SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Analysis and Enforcement of GDPR Rules on Key-Value Stores

Ertugrul Aypek

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Analysis and Enforcement of GDPR Rules on Key-Value Stores

Analyse und Anwendung der DSGVO-Regeln in Schlüsselwert-Datenspeicher

Author:	Ertugrul Aypek
Supervisor:	Professor Pramod Bhatotia, Dr.-Ing.
Advisor:	Dimitrios Stavrakakis, M.Eng.
Submission Date:	16.08.2023

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 16.08.2023 Ertugrul Aypek

Acknowledgments

I would like to thank Dimitrios Stavrakakis, M.Eng. for advising me during my thesis and helping me with all the challenges. Also, I would like to thank Professor Pramod Bhatotia, Dr.-Ing. for supervising the thesis.

I dedicate this work to my beloved wife who has always supported and motivated me, and my parents who have always been there for me.

Abstract

General Data Protection Regulation (GDPR) is a comprehensive data protection law that was implemented by the European Union (EU) on May 25, 2018 [12]. Its primary goal is to give EU citizens more control over their personal data and to simplify the regulatory environment for international business by unifying the regulation within the EU. Non-compliance with the regulations can result in significant fines, which can be up to 4% of a company's annual global turnover or €20 million (whichever is higher) [6]. Therefore, tech companies that handle personal data of EU citizens need to take GDPR compliance seriously to avoid penalties and maintain the trust of their users.

GDPR has forced organizations to rethink and redesign how data is stored and processed. This results in significant performance overheads and changes in each system or data store. In this project, a proxy system is proposed to make key-value stores GDPR-compliant without any change in the database internals. As a showcase, Rocksdb and Redis clients are implemented and evaluated but any other key-value store can be supported. The system extends the user data with metadata to address the GDPR requirements and provides access with a policy language. This thesis contains the architecture, design decisions, implementation, and evaluation of the proposed system. The source code can be found on `https://github.com/dimstav23/GDPRuler`.

Contents

Acknowledgments **iii**

Abstract **iv**

1 Introduction **1**

2 Background **3**
2.1 GDPR definitions . . . 3
2.1.1 Personal data . . . 3
2.1.2 Customer . . . 4
2.1.3 Controller . . . 4
2.1.4 Processor . . . 5
2.1.5 Regulator . . . 5
2.2 GDPR articles . . . 5
2.3 Key-value stores . . . 8

3 Overview **10**
3.1 System goals . . . 10
3.1.1 GDPR compliance . . . 10
3.1.2 Data encryption and pseudonymization . . . 10
3.1.3 Granular access controls . . . 11
3.1.4 Auditing and accountability . . . 11
3.1.5 Efficiency and scalability . . . 11
3.1.6 Integrability . . . 11
3.2 System overview . . . 11
3.3 System flow . . . 13
3.4 Design challenges . . . 14

4 Design **16**
4.1 GDPR metadata . . . 17
4.2 Policy language - defining the query interface . . . 19
4.3 Policy compiler - parsing and validating queries . . . 21

4.4 Query rewriter . . . 21
4.5 GDPR policy validator . . . 22
4.6 Cipher engine . . . 23
4.7 Logging engine . . . 24
4.8 GDPR proxy controller . . . 25

5 Implementation **26**
5.1 GDPR Proxy Controller implementation . . . 26
5.2 Key-value data store client implementation . . . 26
5.3 Logging implementation . . . 27
5.4 Encryption implementation . . . 29

6 Evaluation **31**
6.1 Test setup . . . 31
6.2 GDPR Metadata Overhead . . . 34
6.3 Logging Overhead . . . 34
6.4 Value Encryption Overhead . . . 35
6.5 Log Encryption Overhead . . . 36
6.6 Full GDPR Compliance Overhead . . . 37
6.7 Concurrency and Scaling . . . 37
6.8 Disk usage . . . 38

7 Related Work **40**
7.1 GDPR Benchmarking tools . . . 40
7.2 GDPR Compliance Tools/Methodologies . . . 40

8 Summary and Conclusion **43**

9 Future Work **44**
9.1 Query metadata implementation . . . 44
9.2 Logging performance improvement . . . 44
9.3 Access to all data of a user . . . 45
9.4 System interface with common protocols . . . 45

Abbreviations **46**

List of Figures **48**

List of Tables **49**

Bibliography **50**

1 Introduction

In the era of data-driven technologies, the protection of individuals' privacy and data rights has emerged as a critical global concern. The enactment of the General Data Protection Regulation (GDPR) by the European Union (EU) in 2018 marked a significant milestone in the realm of data privacy laws, setting the standard for safeguarding personal data in the digital age. However, for businesses operating systems while ensuring GDPR compliance presents a unique set of challenges.

GDPR has become one of the most known laws in the EU. After one year of its introduction, 67% of the EU citizens were aware of it [1]. Since the entry into the application of the GDPR, more than €2.5 billion in fines have been imposed by the authorities for GDPR breaches [16]. Some of the companies decided to stop serving EU customers. According to [13], one-third of the top U.S. newspapers chose to block European users instead of complying with the regulations.

In order to comply with the new regulations, software applications, and storage systems had to evolve. So many businesses have been established to benchmark the GDPR compliance of a system or to offer a GDPR-compliant storage system. Examples of such systems are [14], [11], and [2]. The proposed systems work only on Structured Query Language (SQL) data stores. Moreover, they are mostly focusing on costly graph calculations which highly degrades the performance.

On the other hand, key-value data stores have gained widespread adoption due to their simplicity, scalability, and good performance characteristics. These Not Only SQL (NoSQL) databases serve as the backbone for modern applications, ranging from social media networks to real-time data analytics platforms. Nonetheless, their decentralized nature and inherent design principles often clash with the hard requirements of GDPR, requiring a thoughtful and innovative approach to reconcile the two.

The primary objective of this thesis is to introduce a pioneering solution: a GDPR-compliant proxy system tailored specifically for key-value data stores. This novel approach seeks to address the complex interplay between maintaining high-performance

data processing capabilities and ensuring compliance with GDPR principles without affecting the database internals. By effectively bridging the gap between data processing efficiency and privacy protection, the proposed proxy system promises to redefine the landscape of GDPR compliance within the context of key-value data stores.

The proposed GDPR-compliant proxy system has been designed to seamlessly integrate with any key-value data store, providing a flexible and adaptable solution across various database technologies. While the system's architecture and implementation are robust enough to accommodate any KV store, this thesis will showcase its functionality through successful implementations with Redis and RocksDB. By leveraging its intermediary role between applications and data stores, the proxy system enforces the necessary data processing controls, and encryption techniques to ensure GDPR compliance. The system captures and stores essential metadata required for GDPR adherence, such as sharing consent records, purpose limitation, accountability, and data retention policies. This way, the system empowers organizations to confidently navigate complex regulations while safeguarding the privacy and rights of individuals' personal data. A policy language is constructed for organizations or authorities to be able to interact with the system.

To assess the system's performance comprehensively, it is tested with different combinations of workloads, database instances, number of concurrent clients, and encryption settings, among other factors. The performance overheads arising from GDPR compliance are reported through a comparative analysis against the native database performances.

2 Background

This thesis proposes a system that brings GDPR compliance to the key-value data stores. Therefore, it is essential to introduce the necessary definitions and explain the requirements of GDPR and the specifics of key-value stores.

2.1 GDPR definitions

There are a few entities and definitions to explain the GDPR interactions. Figure 2.1 represents the relations between different entities and the following subsections go over them one by one.

2.1.1 Personal data

Personal data is defined as any information that relates to an identified or identifiable person. Examples of personal data include names, identification numbers, location data, online identifiers (such as IP addresses and cookies), physical, physiological, genetic, mental, economic, cultural, or social identity information, and any other data that, when combined or processed, can lead to the identification of an individual. Any personal data falls under the protection of GDPR.

GDPR protects personal data regardless of the medium in which it is stored or processed, including digital formats, physical records, or other means. It applies to personal data held by data controllers and data processors within the European Union (EU) and to organizations outside the EU that offer goods or services to, or monitor the behavior of, individuals within the EU. The regulation aims to ensure that the processing of personal data is done lawfully, fairly, and transparently, with respect to individuals' rights to privacy and data protection.

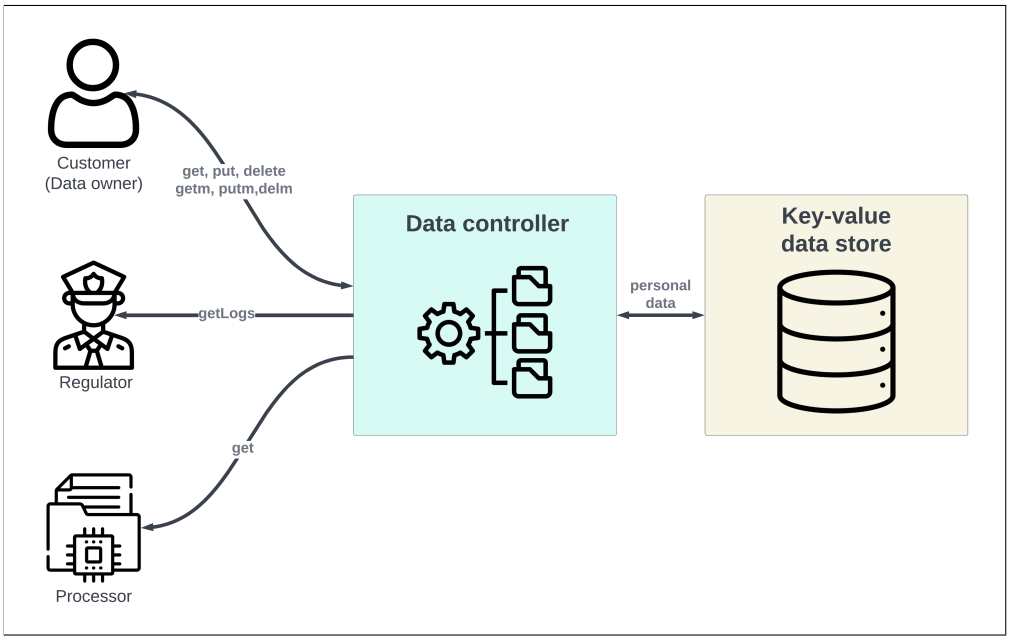

Figure 2.1: Relations between GDPR entities

2.1.2 Customer

Customers (data owners) are individuals whose personal data is being collected, processed, or stored by an organization. They have the right to exercise control over their data and can grant or withhold consent for its use by data controllers or processors. Data owners have the right to access their personal data, rectify inaccuracies, request deletion, and object to processing in certain situations. They play a central role in GDPR compliance as their rights and privacy are protected by the regulation.

2.1.3 Controller

Data controllers are entities or organizations that determine the purposes and means of processing personal data. They are responsible for ensuring that data processing activities comply with the GDPR and that individuals' rights are respected. Data controllers have specific obligations under the regulation, such as obtaining valid consent for data processing, providing transparent information about data processing activities, and implementing appropriate security measures to protect personal data. They have ultimate responsibility for data protection and are accountable for their data processing practices.

