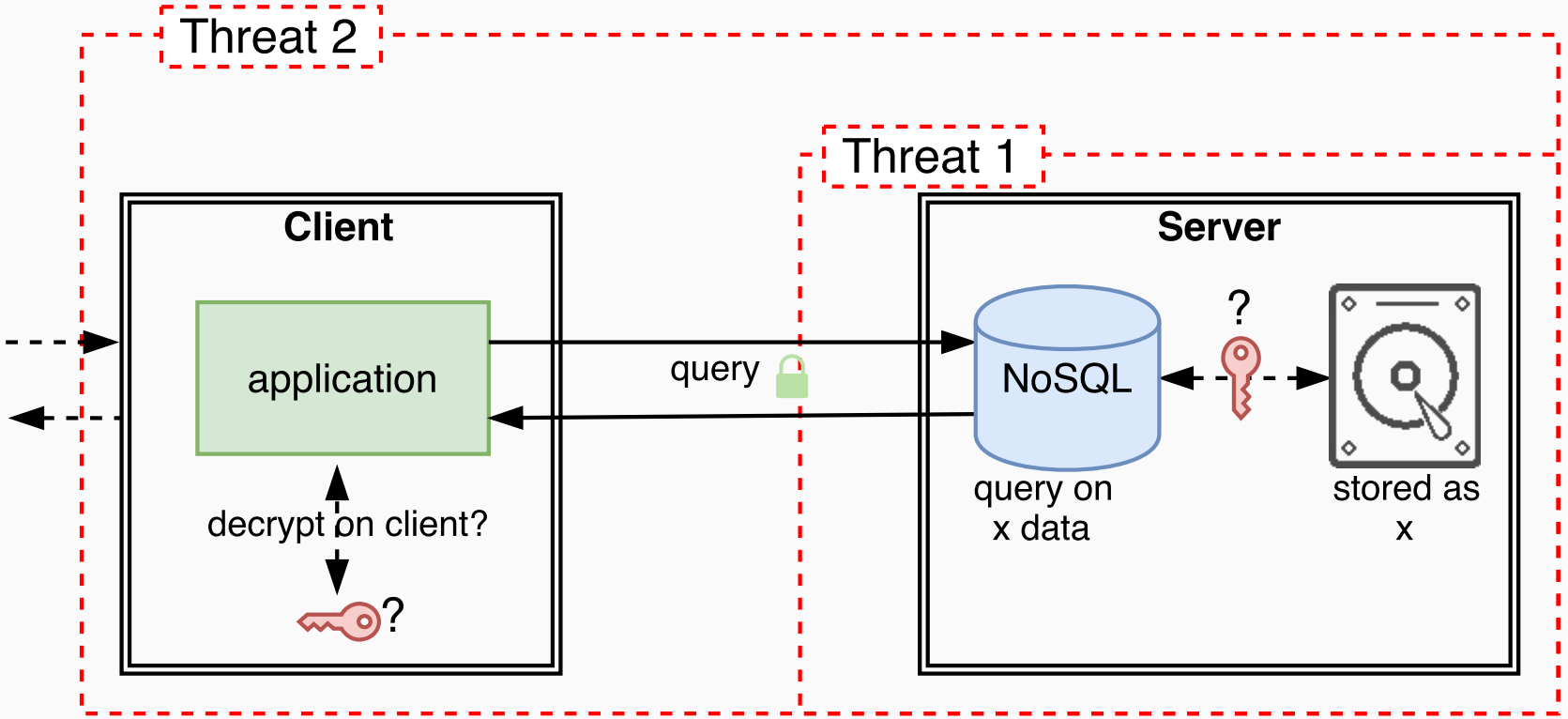




Results security

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

Issue

Key is continuously needed on server.

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

Issue

The plain set-up is still utterly compromised.

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

Issue

Key is continuously needed by the application.

Conclusion

Conclusion

Solution

Encrypt your plain-text data.

Conclusion

Solution

Encrypt your plain-text data.✓

Conclusion

Solution

Encrypt your plain-text data.✓

TradeOff

Security $\leftrightarrow$ Performance

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?