K9db: Privacy-Compliant Storage For Web Applications By Construction

Kinan Dak Albab ● Ishan Sharma ● Justus Adam ● Benjamin Kilimnik ● Aaron Jeyaraj ● Raj Paul ● Artem Agvanian ● Leonhard Spiegelberg ● Malte Schwarzkopf
Privacy laws are important

GDPR

CCPA
Developers must ensure their applications comply with Privacy Laws

Must comply with GDPR
Developers must ensure their applications comply with Privacy Laws.

1. Allow users to request access/deletion of data.
Developers must ensure their applications comply with Privacy Laws

1. Allow users to request access/deletion of data
2. Secure data → encryption at rest
Developers must ensure their applications comply with Privacy Laws

1. Allow users to request access/deletion of data
2. Secure data → encryption at rest
3. Out of scope: consent, cookie banners, purpose limitations
Compliance is challenging for developers

Alice → Delete my data → Application → Must comply with GDPR

Application Developer
Compliance is challenging for developers

1. Identify Alice’s data in complex schema
Compliance is challenging for developers

1. Identify Alice’s data in complex schema
2. Handle shared data

Application w/ friends

chat message vs medical records

shared w/ doctor

Application

Delete my data

Must comply with

GDPR

Alice

Developer

Application
Compliance is challenging for developers

1. Identify Alice’s data in complex schema
2. Handle shared data
3. Retain some data with anonymization

Application Developer

retain anonymized financial transactions for taxes

Alice

Delete my data

Application

Must comply with GDPR

Application

Must comply with GDPR

GDPR
Compliance is challenging for developers

1. Identify Alice’s data in complex schema
2. Handle shared data
3. Retain some data with anonymization
4. Update cache, backups

Alice 
Application 
Must comply with 
GDPR
We need a better way...

1. Identify Alice’s data in complex schema
2. Handle shared data
3. Retain some data with anonymization
4. Update cache, backups

HELP!!!!!!

Traditional DBs are unhelpful
K9db: compliance by construction

Delete Alice’s data

Ensures compliance

Application

Delete my data

Application

Application Developer

I ❤️ K9db!

Alice

GDPR
K9db Goals

• Help developers get compliance right

• Low developer effort

• Performance comparable to widely-used SQL databases
K9db Goals

• Help developers get compliance right

• Low developer effort

• Performance comparable to widely-used SQL databases

Assumption:
Developers are honest but fallible
Penalties deter malicious behavior
Challenges

- Capture application-specific compliance policy
- Correctly handle access/deletion requests and enforce compliance invariants
- Maintain good performance
Challenges

- Capture application-specific compliance policy
- Correctly handle access/deletion requests and enforce compliance invariants
- Maintain good performance

- Schema annotations
- Data Ownership Graph
Challenges

- Capture application-specific compliance policy
- Correctly handle access/deletion requests and enforce compliance invariants
- Maintain good performance

- Schema annotations
- Data Ownership Graph
Application scenario: direct messages

```sql
CREATE TABLE users (  
    name TEXT PRIMARY KEY  
);

CREATE TABLE messages (  
    body TEXT,  
    sender TEXT REFERENCES users(name),  
    receiver TEXT REFERENCES users(name)  
);
```
Application scenario: direct messages

```
CREATE TABLE users (  
    name TEXT PRIMARY KEY
);

CREATE TABLE messages (  
    body TEXT,  
    sender TEXT REFERENCES users(name),  
    receiver TEXT REFERENCES users(name)
);
```

<table>
<thead>
<tr>
<th>users</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td></td>
</tr>
</tbody>
</table>
Application scenario: direct messages

```sql
CREATE DATA_SUBJECT TABLE users (
  name TEXT PRIMARY KEY
);

CREATE TABLE messages ( 
  body TEXT, 
  sender TEXT REFERENCES users(name), 
  receiver TEXT REFERENCES users(name) 
);
```

<table>
<thead>
<tr>
<th>users</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td></td>
</tr>
</tbody>
</table>
Application scenario: direct messages

```
CREATE DATA_SUBJECT TABLE users (
  name TEXT PRIMARY KEY
);

CREATE TABLE messages (
  body TEXT,
  sender TEXT REFERENCES users(name),
  receiver TEXT REFERENCES users(name)
);
```

Alice requested deletion

<table>
<thead>
<tr>
<th>messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Compliance policy is application specific

Who owns each message?