2.1.4 Processor

Data processors are entities that process personal data on behalf of data controllers. They act under the instructions of data controllers and are required to follow the controller's documented instructions when handling personal data. Data processors must ensure the security and confidentiality of the data they process and assist data controllers in meeting their GDPR obligations. They have specific responsibilities under the regulation and must only engage sub-processors with the controller's consent.

2.1.5 Regulator

Regulators, also known as supervisory authorities, are independent public authorities appointed by EU member states to enforce and oversee the application of the GDPR. Each member state has its own supervisory authority responsible for monitoring compliance with the regulation within its territory. Regulators have the power to investigate data breaches, issue fines, and provide guidance on data protection matters. They play a crucial role in enforcing the GDPR and protecting individuals' rights, ensuring that organizations adhere to the principles and requirements of the regulation and take appropriate actions against non-compliant entities.

2.2 GDPR articles

Recently, data privacy laws have been getting more popular and important as a result of various data breaches. One of the most widely recognized laws is European Union's GDPR. Although it is introduced by the EU and aims to protect the EU citizens, GDPR potentially affects the whole world. Any company or business collecting EU citizens' personal data is subject to the regulations and resulting penalties. Therefore, understanding what it requires from the data collector entities is critical.

On the 27th of April, 2016, the definition and requirements of GDPR are published by the European Union in a journal containing 11 chapters and 99 articles [12]. In the following subsections, articles containing the important principles that are relevant to the intersection of GDPR and key-value data stores are listed.

Article 5: Principles Relating to Processing of Personal Data: This article specifies the core principles for processing personal data. It emphasizes that data must be processed lawfully, fairly, and transparently.

Data must be collected for specified, explicit, and legitimate purposes, and should not be further processed in a manner incompatible with those purposes. If an organization intends to use the data for other reasons beyond the original purpose, it must seek additional consent from the data subject. This restriction will be referred to as **purpose limitation**.

Additionally, personal data should not be kept in a form that allows the identification of data subjects for longer than is necessary to achieve the purposes for which the data is processed. Organizations must establish appropriate data retention policies and delete the data once its retention period has expired. This will be referred to as **storage limitation**.

Article 15: Right of access by the data subject: This article grants individuals access to their personal data. It entitles data subjects to request confirmation from data controllers about the processing of their personal data and to access and obtain a copy of that data. In addition to the actual data, the data owners are eligible to access the source of their personal data, the purpose of processing, and the length of time the data will be stored, among other items. The data controller must provide this information free of charge for the first copy. The right of access empowers individuals to be aware of and verify the lawfulness of data processing activities concerning their personal data. This increases the trust between data subjects and organizations.

Article 17: Right to be forgotten: This article grants individuals the right to request the erasure of their personal data, also known as the right to be forgotten. Data subjects can exercise this right whenever they want. Some example scenarios are the data being no longer necessary to be processed or data processing being unlawful. Therefore, the data controllers must comply with erasure requests and ensure the complete removal of the data from their systems. However, the right to erasure is not absolute, and exemptions apply in certain cases, such as when data processing is required for legal compliance or exercising freedom of expression. Thus, article 17 gives individuals great control over their personal data and enforces their privacy rights in the digital world.

Article 21: Right to object: This article provides individuals with the right to object to the processing of their personal data in specific situations. Upon receiving such objections, the data controller must stop the processing unless they have legitimate reasons that override the individual's rights. Individuals have the freedom to exercise this right at any time, and data controllers are required to inform them about this right during data collection.

Article 22: Automated individual decision-making: This article grants the data subject the right to be out of solely automated processing. Data controllers must implement safeguards to keep the data records out of automated processing and seek human intervention.

Article 25: Data protection by design and by default: This article requires data controllers to integrate data protection principles into their systems from the beginning. This means considering privacy and security aspects during system development, minimizing data collection, and applying measures like encryption or anonymization. By following these requirements, organizations can prioritize data privacy, reduce risks, and demonstrate their commitment to protecting individuals' personal data in line with GDPR standards.

Article 28: Processor: This article outlines the requirements for data processors and the contract between processors and controllers. It allows the processors to access only those personal data for which they have access and valid purpose. By complying with the requirements of Article 28, data controllers and processors can establish a strong legal basis for their data processing relationship. This results in accountability and ensures that the user's privacy and rights are honored.

Article 30: Records of processing activities: This article addresses the need for data controllers and processors to keep records of their data processing activities. This article requires organizations to document various information related to their data processing operations such as the purposes of the processing, recipients of the data, data transfers to third parties, and retention periods. These records play a crucial role in promoting transparency and accountability, allowing data protection authorities to assess compliance with GDPR requirements. By maintaining well-organized and up-to-date records as described in Article 30, organizations can demonstrate their commitment to data protection and respond to inquiries from regulatory authorities and data subjects regarding their data processing activities.

Article 32: Security of processing: This article establishes the need for data controllers and processors to implement necessary technical measures to ensure the security of personal data. These measures should be designed to protect the data against unauthorized access, accidental or unlawful destruction, alteration, disclosure, or any other form of unlawful processing. By complying with Article 32, organizations can safeguard personal data and minimize the risk of data breaches, contributing to maintaining the confidentiality, integrity, and availability of the data they process.

Article 33: Notification of a personal data breach to the supervisory authority: This article outlines the requirement for data controllers to notify the supervisory authority in the event of a personal data breach. When a breach occurs and is likely to result in a risk to individuals' rights and freedoms, the data controller must report the incident to the supervisory authority without delay. The notification should include details about the breach such as affected data subjects and personal data, the estimated consequences of the breach, and the measures taken or proposed to address the incident and mitigate its impact. Compliance with Article 33 is essential for ensuring transparency, and accountability. Moreover, it enables timely responses to data breaches, enabling supervisory authorities to assess the risks and take appropriate actions to protect individuals' rights and privacy.

2.3 Key-value stores

In the world of modern data management, Key-Value Stores have emerged as a fundamental variety of NoSQL databases [10]. These data stores offer a simple yet powerful approach to data storage, retrieval, and management, making them increasingly popular for various applications. At their core, Key-Value Stores are designed to efficiently handle vast amounts of data by organizing information in a straightforward manner: each data entry, or "value" is associated with a unique identifier called a "key". This key-value pairing allows for rapid data access without the constraints of complex data models, opposed to the typical traditional relational databases.

The fundamental functionalities of Key-Value Stores revolve around three primary operations: Get, Put, and Delete.

- **Get**: The "Get" operation is the foundation of Key-Value Stores. By providing the unique key associated with a particular value, applications can efficiently retrieve the corresponding data. This simplicity and speed of access make Key-Value Stores an ideal choice for scenarios where rapid data retrieval is critical, such as caching frequently accessed information or handling real-time data processing.
- **Put**: The "Put" operation allows data to be inserted or updated in the Key-Value Store. By providing a new key-value pair, or updating an existing one, applications can efficiently store and manage data. The seamless ability to add or modify data makes Key-Value Stores highly flexible and adaptable for dynamic data environments.
- **Delete**: The "Delete" operation enables the removal of data from the Key-Value

Store. By providing the key associated with a particular value, applications can efficiently delete the corresponding data entry. This functionality is valuable for managing data life cycles and ensuring that only relevant information is retained within the store.

The simplicity and efficiency of these basic functionalities are what set Key-Value Stores apart, making them well-suited for handling a wide range of use cases and thus popular. Some popular key-value data store systems in the industry are Redis, Memcached, DynamoDB, and RocksDB. They are often used to improve the performance of applications by reducing the need to access slower data sources, such as databases or external APIs, for frequently requested data. On top of these, they are used in web server backends for user session storage, user preference storage, and configuration management. Thanks to their speed and scale capacity, key-value stores are also widely used in online gaming, the Internet of Things (IoT), and real-time analytics sectors.

3 Overview

3.1 System goals

The primary objective of this project is to facilitate the seamless integration of key-value data stores into existing data management architectures while ensuring strict adherence to the principles and requirements defined by the General Data Protection Regulation (GDPR). By acting as an intermediary layer between applications and the underlying data stores, the aim is to enhance data privacy, security, and accountability while enabling efficient and transparent data processing.

3.1.1 GDPR compliance

The core goal of the proxy system is to ensure full compliance with the GDPR, safeguarding the rights and privacy of data subjects. The mechanisms to enforce data protection principles, such as purpose limitation, are implemented. The system also facilitates data subject rights, including the right to access, rectification, erasure, and objection. By adhering to GDPR guidelines, the proxy system fosters trust between data controllers, data processors, and the individuals whose personal data is being processed.

3.1.2 Data encryption and pseudonymization

To bolster data security, the proxy system supports end-to-end encryption and pseudonymization techniques. By encrypting sensitive data at rest and in transit, we will mitigate the risk of unauthorized access and data breaches. Additionally, pseudonymization allows for anonymizing or replacing identifying information, further protecting data subjects' privacy while maintaining data utility.

3.1.3 Granular access controls

The proxy system implements granular access controls to restrict data access based on user roles and permissions. Only authorized personnel can have access to specific data, reducing the risk of data misuse and unauthorized processing.

3.1.4 Auditing and accountability

The system incorporates comprehensive logging and auditing functionalities. These features track data processing activities, enabling detailed audit trails of who accessed, modified, or deleted data. This level of accountability empowers data controllers to demonstrate compliance with the GDPR's accountability principle and aid in the event of data breach investigations.

3.1.5 Efficiency and scalability

While prioritizing data protection, the proxy system ensures optimal performance, efficiency, and scalability. By leveraging innovative data processing techniques, it delivers fast response times and handles large volumes of data efficiently, catering to diverse and dynamic data processing requirements.

3.1.6 Integrability

Another system goal is to provide a generic and flexible interface that enables seamless integration with various key-value data stores, including popular solutions like Redis and RocksDB. By abstracting the underlying data store specifics, the system empowers data controllers and processors to effortlessly switch between different key-value databases without major modifications to their applications. This versatility reduces development efforts, fosters code reusability, and promotes interoperability across diverse data storage environments. As a result, organizations can readily adapt to changing business requirements, capitalize on the strengths of different data stores, and implement efficient data management strategies while maintaining GDPR compliance at all times.

