Tamperproof Logging System for GDPR-compliant Key-Value Stores

Christian Karidas
Advisor: Dimitrios Stavrakakis
Chair of Computer Systems
https://dse.in.tum.de/

15.01.2025 – 16.06.2025

Motivation

- GDPR requires comprehensive audit trails for personal data processing
- Key-value stores are designed for simplicity and high-performance—not for fine-grained compliance logging
- Logging is crucial for demonstrating accountability—but existing systems are either insecure, too complex or slow

Background - GDPR

Since 2018, the GDPR mandates strict rules on how personal data is handled.

- **Accountability - Art. 5(2):** demonstrate compliance with data protection principles
- **Integrity and Confidentiality - Art. 5(1)(f):** ensure security against unauthorized access, alteration, or deletion
- **Lawfulness & Transparency - Art. 5(1)(a):** processing activities can be audited and reviewed

Comprehensive audit trails with encryption and tamper-evidence

State-of-the-art & research gap

Three categories, one limitation each:

- High-performance logging (NanoLog[1], CORFU[2]): Fast but no security/encryption
- Secure logging (Forward-secure schemes, blockchain): Strong integrity but slow/complex
- GDPR-compliant systems: Specialized solutions, not general-purpose

No system combines high performance, encryption, tamper-evidence and easy integration

[1] NanoLog: https://www.usenix.org/conference/atc18/presentation/yang-stephen
[2] CORFU: https://www.cs.cornell.edu/courses/cs5414/2017fa/papers/Corfu.pdf

Problem statement

Can we build a logging system that enables compliance with GDPR, seamlessly integrates into database systems without changes to the database architecture, and minimizes performance and storage overhead?

Key requirements:

- Capable of high-volume log handling -> High throughput, low latency, scalability
- Reduced disk footprint -> Compression
- Data confidentiality -> Encryption
- Cryptographic tamper detection -> Cryptographic measurements

Outline

- ~~Background & Motivation~~
- Design
- Implementation
- Evaluation
- Discussion & Future Work

System overview

Key design properties:

- Multi-threaded architecture for scalability
- Buffering to achieve high throughput
- Compression to minimize disk footprint
- Encryption for data confidentiality
- Cryptographic tamper detection for integrity & authenticity