CREATE DATA_SUBJECT TABLE users (  
  name TEXT PRIMARY KEY  
);

CREATE TABLE messages (  
  body TEXT,  
  sender TEXT REFERENCES users(name),  
  receiver TEXT REFERENCES users(name)  
);

Alice requested deletion

<table>
<thead>
<tr>
<th>messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Compliance policy is application specific

Chosen policy: Joint-ownership

Alice requested deletion

CREATE DATA_SUBJECT TABLE users (  
  name TEXT PRIMARY KEY  
);

CREATE TABLE messages (  
  body TEXT,  
  sender TEXT REFERENCES users(name),  
  receiver TEXT REFERENCES users(name)  
);
Developers express policy via schema annotations

Alice requested deletion

→ K9db retains messages because they are shared with others

<table>
<thead>
<tr>
<th>messages</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>sender</td>
<td>receiver</td>
</tr>
<tr>
<td>...</td>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>...</td>
<td>Alice</td>
<td>Carol</td>
</tr>
<tr>
<td>...</td>
<td>Bob</td>
<td>Carol</td>
</tr>
</tbody>
</table>
Developers express policy via schema annotations

```sql
CREATE DATA_SUBJECT TABLE users (  
   name TEXT PRIMARY KEY
);
CREATE TABLE messages (  
   body TEXT,  
   sender TEXT OWNED_BY users(name),  
   receiver TEXT OWNED_BY users(name)
);
```

Bob requested deletion

→ K9db only deletes the first message because both Alice and Bob are gone

<table>
<thead>
<tr>
<th>messages</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>sender</td>
<td>receiver</td>
</tr>
<tr>
<td>...</td>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>...</td>
<td>Alice</td>
<td>Carol</td>
</tr>
<tr>
<td>...</td>
<td>Bob</td>
<td>Carol</td>
</tr>
</tbody>
</table>
Developers express policy via schema annotations

- K9db provides more schema annotations (see paper)

Developer can express alternative policies:
1. Delete when either sender or receiver deletes
2. Delete only when sender deletes
Developers express policy via schema annotations

- K9db provides more schema annotations (see paper)

- K9db provides **EXPLAIN COMPLIANCE** command to help developers reason about their policy and annotations

Developer can express alternative policies:
1. Delete when either sender or receiver deletes
2. Delete only when sender deletes
Data Ownership Graph (DOG 🐶)

CREATE DATA_SUBJECT TABLE users (
    name TEXT PRIMARY KEY
);

CREATE TABLE messages (
    body TEXT,
    sender TEXT OWNED_BY users(name),
    receiver TEXT OWNED_BY users(name)
);
Data Ownership Graph (DOG 🐶)

```sql
CREATE DATA_SUBJECT TABLE users (  
    name TEXT PRIMARY KEY
);

CREATE TABLE messages (  
    body TEXT,  
    sender TEXT OWNED_BY users(name),  
    receiver TEXT OWNED_BY users(name)
);
```
Data Ownership Graph (DOG 🐶)

CREATE DATA_SUBJECT TABLE users (name TEXT PRIMARY KEY);

CREATE TABLE messages (body TEXT,
sender TEXT OWNED_BY users(name),
receiver TEXT OWNED_BY users(name));

sender

messages

n

messages

n

users

1

1

receiver
Data Ownership Graph (DOG 🐶)

CREATE DATA_SUBJECT TABLE users (  
  name TEXT PRIMARY KEY  
);  

CREATE TABLE messages (  
  body TEXT,  
  sender TEXT OWNED_BY users(name),  
  receiver TEXT OWNED_BY users(name)  
);
Data Ownership Graph (DOG 🐶)

- Taggings
- Stories
- Users
- Messages

Relations:
- Taggings: n → Stories
- Stories: 1 → Users
- Users: 1 → Messages
- Messages: n → Users
Data Ownership Graph (DOG 🐶)

- **taggings**: n
- **stories**: n
- **users**: 1
- **messages**: n
- **sender**: 1
- **receiver**: 1
- **author**: 1
- **shared data**: }

- **story**: 1

- **Data Ownership Graph (DOG 🐶)**

- **taggings**: n
- **stories**: n
- **users**: 1
- **messages**: n
- **sender**: 1
- **receiver**: 1
- **author**: 1
- **shared data**: ∅
Data Ownership Graph (DOG 🐶)

- Taggings
- Stories
- Users
- Messages

Transitive chain

Stories → Users
Sender → Messages
Receiver → Messages
Author → Users

Story → Transitive chain

n:n
1:1
Data Ownership Graph (DOG)