3.2 System overview

Figure 3.1 presents the high-level entities and components of the proposed system. The GDPR-compliant proxy system acts as a critical intermediary layer bridging three

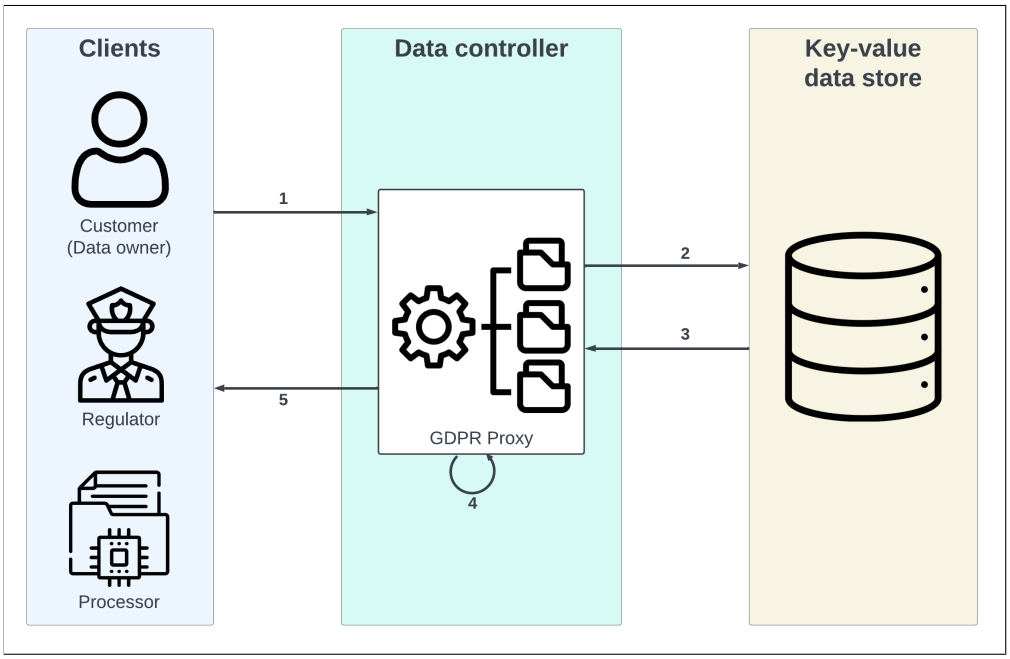

Figure 3.1: High-level architecture of the GDPR-compliant proxy system

fundamental components: clients, data controller, and key-value store. At the heart of this design, the data controller serves as the core processing hub where the proxy system is integrated, ensuring GDPR compliance while managing data interactions between clients and the key-value store. The clients segment encompasses different entities, including data owners, regulatory authorities, and data processors. These clients initiate data requests and interact with the data controller to access, modify, or delete personal data. All connections between clients and the data controller are securely established, ensuring the confidentiality and integrity of the data exchanged.

The data controller acts as the central processing unit within the system. Here, the GDPR-compliant proxy system is integrated to ensure that data processing activities align with GDPR principles and requirements. It enforces GDPR compliance by implementing measures such as encryption, authorization, and auditing functionalities. In order to keep track of purposes, objections, data access permissions, data storage expirations, and data origins; it enriches the personal data with additional GDPR metadata. It provides query Application Programming Interface (API) through a secure channel for client interactions. On top of the regular key-value data store operations of get(), put(), and delete(); the API provides getm(), putm(), delm(), and getLogs()

queries. Before accepting any queries, the GDPR Proxy expects a client policy that helps to identify the client and set his/her default query preferences. Note that the getm(), putm(), and delm() queries are not currently completely implemented but their finalization is part of the design and future work.

The key-value store is the underlying data storage component that holds the actual data entries in the form of key-value pairs. The data controller interacts with the key-value store to access, update, or remove data on behalf of clients. Any value written to the storage unit is encrypted and it contains additional metadata on top of the personal data. This storage layer can be various key-value databases like Redis and RocksDB. The generic interface of the GDPR-compliant allows seamless integration with any key-value store, enabling effortless switching between data storage environments without compromising GDPR compliance.

3.3 System flow

The end-to-end high-level system flow contains several steps which are corresponding to the numbers specified in Figure 3.1.

1. Client request: The system flow begins when various entities, including data owners, regulatory authorities, and data processors, initiate data interactions with the Data Controller. These entities establish secure connections with the data controller to protect the confidentiality and integrity of data transmissions. Upon the establishment of the connection, they first send their query policies which contain their session key, and default preferences. Data owners may request access to their personal data, seek updates, or exercise their rights to erasure or rectification. They can perform get(), put(), delete(), getm(), putm(), delm() queries. Regulatory authorities might perform getm() and getLogs() queries to get audits for their investigations. Data Processors engage in data processing activities on behalf of Data controllers and they can use get() query.
2. Key-value store request: Upon receiving a connection request, the data controller expects the client default policy and saves it until the session closes. A session can contain multiple queries. For each query, the data controller sends a request to the key-value store. An important note on put() query is that the data controller adds metadata to the value and encrypts it before sending the query. The query is sent on a secure channel.
3. Key-value store response: Key-value store executes the query and returns the

corresponding response over the secure channel. For put() and delete() queries, it returns a success/failure response and for get() query, it returns the corresponding value if it exists.

4. Access and compliance checks: Once the controller receives the value, it first deciphers it and checks if the client is allowed to access/modify it. A series of GDPR compliance checks follow the authorization such as purpose, objection, and expiration checks. Based on this filtering step, the query response is created. Note that all the delete(), delm(), put(), and putm() queries need an additional getm() query to be able to run the filtering and see if the original query is allowed. In addition to creating the response message, the data controller logs the operation and its result as well for auditing purposes.
5. Server response: The data controller returns the query response to the client.

3.4 Design challenges

Building a robust and GDPR-compliant proxy system comes with several design challenges that need thoughtful consideration and innovative solutions. Overcoming these challenges is crucial to ensuring seamless data management while upholding data subject rights and privacy.

Metadata explosion: Dealing with GDPR metadata within each key-value pair can lead to metadata explosion and significant data storage requirements. As the system accumulates more data and users, the metadata overhead may become more important. Balancing the need for metadata with efficient data storage and retrieval mechanisms becomes a critical challenge to maintain system performance and scalability.

Scalability under huge amount of data: Handling huge amounts of data while maintaining system responsiveness and low latencies is a big challenge. As data volumes grow, the system must exhibit horizontal scalability, allowing seamless expansion across distributed resources without compromising performance.

Correctness under concurrency: Concurrency introduces complexities when multiple users and processors interact with the system simultaneously. Ensuring data integrity, consistency, and correctness during concurrent operations, such as data reads and writes, is mandatory.

Audit log management: Maintaining comprehensive audit logs for all data process-

ing activities is crucial for GDPR compliance and accountability. The challenge is to manage and store these logs without heavily affecting the system's performance. Possible solutions can include implementing efficient log compression, archival strategies, etc.

4 Design

Figure 4.1 shows the detailed design of the system. The enumerations in the figure represent the flow of a read request through the components. Below is the explanation of each step.

1. A request is initiated by the clients. GDPR Proxy Controller component is the only component that clients can directly interact with. It creates a new session thread for each connection that handles the queries. The first input after the connection establishment is the default policy. Then, the queries can be submitted.
2. Policy Compiler component parses and verifies the syntax of the query.
3. Using default policy and query policy, the Query Rewriter component finalizes the policy. Default policies are overridden by the query policies, ending up with the final query.
4. Final query is forwarded to the Key-value Datastore Client.
5. Key-value Datastore Client sends a request to the key-value datastore instance and gets the result.
6. If encryption is enabled, retrieved data in step 5 is decrypted.
7. Value is forwarded to the GPDR Policy Validator component. This component employs all the GDPR compliance checks by comparing the existing metadata against the query. It is the control mechanism to allow or deny a query.
8. Before directly forwarding the response back to the client, the query and its result are forwarded to the Logging Engine component.
9. If encryption is enabled, the log record is encrypted by the Cipher Engine.
10. The log record is written to the filesystem.
11. After logging the query, the query result is forwarded to the GDPR Proxy Controller.
12. GDPR Proxy Controller responds to the query result back to the client.

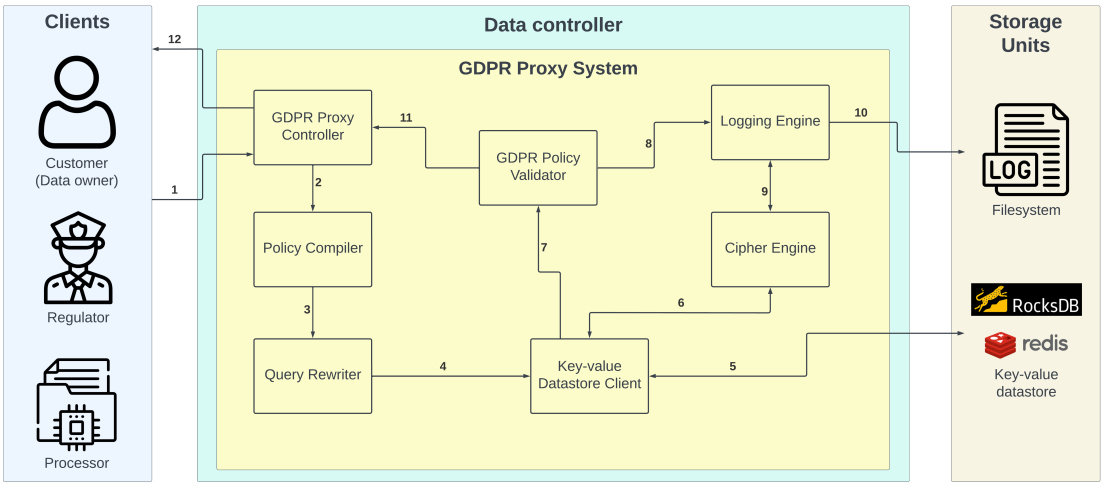

Figure 4.1: GDPR Proxy's detailed design diagram

The difference for a write query would be that after the 10th step, the new value will be written/deleted to/from the datastore. So, write queries first employ a read query to validate the policy and then execute the original write query if it is valid. The following sections will include the detailed design of each component within the system as well as the data structures used.

4.1 GDPR metadata

To ensure full compliance with the General Data Protection Regulation (GDPR) and enhance data governance, the GDPR-compliant proxy system incorporates a comprehensive set of metadata fields within each key-value pair. These metadata fields provide essential information about the data, its usage, and associated user rights, enabling transparent and accountable data processing. Below is the list of metadata fields in use. Different metadata fields are separated by "|" character.

1. userKey: This field represents the unique key of the data subject or user who owns the specific key-value pair. This metadata ensures that data processing activities are linked to the rightful owner, allowing the data controller to facilitate data subject rights, including access, rectification, or deletion requests.

2. encryption: This field defines whether the value should be encrypted or not.

When set to true, it signifies that the value is encrypted to safeguard sensitive data from unauthorized access, ensuring confidentiality and data security throughout its lifecycle. The default value for this metadata is true.

3. purpose: The "purpose" field contains a set of allowed purposes for which the data can be processed. A data processor can only access the key-value pair if the data processor's purposes are a subset of the value's purposes. In order to scale better and save time and space, the comprehensive set of all possible purposes is represented as a bitmap. By setting the specific bits, the metadata establishes purpose limitation, ensuring that data is not used beyond the specified, legitimate objectives. The default value for this metadata is empty.
4. objection: This field denotes the set of objections raised by the data subject against specific data processing purposes. If the data subject objects to certain purposes, the key-value pair cannot be used for those activities, thus respecting the data subject's rights and choices. Similar to purposes, the comprehensive set of all possible objections is represented as a bitmap.
5. origin: This field denotes the source of the data object. It is used by the data processors to assess the data's credibility.
6. expiration: The "expiration" field indicates the expiration time of the key-value pair. When the expiration time is reached, the data can no longer be used, ensuring compliance with the data retention and storage limitation principles of the GDPR. The default value for this metadata is empty.
7. share: This field maintains a list of users with whom the key-value pair is shared. This metadata helps track data sharing activities, enhancing transparency and accountability, and ensuring that data is only shared with authorized recipients. The default value for this metadata is empty.
8. monitor: The "monitor" field acts as a flag that enables or disables logging for operations on the specific key-value pair. When logging is enabled, the system generates a detailed audit log, recording all interactions and modifications, and facilitating compliance verification and data breach investigations.

The metadata is stored in the value section of the key-value pair by prefixing the original value. Below is an example value with metadata enrichment.

user486 | 1 | 15 | 16 | origin22 | 1690684360 | user2,user5,user10 | 1 | user-value

In this example, the first metadata field, *user486*, is the user key of the owner of this key-value pair. The second field, *1*, represents that encryption is enabled for this value. This means that updating the value will be kept encrypted as well unless the metadata is updated and set to *0*. The third field, *15*, represents the allowed purposes in decimal form. The binary equivalent of *15* is *1111*. Thus, this key-value pair can be used for purpose0, purpose1, purpose2, and purpose3. The purpose numbers can be mapped to real purposes by the data controller. The fourth field, *16*, is the decimal version of the objections. The binary equivalent of *16* is *10000*. This means that the owner has objected to the 4th purpose and the key-value pair can not be used for this purpose. The fifth metadata field, *origin22*, can be interpreted as the origin of the data. The sixth metadata field, *1690684360*, is the epoch timestamp of the storage/processing expiration. The seventh metadata field, *user2*,*user5*,*user10*, is the list of user keys that this data object is shared with. Each user key is separated by "," character. The eighth field, *1*, shows that the monitoring is activated for this pair and each operation on it will be logged. The last field, *user-value* is basically the original value stored by the owner.

4.2 Policy language - defining the query interface

The GDPR-compliant proxy system allows entities to interact with it by providing a query interface. The query can have several predicates and specifications. A complete query with predicates describes a policy. The policy language in the GDPR-compliant proxy system provides a powerful query interface that enables users, data processors, and regulatory authorities to interact with the system. Each policy consists of specific predicates that allow clients to express their data access and processing preferences in a structured manner.

Below is a list of policy language predicates. Each predicate is separated by "&" character. Note that the predicates ending with "Is" suffix are used for retrieval purposes such that retrieval fails if the metadata conditions are not met.

- **Arithmetic operations**: *eq*, *le*, *lt*, *ge*, and *gt* are the set of arithmetic operations. Each of them takes two parameters. Their respective meanings are equal to, less than or equal to, less than, greater than or equal to, greater than.
- **userKey**: The "userKey" predicate refers to the user's session key, ensuring that access to data is authorized and associated with a specific user's session. This predicate plays a crucial role in user authentication and authorization, guaranteeing that only authorized users can initiate data queries.

- **owner and ownerIs**: This predicate identifies the object owner, linking data processing activities to the rightful data subject. By specifying the object owner, the policy ensures that data processing is aligned with the data subject's rights and consent.
- **expiration and expirationIs**: This predicate contains the object's expiration timestamp, indicating the validity period of the data. This predicate helps enforce data retention policies, ensuring that data is not processed beyond the specified duration.
- **purpose and purposeIs**: This predicate lists the purposes of use for an object, explicitly defining the legitimate reasons for data processing. This predicate implements the purpose limitation principle, preventing data from being used for unauthorized purposes.
- **origin and originIs**: This predicate captures the origin of an object, providing information about its source. This predicate is valuable for data provenance and compliance verification, ensuring that data comes from trusted sources.
- **share and shareIs**: This predicate maintains a list of users with whom an object is shared. This predicate enhances transparency and accountability, allowing data subjects to track data sharing activities and providing an essential component for data sharing consent management.
- **objection and objectionIs**: This predicate comprises the list of objections raised by the data subject against specific data processing purposes. This predicate ensures that data processing does not occur when objections are present, respecting data subject rights and choices.
- **monitor**: The "monitor" predicate acts as a flag to define whether logging for operations on an object is necessary or not. Enabling monitoring allows for detailed audit trails, facilitating compliance verification and data breach investigations.
- **query(put | get | delete | putm | getm | delm | getLogs)**: The "query" predicate defines various query operations that users can perform on objects, including "put" for storing new data, "get" for retrieving data, "delete" for removing data, "putm" for metadata insertion, "getm" for metadata retrieval, "delm" for metadata deletion, and "getLogs" for accessing audit logs.

An example query with all policies can be seen below.

query(PUT("gdpr1","VAL"))&userKey("user1")&origin("src1")&monitor("false")
&objection("purpose3")&purpose("purpose0,purpose1,purpose2")

&share("user0")&expiration("0")

In this example, *user1* aims to put *(gdpr1,VAL)* key-value pair with origin *src1*, disabling audit logging for operations on this object, objection to *purpose3*, allowed purposes of *purpose0, purpose1,* and *purpose2*, sharing the object with *user0*, and without expiration.

4.3 Policy compiler - parsing and validating queries

The policy compiler plays a crucial role in the data processing workflow of the GDPR-compliant proxy system. When a query with a policy is received, the compiler parses and validates the syntax to ensure that it complies with the defined policy language syntax. Note that it does not employ the GDPR compliance checks.

The policy compiler employs parsing algorithms to extract the relevant predicates and their corresponding values from the query. It then validates the syntax and structure of the policy to ensure that it adheres to the predefined syntax of the policy language. During validation, the compiler checks for incorrect predicates and verifies the accuracy of data formats (e.g. timestamps).

4.4 Query rewriter

As explained in Policy language - defining the query interface, queries have predicates to define the policies. However, none of the predicates except the *query* predicate is required for a valid query. This is accomplished by default policies. A default policy is a JavaScript Object Notation (JSON) file containing the default values for any predicates. Upon the connection establishment with the GDPR proxy controller, clients are expected to provide a valid default policy. Below is a sample default policy:

```
{
    "userKey": "user0",
    "default_policy": {
        "purpose": [
            "purpose0",
            "purpose1",
            "purpose2"
```

```
        ],
        "objection": [
            "purpose3"
        ],
        "origin": [
            "src0"
        ],
        "expiration": [
            "0"
        ],
        "share": [
            "user1",
            "user2",
            "user3"
        ],
        "monitor": [
            "false"
        ]
    }
}
```

Listing 4.1: Sample default policy

The query rewriter component is responsible for merging the default policy and query policy. The precedence is with the query policy. In other words, query policy overrides the default policy predicate values. This component is only executed for *put* queries upon writing a non-existent key-value pair. If the pair already exists, then this component does not interfere with any settings.

4.5 GDPR policy validator

GDPR policy validator is one of the most critical components in the system. After the policy compiler is finished processing the query, the GDPR policy validator component ensures GDPR compliance. The component is used for all query types except the put query when the key does not exist. The main logic of this component is to compare the existing metadata with the query policies to deny or allow it. This is done as a chain of filters and if the filtering fails at any step, the query is blocked.

The component first validates the *session key* by checking if the object's owner matches

the given key or not. Additionally, it checks if the given key is among the list of shared users. If none of these two conditions succeed, then the query is failed.

Next, purposes are validated by the GDPR policy validator. The expectation is that all the query purposes must be a subset of the existing object's purposes.

Objections are the next filtering topic. The component expects that none of the query purposes are objected to by the data owner.

Then, the query origin is checked against the data object origin. If the origin is not the one expected by the query, it fails.

Lastly, the expiration is checked. If the object's expiration timestamp is set to 0, it means that no expiration is set. In such a case, this filtering step succeeds regardless of the current time. Otherwise, the query is regarded as valid only if the defined expiration time is not passed.

4.6 Cipher engine

The cipher engine is responsible for encryption and decryption which are one of the most essential processes of the GDPR proxy system. It ensures that the personal data is stored encrypted. Both the key-value object's value part and audit logs are encrypted. This enables to have the key-value data store and log storage unit hosted outside of the data controller cluster such as different cloud providers.

The cipher engine employs Advanced Encryption Standard - Galois/Counter Mode (AES-GCM) [8] 128-bit encryption algorithm. The GDPR proxy system uses different keys of size 128-bit for log encryption and value encryption for better security. Note that these keys are created and maintained by the data controller and they should be kept secure. The cipher engine is integrated into the key-value data store client interface. The data is decrypted after each read and it is encrypted before each write. This way, the other components of the overall system are relieved of concerns about the encryption and they only focus on processing the raw data.

Figure 4.2 shows the interactions of the Cipher Engine with the other components and the numbers in the figure show the steps for a write operation. In the first step, any other GPDR Proxy component calls either the key-value datastore client or the logging engine. In the second step, the Cipher Engine handles the encryption and

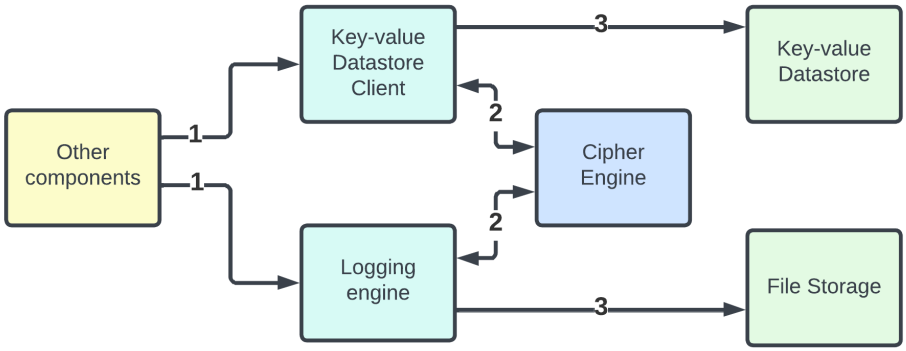

Figure 4.2: Cipher Engine's interaction with the other components

returns the response back. Finally, the encrypted value/log is written to the external systems/storage. For a read operation, the only change is the priorities of steps 2 and 3. Basically, first, the encrypted data is retrieved from the external storage systems, and then it is decrypted.

4.7 Logging engine

Logging or monitoring is another necessary component to achieve full GDPR compliance. When a data breach is detected or supervision is needed, regulatory authorities can use the *getLogs* query to access the logs. This way, they can see if the data controller is obeying the GDPR policies or not. As noted in the Policy language - defining the query interface section, monitoring can be turned on and off based on the data owner's choice. The log structure contains the operation timestamp, the user key who is the initiator of the operation, the operation type (get, put, etc.), the operation result (success/failure), and the new value if the operation type is *put* or *putm*.

Audit logs are basically stored in the filesystem and they are grouped by keys. So, each key has its own log file and this makes it easier to see the operation/access history on

a single key by different entities.