- Specifies how to handle **access** and **deletion** requests

Diagram:
- Users
  - Author 1
  - Sender 1
  - Receiver 1
- Messages
  - Sender n
  - Receiver n
- Stories
  -Author n
  -Sender 1
- Taggings
  -Story n
DOG identifies the owners of every row of every story.
DOG identifies data owned by data subject
Challenges

- Capture application-specific compliance policy
  - Schema annotations
  - Data Ownership Graph

- Correctly handle access/deletion requests and enforce compliance invariants

- Maintain good performance
Challenges

- Capture application-specific compliance policy
- Correctly handle access/deletion requests and enforce compliance invariants
- Maintain good performance
Ownership-aware storage: per-user μDBs

K9db organizes data by owner(s)

<table>
<thead>
<tr>
<th>messages</th>
<th>body</th>
<th>sender</th>
<th>receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>Alice</td>
<td>Bob</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>messages</th>
<th>body</th>
<th>sender</th>
<th>receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bob</td>
<td>Carol</td>
</tr>
</tbody>
</table>

Alice μDB

Bob μDB
Ownership-aware storage: per-user μDBs

K9db organizes data by owner(s)

<table>
<thead>
<tr>
<th>messages</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>sender</td>
<td>receiver</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Alice</td>
<td>Bob</td>
<td></td>
</tr>
</tbody>
</table>

Application

Alice μDB

Bob μDB

Who owns this?

Alice and Carol

Alice

Carol

Alice

Bob

Bob

Carol
Ownership-aware storage: per-user $\mu$DBs

K9db organizes data by owner(s)

Who owns this?

Alice and Carol

Application

K9db

Alice $\mu$DB

Bob $\mu$DB

<table>
<thead>
<tr>
<th>messages</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>sender</td>
<td>receiver</td>
</tr>
<tr>
<td>...</td>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>...</td>
<td>Alice</td>
<td>Carol</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>messages</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>sender</td>
<td>receiver</td>
</tr>
<tr>
<td>...</td>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>...</td>
<td>Bob</td>
<td>Carol</td>
</tr>
</tbody>
</table>
Ownership-aware storage: per-user μDBs

K9db stores copies of jointly-owned data

<table>
<thead>
<tr>
<th>messages</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>sender</td>
<td>receiver</td>
</tr>
<tr>
<td>...</td>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>...</td>
<td>Alice</td>
<td>Carol</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>messages</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>sender</td>
<td>receiver</td>
</tr>
<tr>
<td>...</td>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>...</td>
<td>Bob</td>
<td>Carol</td>
</tr>
</tbody>
</table>
Advantage 1: $\mu$DBs provide correct deletion by construction

Delete Alice’s data

<table>
<thead>
<tr>
<th>messages</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>sender</td>
<td>receiver</td>
</tr>
<tr>
<td></td>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td></td>
<td>Alice</td>
<td>Carol</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>messages</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>sender</td>
<td>receiver</td>
</tr>
<tr>
<td>...</td>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>...</td>
<td>Bob</td>
<td>Carol</td>
</tr>
</tbody>
</table>
Advantage 2: μDBs allow encryption with user-specific keys
Advantage 2: $\mu$DBs allow encryption with user-specific keys

Delete Alice’s data $\rightarrow$ Delete Alice’s key $\rightarrow$ Backups of Alice’s data are “deleted”
Advantage 3: μDBs ensure applications maintain compliance throughout execution

• Some **regular application operations** may violate compliance
  • Cause data to be orphaned (*i.e.*, has no owners)

• K9db detects and rejects violating operations with the help of μDBs
  • *E.g.*, orphaned data cannot be stored in any user μDB
Advantage 3: μDBs ensure applications maintain compliance throughout execution

• Some **regular application operations** may violate compliance
  • Cause data to be orphaned (*i.e.*, has no owners)

• K9db detects and rejects violating operations with the help of μDBs
  • *E.g.*, Orphaned data cannot be stored in any user μDB

• Common pattern: temporarily create orphaned data then delete it
  • K9db safely supports such sequences using **compliance transactions (CTX)**
  • K9db commits CTX only if compliance is restored
Challenges

- Capture application-specific compliance policy
  - Schema annotations
  - Data Ownership Graph

- Correctly handle access/deletion requests and enforce compliance invariants
  - Ownership-aware storage
  - Compliance transactions

- Maintain good performance
K9db Implementation

- K9db realizes μDBs over a single datastore (RocksDB)
  - K9db secondary indexes identify all the μDBs where a row is stored
  - RocksDB iterators, snapshots, transactions ...
K9db Implementation

• K9db realizes μDBs over a single datastore (RocksDB)
  • K9db secondary indexes identify all the μDBs where a row is stored
  • RocksDB iterators, snapshots, transactions ...