4.8 GDPR proxy controller

The GDPR proxy controller is the main component of the system. It is the orchestrator of almost all of the other components of the system. It enables data transmission between the clients and the system over a secure Transmission Control Protocol (TCP) channel. The connection to the key-value data stores is also managed by this component.

In terms of orchestration, it creates a session, reads the default policy, takes the queries, hands them over to the policy compiler, then to the query rewriter, GDPR policy validator, logging engine, and finally to the key-value client. The results of the queries are also written back to the session socket. The proxy controller can handle multiple sessions simultaneously.

5 Implementation

In this chapter, the most important and complex components of the GDPR proxy system are explained in further detail.

5.1 GDPR Proxy Controller implementation

The GDPR Proxy Controller is the main entry (frontend) to the system for the outside world to interact with it. The component is written in C++ programming language and contains around 250 lines of code. The users can communicate with the GDPR Proxy system using this component's exposed secure TCP channel. It listens to the clients' connection requests and handles them. The controller runs in a single thread but it spawns a new thread for each session to be able to handle multiple clients concurrently and scale.

The GDPR Proxy Controller is also the main orchestrator of the system. Upon startup, it first configures the log encryption key, value encryption key, datastore connections, etc. Then, it hands off the session request to a new thread where the client's default policy is parsed and stored, queries are parsed, validated, and served, and monitoring actions are taken.

5.2 Key-value data store client implementation

Figure 5.1 shows the class diagram of kv_clients. In this project, two of the popular key-value store instances, Redis [4] and RocksDB [7], are used to showcase the system's capabilities. So, the key-value client implementation includes an abstract kv_client class, along with concrete classes rocksdb_client and redis_client to support these database backends. The kv_factory class is responsible for creating instances of kv_client, promoting system extensibility. The component is written in C++ and contains around 300 lines of code.

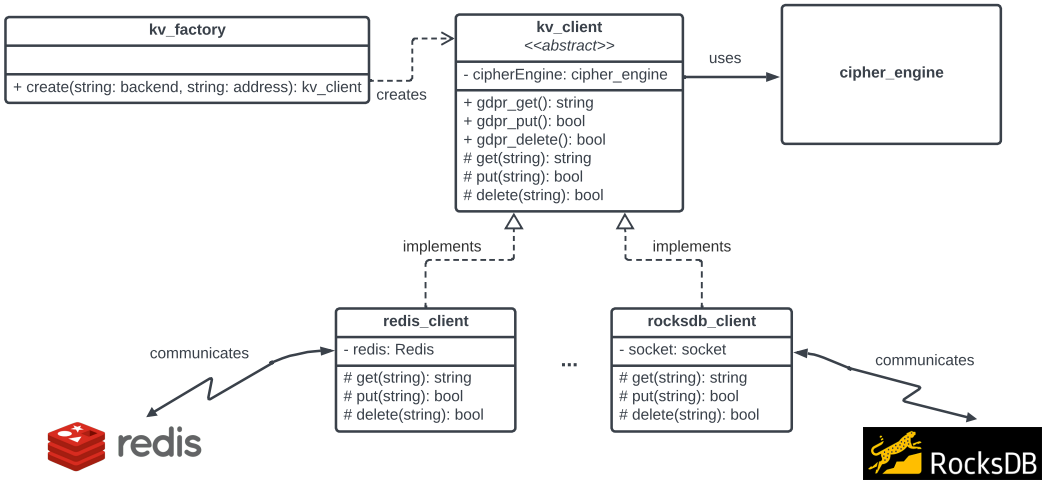

Figure 5.1: Key-value clients class diagram

Template method pattern is employed for better reusability. Public concrete methods of kv_client class (gdpr_get, gdpr_put, and gdpr_delete) already handle the encryption and decryption of the values and call the abstract get, put, and delete methods which are implemented by concrete client classes. This enables the rest of the components not to worry about the value being encrypted.

Redis and RocksDB instances are implemented to showcase the application. Note that Redis already comes with a client but RocksDB does not. Hence, the RocksDB instance holds a socket to communicate with the RocksDB server that we implemented. Although there are currently two instances, this design makes it very simple to extend the GDPR proxy system to be used with other key-value stores. The future maintainers just need to update the factory class and implement a new concrete client.

5.3 Logging implementation

The Logging Engine in the GDPR-compliant proxy system plays a crucial role in maintaining comprehensive audit trails for all data processing activities. The logging engine is written in C++ programming language and contains around 600 lines of code. To ensure efficient and secure logging, we have the following strategies.

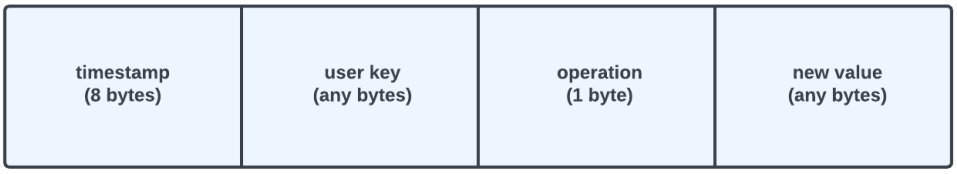

Figure 5.2: Unencrypted log record structure

Individual log files: Each key in the system is associated with a specific log file named "<key_name>.log." This approach allows for granular and organized logging. This allows the engine to be able to scale easily against increasing key-value pairs.

Persistent log file streams: To reduce the overhead of repeatedly opening and closing log file streams upon request, the Logging Engine opens the log files once for a request and keeps them opened afterward. To achieve fast retrieval and access, the engine maintains a hashmap that maps each key to its corresponding open log file stream.

Mutex-based concurrent access: Considering the possibility of multiple concurrent connections and data processing activities, the Logging Engine employs mutexes to safeguard access to each log file. To ensure that a log file is accessed by only one session or thread at a time, the engine maintains another hashmap that links each log file name to its corresponding mutex. In order for mutexes to make sense, the logging engine is implemented as a Singleton object. This concurrency control mechanism prevents data corruption and ensures consistent and synchronized logging.

Log Record Structure: Figure 5.2 shows a bare log record structure. An unencrypted log record contains 5 elements which are separated by "," character. The 8 bytes timestamp indicates the time of the query execution. The user key denotes the user initiating the query and it can be any number of bytes. Operation type {get, put, delete, getm, putm, delm, getLogs} and operation result {true, false} are compressed into a single byte. 3 bits of the byte are used for the operation type and 1 bit is used for the operation result. Lastly, there is an optional field of new value which is only populated upon write queries.

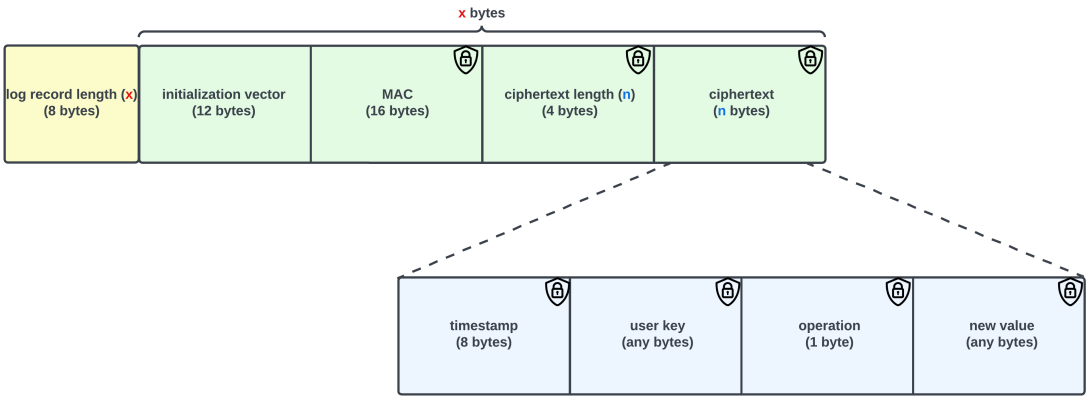

Figure 5.3: Encrypted log record structure

Figure 5.3 shows an encrypted log record structure. The whole record shown in Figure 5.2 is kept as encrypted. In order to successfully decrypt the encrypted log records, the size of each record needs to be known. This is handled by putting the record length of 8 bytes before the encrypted log record. The Logging Engine first reads the first 8 bytes to get the record length (x bytes). Then, it reads the next n bytes and decrypts them. These x bytes contain the whole structure of a ciphertext.

5.4 Encryption implementation

The encryption layer in our GDPR-compliant proxy system ensures data protection using the AES-GCM algorithm implemented by OpenSSL. This layer encrypts and decrypts both audit logs and values, employing two distinct symmetric keys for each operation to enhance data security and separation. The component is implemented with C++ programming language and contains around 400 lines of code.

AES-GCM algorithm: The encryption layer utilizes the AES-GCM 128 bit algorithm, a widely adopted and secure symmetric encryption technique. GCM provides both data confidentiality and authentication by cryptographic keys (encryption key and initialization vector) and Message Authentication Codes (MAC). By leveraging OpenSSL's implementation of AES-GCM, the encryption layer ensures data security.

Encryption keys: For better security, the encryption layer uses two different encryption

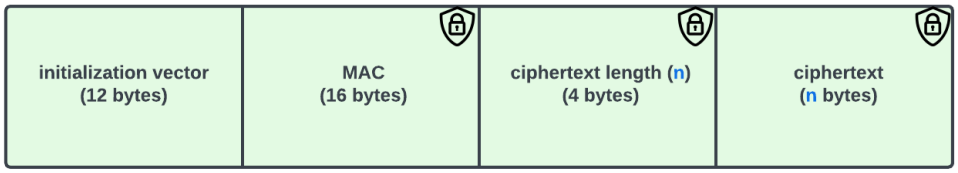

Figure 5.4: Encrypted text storage structure

keys. One key is dedicated to encrypting and decrypting audit logs, while the other key is used for values. These keys are configurable during the deployment of the GDPR proxy.

Storage structure: Figure 5.4 shows how an encrypted text is stored in external storage systems. The first 12 bytes represent the Initialization Vector (IV). The IV is a random value generated for each data encryption and it helps to avoid patterns and enhance encryption strength. The IV is kept unencrypted to enable proper decryption during data retrieval.

The next 16 bytes represent the Message Authentication Code (MAC) or authentication tag. The MAC serves as a security check to ensure data integrity during decryption. Verifying the MAC helps detect any unauthorized changes or corruption of encrypted data.

The following 4 bytes indicate the ciphertext size (n). These bytes specify the length of the encrypted data to be able to know the decryption scope.

The remaining n bytes represent the encrypted data. This segment contains the actual ciphertext resulting from the AES-GCM 128 bit encryption process.

6 Evaluation

To assess the performance and correctness of the GDPR-compliant proxy system, we conduct a comprehensive evaluation using the GDPRBench tool [15]. This is an open-source workload generator specifically designed for evaluating GDPR compliance in data management systems. GDPRBench provides a robust framework for generating realistic workloads that mimic real-world data processing scenarios. The tool allows us to define workloads with varying numbers of read and write queries, simulating the demands of multiple users, data processors, and regulatory authorities interacting with the system. We extend this tool to comply with our metadata structure so that the queries can be successfully parsed and analyzed.

The evaluation aims to measure the system's correctness, completion time, and storage space consumption under various workload scenarios.

1. **Correctness:** We define the correctness as successful completion of GDPRBench-generated workloads. The queries in these workloads are enriched with metadata and the controller needs to be GDPR-compliant in order to complete them without an error.
2. **Completion Time (s):** The completion time metric represents the time in seconds to complete all the GDPR queries in various workloads. The metric reveals the end-to-end performance of the system in terms of speed.
3. **Disk space consumption (MB):** The system stores the key-value records with additional metadata. This makes the storage space used by logs and datastore records larger. This metric aims to reveal the difference in Megabytes (MB).

6.1 Test setup

Compute environment: All of the experiments are conducted on a Non-Uniform Memory Access (NUMA) server with two nodes. Table 6.1 lists the features of a single

Computation environment aspect	Value
CPU model	AMD EPYC 7413 24-Core Processor
CPU Threads per core	2
CPU max MHz	3630 MHz
CPU min MHz	1500 MHz
CPU L1d, L1i, L2, L3 caches	1.5 MiB, 1.5 MiB, 24 MiB, 256 MiB
System Memory	128 GiB
Disk model	SAMSUNG MZQL21T9HCJR-00A07
Disk Storage	1920 GiB
Disk Type	NVMe
Operating System	NixOS 23.05 (Stoat)
Linux Kernel	6.3.0-rc2

Table 6.1: Features of a NUMA node used for the experiments

node. All of the GDPR Proxy Controller system, key-value datastore instances, and the clients' programs are executed on the same server.

Workloads: The workloads A, B, C, D, and F described in the GDPRBench [15] are used to test the various aspects of the system. GDPRBench workloads are based on Yahoo! Cloud Serving Benchmark (YCSB) [5] workloads. YCSB is a workload generator for benchmarking the key-value stores. It allows users to configure key-value sizes, total number of records, operations, and access distributions. The addition GDPRBench introduces on top of YCSB is the ability to specify metadata on queries. This way, we can generate the queries to comply with the policy predicates expected.

Table 6.2 shows the operation distribution of these workloads and their application areas. Each of these workloads contains 1 million records which are inserted via *put* queries and 1 million operations on these records. In total, each workload contains 2 million queries. The distributions of the read/update/insert/read-modify-write operations are specified in Table 6.2 along with their real-world application scenarios.

In our project, we further customized the GDPRBench project to make it compatible with our own predicates. For example, we allow users to specify if logging of the query is required or not, how many users an entry is shared with, or the list of objections. In the following sections, these five workloads (A, B, C, D, F) are further customized to test the logging performance of the system such that logging is enabled for 0, 10, 20, 50, or 100% of the queries. There are also *vanilla* variants of each of these workloads that have no metadata. So, these are equivalent to the plain YCSB workloads. They only

Workload	Operation	Application
A	Read/Update (50/50%)	Session store
B	Read/Update (95/5%)	Photo tagging
C	Read (100%)	User profile cache
D	Read/Insert (95/5%)	User status update
F	Read-Modify-Write (100%)	User activity record

Table 6.2: YCSB workloads

Functionality	Native Controller	GDPR Controller
GDPR Metadata	no	yes
Policy Language	no	yes
Policy Compiler	no	yes
Query Rewriter	no	yes
Cipher Engine	optional	optional
Key-value Client	yes	yes
Logging Engine	no	optional
Policy Validator	no	yes

Table 6.3: Functionalities contained in Native and GDPR Controllers

contain query type, key, and optionally value. All the queries except the ones in *vanilla* workloads have all the metadata fields set explicitly. The metadata contains a session key, one objection, one purpose, one origin, and no expiration. The key size is 8 bytes and the value size is 1024 bytes for each entry.

Controllers: The GDPR Proxy Controller component of the system expects the queries to contain valid policies to succeed. In order to measure the overheads it causes, we implemented another component called *Native Controller*. This controller can work with queries that do not have any policy predicates. Therefore, it does not expect a default policy as well. Table 6.3 shows which components are used or not in Native and GDPR controller variants.

Number of clients: Another variant used in the evaluation step is the number of controllers. In real scenarios, databases handle multiple concurrent queries. This variant simulates them by running 1, 2, 4, 8, 16, or 32 clients at the same time. This is very useful for scale and concurrency testing.

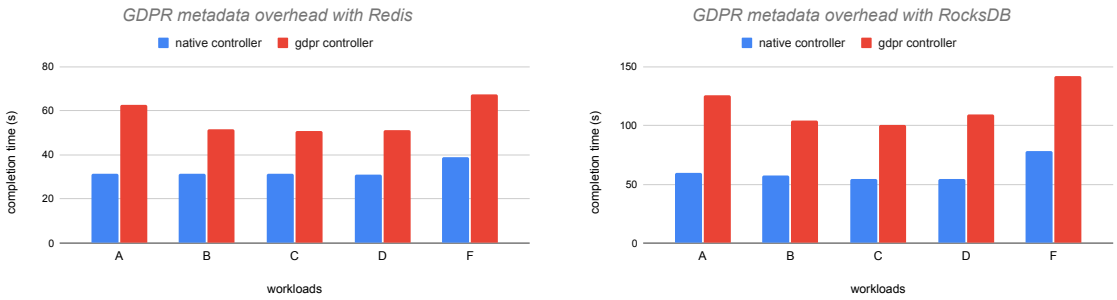

Figure 6.1: GDPR metadata overhead tests

Encryption: Encryption is another test variant. It is used to encrypt and decrypt both logs and values. We can enable and disable it for both GDPR and Native controllers to see how much overhead it adds to the overall system.

6.2 GDPR Metadata Overhead

GDPR compliance is achieved by extending the key-value pairs with additional metadata. Using excessive amounts of metadata can also result in metadata explosion as explained in section 3.4. Figure 6.1 shows the test results where we run the vanilla workloads with both GDPR Controller and Native Controller. As stated in Table 6.3, the Native Controller lacks most of the capabilities compared to the GDPR Controller. Thus, we expect the Native Controller to be more performant. In these scenarios, both logging and encryption capabilities are disabled to measure the metadata processing overhead directly. The tests are conducted with a single client running vanilla variants of workloads A-F. The average overhead of GDPR metadata processing of all workloads is 73% for Redis and 90% for RocksDB which is attributed to the metadata parsing, their storing in the Database and the respective copmliance checks performed by the GDPR controller.

6.3 Logging Overhead

Logging/monitoring is one of the most essential parts of GDPR compliance and provides proof of compliance. However, a high number of transactions can easily end up with huge log blocks and result in performance degradation. To see how the system responds to different levels of logging, we run GDPR Proxy Controller against

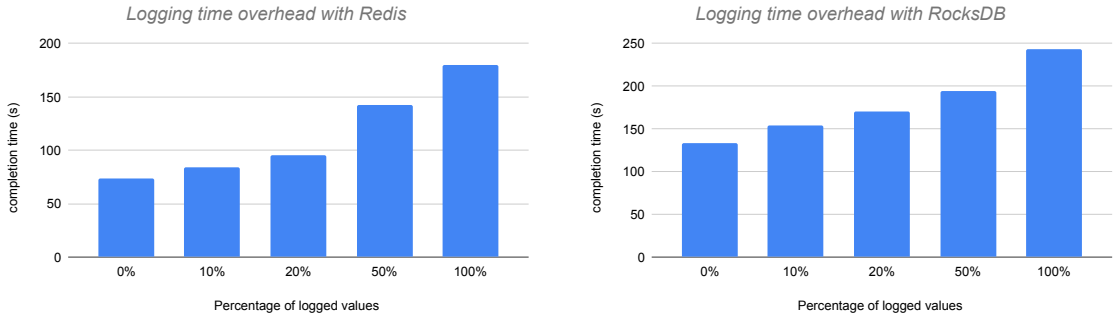

Figure 6.2: Logging overhead tests with one client, workload A, no encryption, GDPR controller, and varying logging percentages

different percentages of logging enabled for workload A. So, the only variable is the percentage of queries for which the logs are enabled. Encryption is again disabled to get the logging overhead directly and only one client is used. The results for both of the database instances are shown in Figure 6.2. For RocksDB, it takes 133 seconds for 0% and 243 seconds for 100%. On the other hand, for Redis, it takes 73 seconds for 0% and 179 seconds for 100%. So, logging all 2 million queries adds overheads of 145% for Redis and 82% for RocksDB instances by comparing 0% log and 100% log workload results. These overheads are justified due to the frequent I/O that the system performs on different files. Additionally, the reason for the big difference in percentages between the two of the datastore instances is that the logging overhead adds around 100 seconds to both of them but Redis is faster compared to RocksDB on log 0% workload. Therefore, 100 seconds difference adds a bigger relative overhead on Redis compared to RocksDB. Note that, 100% logging is a worst case scenario for our system, as typically not all the data processed within an application is considered private.

6.4 Value Encryption Overhead

The encryption layer is another core component of the system. Both logs and values can be encrypted. When encryption is enabled, it is enabled for both logs and values. In this section, the focus is on the value encryption overhead. In order to measure this independently, we use vanilla variants of workloads A-F, Native Controller, and one client. This way, although encryption is enabled, there will be no logging overheads. All results are shown in Figure 6.3. The average value encryption overhead for all the workloads is 23% for Redis and 24% for RocksDB.