• K9db serves complex queries from an integrated in-memory cache
  • based on materialized views maintained via dataflow processing
  • K9db updates cache in response to deletion requests
    → cache is always compliant
Evaluation Questions
Evaluation Questions

1. What is the impact of using K9db on E2E web application performance?
Evaluation Questions

1. What is the impact of using K9db on E2E web application performance?

2. What effect do K9db’s design features and optimizations have on performance?
Evaluation Questions

1. What is the impact of using K9db on E2E web application performance?

2. What effect do K9db’s design features and optimizations have on performance?

3. How much application developer effort is required to use K9db?
Evaluation Questions

1. What is the impact of using K9db on E2E web application performance?
   • Two real world applications: Lobsters, ownCloud

2. What effect do K9db’s design features and optimizations have on performance?

3. How much application developer effort is required to use K9db?
What is the impact of using K9db on E2E web application performance?

The diagram shows the median latencies for various actions in a web application. The activities are categorized into read-heavy and write-heavy operations. The median latencies for both categories are compared to illustrate the performance impact of using K9db.
What is the impact of using K9db on E2E web application performance?

![Median latencies chart]

- **Median latencies**
  - Read story
  - Frontpage
  - User profile
  - Comments
  - Recent stories
  - Vote on comment
  - Vote on story
  - Post comment
  - Post story

- **Better**
  - Read-heavy
  - Write-heavy
What is the impact of using K9db on E2E web application performance?

- **Median latencies**
  - **Read story**
  - **Frontpage**
  - **User profile**
  - **Comments**
  - **Recent stories**
  - **Vote on comment**
  - **Vote on story**
  - **Post comment**
  - **Post story**

**Better performance**

**better**

**Median Latency [ms]**

- **MariaDB**
- **K9db**

**read-heavy**

**write-heavy**
What is the impact of using K9db on E2E web application performance?

K9db achieves compliance with similar performance to baseline.
What is the impact of using K9db on E2E web application performance?

**Median latencies**

- **K9db caching**

- **Latency (ms)**
  - **0 to 10**
  - **10 to 20**
  - **20 to 30**

- **Better**

- **Read story**
- **Frontpage**
- **User profile**
- **Comments**
- **Recent stories**
- **Vote on comment**
- **Vote on story**
- **Post comment**
- **Post story**

**Categories**

- **Read-heavy**
- **Write-heavy**
What is the impact of using K9db on E2E web application performance?

**Median latencies**

- **Read story**
- **Frontpage**
- **User profile**
- **Comments**
- **Recent stories**
- **Vote on comment**
- **Vote on story**
- **Post comment**
- **Post story**

**Competitive Writes**

- MariaDB
- K9db

Better performance in write-heavy scenarios.
What is the impact of using K9db on E2E web application performance?

![95th percentile tail latencies graph]

- Better tail latencies for read-heavy and write-heavy operations with K9db compared to MariaDB.
What is the impact of using K9db on E2E web application performance?

![95th percentile tail latencies graph]

**95th percentile tail latencies**

- **Better read-heavy**
- **More Write Overhead**

- **Overhead?**
- **MariaDB**
- **K9db**
What is the impact of using K9db on E2E web application performance?

![Diagram showing 95th percentile tail latencies for different operations in read-heavy and write-heavy scenarios.](image)
What is the impact of using K9db on E2E web application performance?

95th percentile tail latencies

- Decrypting 1000s of comments

95th Percentile Latency [ms]

Read story - Frontpage - User profile - Comments - Recent stories - Vote on comment - Vote on story - Post comment - Post story

better

read-heavy write-heavy
Evaluation – more in the paper

• Comparable performance to on-demand caching (Memcached), reasonable memory overhead

• Storage layout optimizations critical for good performance

• K9db’s schema annotations can express policies for 10 real web applications
Conclusion: K9db is a database that helps developers get compliance with GDPR right!

• **Key abstraction: data ownership graph (DOG)** captures compliance policies of real applications

• Ownership aware storage layer with per-user μDBs

• K9db achieves performance comparable to SQL databases

https://github.com/brownsys/K9db

babman@brown.edu