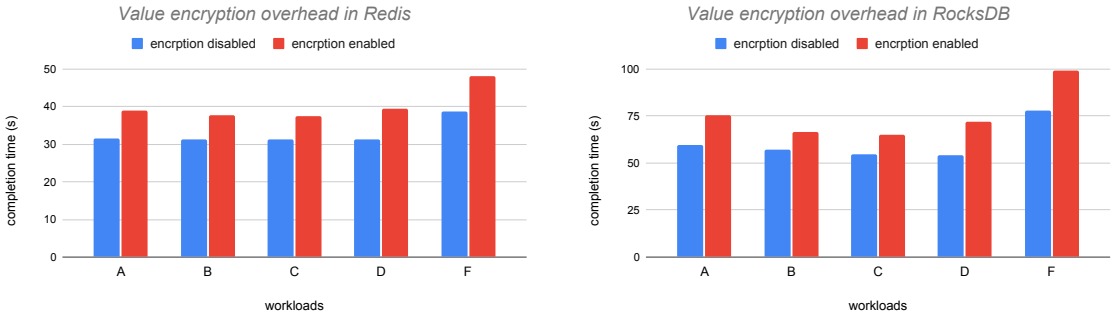

Figure 6.3: Value encryption overhead tests with one client, vanilla variants of workloads A-F, Native controller, and encryption enabled/disabled

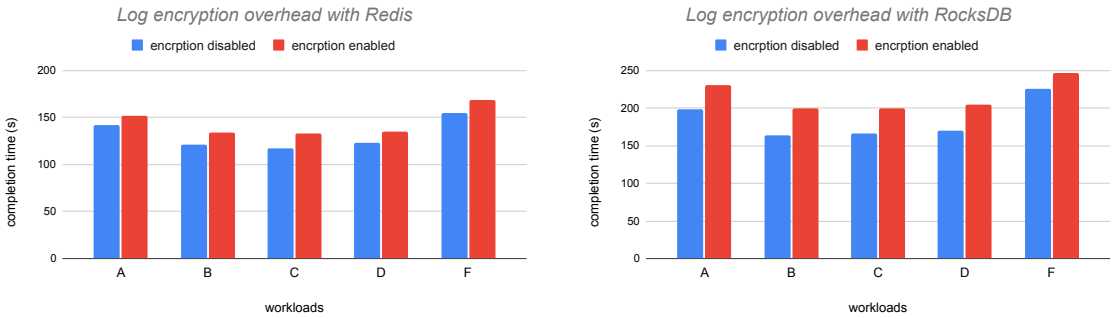

Figure 6.4: Log encryption overhead tests with one client, 50% logging variants of workloads A-F, GDPR controller, and encryption enabled/disabled

6.5 Log Encryption Overhead

As explained in the Value Encryption Overhead section above, encryption of logging and values are tied together. There is no way to measure the log encryption performance in an isolated way. However, we run the GDPR Proxy Controller against 50% logging enabled variants of workloads A-F with encryption enabled and disabled modes. Note that switching the encryption on brings the value encryption as well on top of the log encryption. Figure 6.4 represents the results of the related tests. The average overhead of log and value encryption overheads combined is 9.80% for Redis and 17.65% for RocksDB. The lower relative encryption overheads here, compared to the Value Encryption Overhead section, are observed as logging introduces further overheads which constitute the relative performance degradation due to encryption/decryption lower.

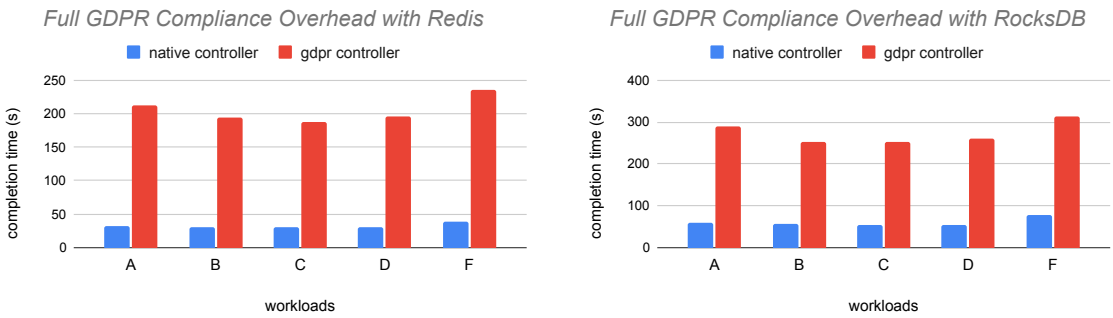

Figure 6.5: Full GDPR compliance tests with one client. For GDPR Controller, 100% logging variants of workloads A-F and encryption enabled. For Native Controller, vanilla variants of workloads A-F and encryption disabled

6.6 Full GDPR Compliance Overhead

The tests conducted so far measure some of the individual system components in an isolated manner by disabling some of the needs for GDPR compliance. The full compliance tests unlock all the features by enabling encryption and 100% logging. We compare these with the native controller running the vanilla variants of workloads A-F to see how much of a total overhead (metadata, encryption, logging, etc) is caused by the system. Figure 6.5 represents the results of the tests. The average overhead for Redis is 526% and it is 354% for RocksDB. Note that this is the worst case scenario, as we assume here that all the data is private. In normal workloads, only a part of the processed data is private and requires extensive auditable logging. To lessen this overhead, further optimizations such as more aggressive log compression, metadata indexing and asynchronous data flushing can be enforced.

6.7 Concurrency and Scaling

The GDPR-Compliant Proxy System is designed to handle multiple concurrent sessions. The concurrency tests are critical to showcase the system's scalability and correctness under heavy and concurrent loads. These tests are conducted using GDPR Controller, 100% logging variant of workload A, and encryption enabled mode. It is clear from the Figure 6.6 that increasing the number of concurrent clients does not result in the same

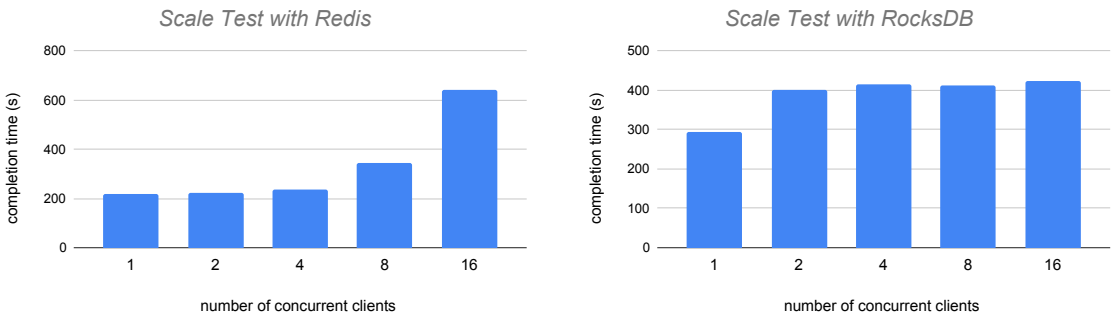

Figure 6.6: Scale test with different numbers of concurrent clients under GDPR controller with encryption enabled, and 100% logging variant of workload A

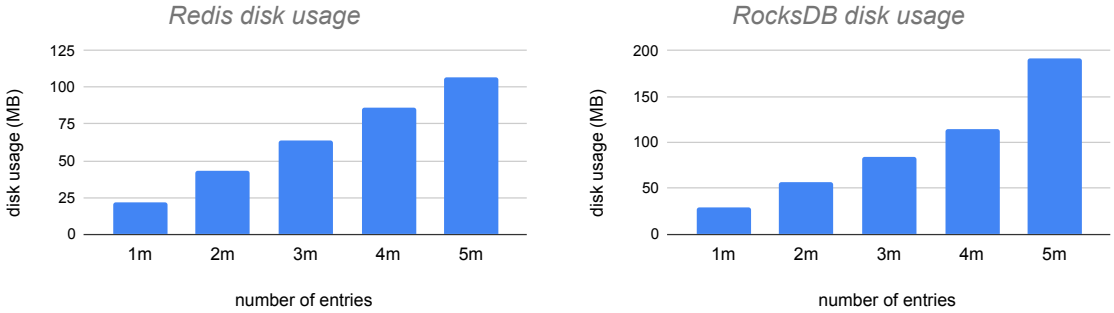

Figure 6.7: Key-value datastore disk usages after inserting given number of entries

percentage of increase in the completion time. RocksDB seems to be more performant to deal with the load as only 44% of average overhead is observed between the tests with 1 and 16 concurrent clients. The same overhead for Redis is 196%. These results show that although the number of queries increases by 1600%, its reflection on the overall system completion time is way less and this proves the system's concurrency capabilities while preserving the correctness.

6.8 Disk usage

Extending the datastore values and logs increase the need for disk space. In this section, we write 1, 2, 3, 4, and 5 million entries to the key-value store using put queries. We enable the logging for all the values (100%) and disable the encryption.

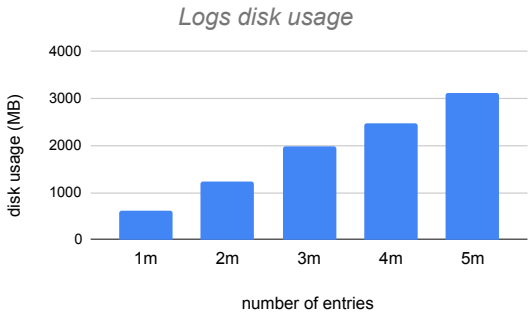

Figure 6.8: Logs disk usages after inserting given number of entries

Figure 6.7 represents the disk space used by RocksDB and Redis. The disk usage is close to being linear with increasing entries. Figure 6.8 shows that the increase in disk space used for logging has a linear increase while increasing the number of database entries, as all of them need to be logged. The additional data that the logs and the database entries require are reasonable and mandatory so that our GDPR proxy controller can provide the auditable logs and perform the GDPR compliant access control for the private data. These overheads can be further minimized in the future by enforcing a more aggressive log entry and metadata compression algorithm.

7 Related Work

7.1 GDPR Benchmarking tools

Checking and benchmarking GDPR compliance is a critical step for data controllers to prove when they are asked. GDPRBench [15] offers the design and implementation of a tool that generates workloads with metadata. The tool is based on YCSB [5] and extends the YCSB workloads with metadata. The generated workloads challenge the systems in terms of correctness and performance under GDPR. Users of the tool provide as input the configuration for workloads such as the percentages of write operations, read operations, update operations, and some other distribution-related settings. The tool then generates the actual workload files containing the given number of records and operations. GDPRBench open source tool's code can be found on `https://github.com/GDPRbench/GDPRbench`.

7.2 GDPR Compliance Tools/Methodologies

At the time of writing this thesis, there are several articles focusing on GDPR compliance in database systems [2, 3, 9, 11, 14]. In the following paragraphs, some key ones are mentioned.

Position: GDPR Compliance by Construction [14]: This work proposes a web service backend architecture that puts users in control of their data, and which aims to be GDPR-compliant by construction. The focus area is the applications with relational databases. The paper proposes to keep each user's data in shards where each shard is isolated and used to store only one user's information. The shards are kept encrypted for better security. Shards are updated while the users are interacting with the system. The paper introduces materialized views on top of the shards and these allow different applications to reach the user data based on their needs and render them. The only way for applications or processors to reach the user data is through the already-defined materialized views. The materialized views are basically some other database schemas. Each update to a shard is streamed to the interested materialized views via a data flow.

The authors mention the possibility to extend the relational database application area of the proposed method and to apply it on blob stores, logs, notifications, etc. Although it is not mentioned, this can also be applied to the key-value entries.

The advantage of the proposed architecture is to keep the users' data in logical blocks and make it easily accessible to the data owners. The other advantage is its extensibility to potentially work with all kinds of data storage units. The main disadvantage of the system is its complex architecture.

SchengenDB: A Data Protection Database Proposal [11]: This paper mainly focuses on the right to be forgotten requirements. It again focuses on relational database systems and approaches the problem with two possible cases.

The first case is the "Green Field" applications which contain applications to be redesigned or created from scratch. For this case, the authors propose an Entity-Relationship diagram to store the user information once and avoid repeated materialized views. This approach eases the deletion of user data.

The second case is the existing schema within a system. This one is proposed when the schema redesign is not cost-effective. To handle the case, each insert operation to the database is tied to a user (data owner). Furthermore, every derived record (like materialized view) inherits owners of the parent records. This allows the system to delete all the records that are associated with a user when he/she asks to do so.

Retrofiting GDPR Compliance onto Legacy Databases [2]: This paper introduces a tool called GDPRizer. The tool focuses on relational database systems and aims to help database administrators to extract and delete users' information on legacy databases with unorganized schemas. The goal of the project is to generate a set of queries that extract or delete an individual's information in accordance with data access requests.

The authors base the GDPR compliance on the relationship graphs. A relationship graph represents the relationships of columns across tables. Each vertex means a column and each edge means a foreign key relationship between columns. A table containing multiple columns is represented as a group of vertices. In order to construct the graph, the tool makes use of database schema, queries, and data patterns. The database schema alone is not enough information since not each schema perfectly represents the data. The consideration of queries helps because when a query contains a join statement, the tool considers the joining columns are likely related. The tool also leverages the stored data to detect relationships through some statistical patterns explained in the paper. The database administrators are also allowed to manually change the graph structure. This way, acting as a plugin, the tool intercepts the queries

from database administrators or developers and limits the data returned or deleted for a user to keep the other users' information private.

Summary: Our literature review on GDPR compliance from the database level shows a high interest in bringing GDPR compliance to the relational databases while skipping NoSQL databases. The NoSQL databases, especially the key-value stores, have been the subject of benchmarking projects which we also use in this project. Most of the proposed compliance tools do not conduct a comprehensive test or benchmarking. They also do not reveal the GDPR articles as a whole but they focus on some of the most well-known ones such as the right to be forgotten.

8 Summary and Conclusion

GDPR requirements explained in Section 2.2 made applications and systems undergo big changes and face performance degradation. In this thesis, we proposed a generic and easy way to comply with the requirements at the database level without changing the database internals. We focused on key-value datastores and designed, implemented, and evaluated a robust and GDPR-compliant proxy system.

We addressed the GDPR requirements by implementing encryption, activity logging, metadata extension, and a policy language. We exposed a secure TCP channel to interact with data controllers, data owners, data processors, and regulators through the proxy controller component. The clients can send their queries to this component. The data access is checked using the stored metadata and given policy. The access control includes identity, expiration, purpose, objection, and origin checks. We logged both the successful and failed query attempts for the data controller's accountability requirements. In order to keep the data safe in untrusted data storage environments, both values and audit logs are kept encrypted.

One of the advantages of the system is its easy integration with various key-value datastores. We showcased this ability with Redis and RocskDB instances and explained the two simple steps for compatibility with other instances in Section 5.2.

In order to test the system, we used the GDPRBench tool to check correctness, speed, disk usage, and scalability. We customized the tool to comply with our policy interface and ran various large workloads.

In conclusion, the GDPR Compliant Proxy system for key-value datastores provides a generic solution to foster user rights and privacy. It promises easy integration and extension to the current systems without modifying the database internals. It removes the burden of GDPR compliance from application developers and allows them to focus on their core business logic. The source code, evaluation scripts, and low-level code explanation can be found on `https://github.com/dimstav23/GDPRuler`.

9 Future Work

The GDPR Compliant Proxy System can be further improved in terms of performance and features. These areas are listed in the following paragraphs.

9.1 Query metadata implementation

Although the system is designed to have *getm, putm, delm* queries, the implementation for those is not complete yet. At the time of writing this paper, the metadata handling is done by *get*, *put*, *delete* queries. Each *put* query updates the metadata as well along with the user value. Implementing metadata queries and separating the value and metadata handling parts can make the interface simpler to work with.

9.2 Logging performance improvement

As explained in Section 6.3, Logging Overhead, activity logging is one of the obvious bottlenecks of the system. It affects the performance of the system more than any other component or flow. Below are some ways to reduce its effect on the system.

1. The logs for a *put* query contain the new value as well. To reduce the length of a log record, this can be omitted. Moreover, further compression of the logs can be researched.
2. The logs are written synchronously to the filesystem before returning the result of any query back to the client. Instead, this can be done asynchronously to have a better performance. The log record can be submitted to an in-memory or a third-party queue and a log writer thread pool can poll it and write the logs to the filesystem.
3. The logs are written to the proxy controller's filesystem. In order to scale better, third-party tools such as Elasticsearch can be used. This not only unlocks a new feature to search the logs by a keyword but also improves the scalability and reliability of the system.

9.3 Access to all data of a user

Based on GDPR regulations, a user can ask for all of his/her data stored by the controller. The GDPR Proxy System enables data access based on keys and stores the data owner information in values' metadata. In order to get all data of a user with the current state of the system, all of the values in the system need to be fetched, decrypted, and checked if their owner matches the requester. This would be a very costly operation. Instead, we can store a mapping from users to the set of keys that they own. This way, all the keys a user has can be quickly fetched and then the current *get* query can be used to fetch the corresponding values.

9.4 System interface with common protocols

The GDPR Proxy Controller component exposes a secure TCP port for clients to connect with and send queries to. Although it is a performant way to expose the system, it is not easily accessible, especially for the data owners. On top of TCP, Representational State Transfer (REST) endpoints can be exposed as well. The REST verbs (POST, PUT, GET, DELETE) can be matched with the system's queries. The query predicates can be embedded in the request header or body. And the REST status codes can be used to reveal query failure reasons. The advantages of this protocol are easier access for users and easier integrity with the other tools as REST is one of the most popular protocols these days. The main disadvantage is the increase in the request and response payloads.

Abbreviations

AES-GCM Advanced Encryption Standard - Galois/Counter Mode

API Application Programming Interface

EU European Union

GDPR General Data Protection Regulation

IoT Internet of Things

IV Initialization Vector

JSON JavaScript Object Notation

MAC Message Authentication Codes

MB Megabytes

NoSQL Not Only SQL

NUMA Non-Uniform Memory Access

REST Representational State Transfer

SQL Structured Query Language

TCP Transmission Control Protocol

YCSB Yahoo! Cloud Serving Benchmark

List of Figures

2.1 Relations between GDPR entities 4

3.1 High-level architecture of the GDPR-compliant proxy system 12

4.1 GDPR Proxy's detailed design diagram 17
4.2 Cipher Engine's interaction with the other components 24

5.1 Key-value clients class diagram 27
5.2 Unencrypted log record structure 28
5.3 Encrypted log record structure 29
5.4 Encrypted text storage structure 30

6.1 GDPR metadata overhead tests 34
6.2 Logging overhead tests with one client, workload A, no encryption, GDPR controller, and varying logging percentages 35
6.3 Value encryption overhead tests with one client, vanilla variants of workloads A-F, Native controller, and encryption enabled/disabled . . 36
6.4 Log encryption overhead tests with one client, 50% logging variants of workloads A-F, GDPR controller, and encryption enabled/disabled . . . 36
6.5 Full GDPR compliance tests with one client. For GDPR Controller, 100% logging variants of workloads A-F and encryption enabled. For Native Controller, vanilla variants of workloads A-F and encryption disabled . 37
6.6 Scale test with different numbers of concurrent clients under GDPR controller with encryption enabled, and 100% logging variant of workload A 38
6.7 Key-value datastore disk usages after inserting given number of entries 38
6.8 Logs disk usages after inserting given number of entries 39

List of Tables

6.1 Features of a NUMA node used for the experiments 32
6.2 YCSB workloads 33
6.3 Functionalities contained in Native and GDPR Controllers 33

Bibliography

[1] *1 year GDPR – taking stock*. 2019. URL: https://edpb.europa.eu/news/news/2019/1-year-gdpr-taking-stock_en.

[2] Archita Agarwal et al. "Retrofitting GDPR Compliance onto Legacy Databases." In: *Proc. VLDB Endow.* 15.4 (Dec. 2021), pp. 958–970. ISSN: 2150-8097. DOI: 10.14778/3503585.3503603. URL: https://doi.org/10.14778/3503585.3503603.

[3] Kinan Dak Albab et al. "K9db: Privacy-Compliant Storage For Web Applications By Construction." In: *17th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2023, Boston, MA, USA, July 10-12, 2023*. Ed. by Roxana Geambasu and Ed Nightingale. USENIX Association, 2023, pp. 99–116. URL: https://www.usenix.org/conference/osdi23/presentation/albab.

[4] Shanshan Chen et al. "Towards Scalable and Reliable In-Memory Storage System: A Case Study with Redis." In: *2016 IEEE Trustcom/BigDataSE/ISPA*. 2016, pp. 1660–1667. DOI: 10.1109/TrustCom.2016.0255.

[5] Brian Cooper et al. "Benchmarking cloud serving systems with YCSB." In: Sept. 2010, pp. 143–154. DOI: 10.1145/1807128.1807152.

[6] *Data protection under GDPR*. 2022. URL: https://europa.eu/youreurope/business/dealing-with-customers/data-protection/data-protection-gdpr.

[7] Siying Dong et al. "RocksDB: Evolution of Development Priorities in a Key-Value Store Serving Large-Scale Applications." In: *ACM Trans. Storage* 17.4 (Oct. 2021). ISSN: 1553-3077. DOI: 10.1145/3483840. URL: https://doi.org/10.1145/3483840.

[8] Morris Dworkin. "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC." In: *NIST Special Publication 800-38D* (2007).

[9] M. Ferreira et al. "RuleKeeper: GDPR-Aware Personal Data Compliance for Web Frameworks." In: *2023 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2023, pp. 2817–2834. DOI: 10.1109/SP46215.2023.00058. URL: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.00058.

[10] Nikhil Dasharath Karande. "A Survey Paper on NoSQL Databases: Key-Value Data Stores and Document Stores." In: *International Journal of Research in Advent Technology* 6.2 (2018). ISSN: 2321-9637.

[11] Tim Kraska et al. "SchengenDB: A Data Protection Database Proposal." In: *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Ed. by Vijay Gadepally et al. Cham: Springer International Publishing, 2019, pp. 24–38. ISBN: 978-3-030-33752-0.

[12] "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)." In: *Offcial Journal of the European Union* (2016). URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[13] Renae Reints. "These Major U.S. News Sites Are Blocked in the EU." In: *Fortune: GDPR* (2018). URL: https://fortune.com/2018/08/09/news-sites-blocked-gdpr/.

[14] Malte Schwarzkopf et al. "Position: GDPR Compliance by Construction." In: *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Berlin, Heidelberg: Springer-Verlag, 2022, pp. 39–53. ISBN: 978-3-030-33751-3. DOI: 10.1007/978-3-030-33752-0_3. URL: https://doi.org/10.1007/978-3-030-33752-0_3.

[15] Supreeth Shastri et al. "Understanding and Benchmarking the Impact of GDPR on Database Systems." In: *Proceedings of the VLDB Endowment* 13.7 (Mar. 2020), pp. 1064–1077. DOI: 10.14778/3384345.3384354. URL: https://doi.org/10.14778%2F3384345.3384354.

[16] *Statement ahead of the 5th anniversary of the General Data Protection Regulation*. 2023. URL: https://ec.europa.eu/commission/presscorner/detail/en/statement_23_2884.