Design details – log entry structure

destination filename: **user_42.log**

Log entry

Operation type: **READ**

Data location: **user_42/email**

Data controller: **ctrl_1**

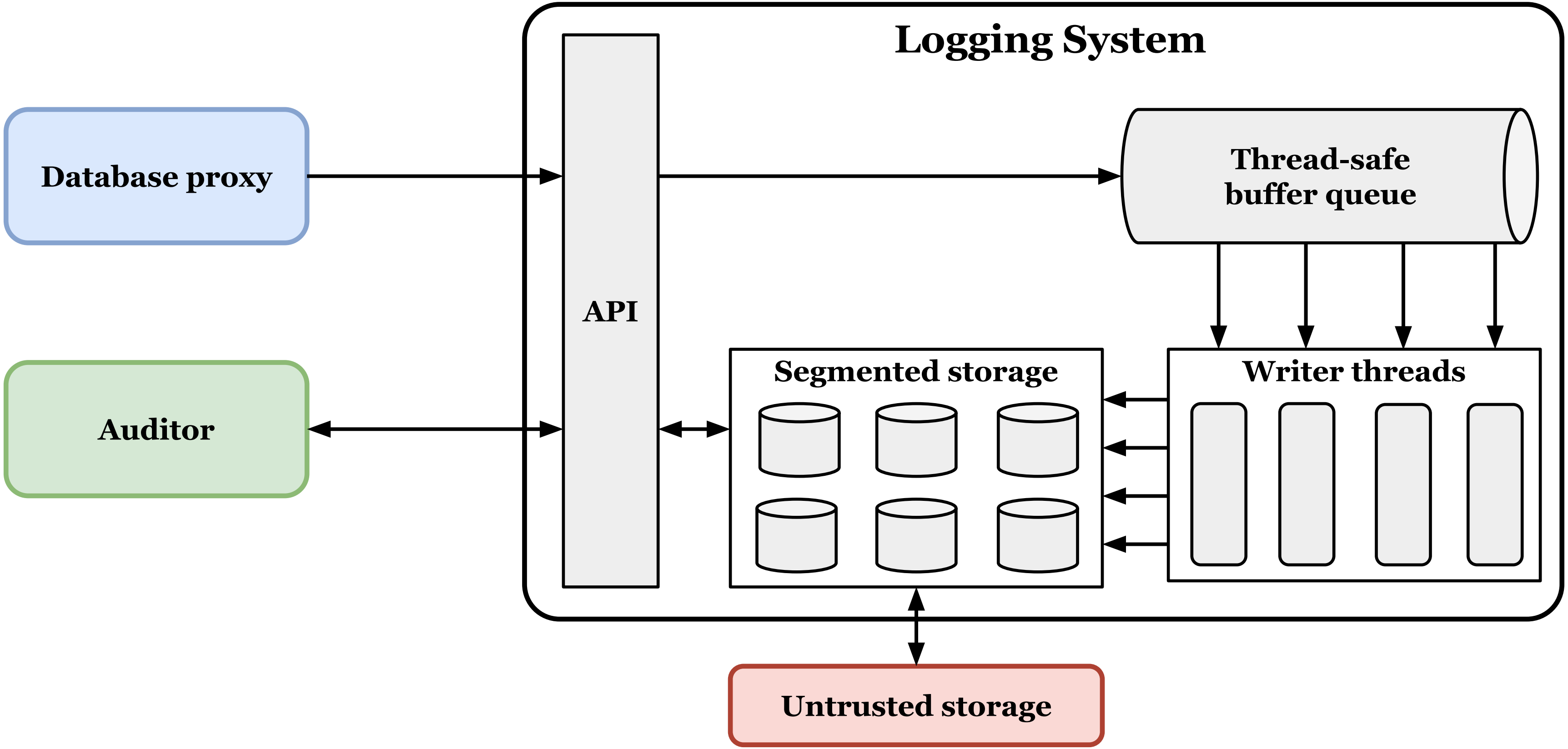
Data processor: **prcs_1**

Data subject: **user_42**

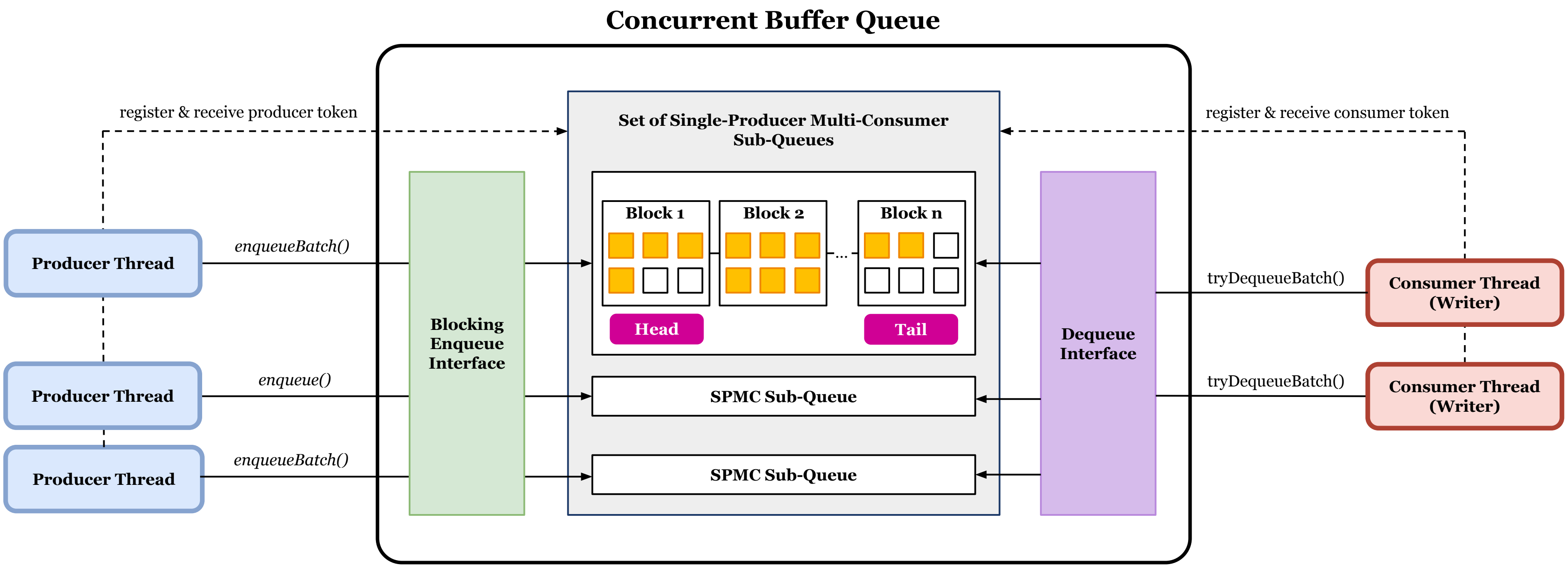
Timestamp: **16-06-25/09:59:32**

Additional payload: **purpose = fulfilling user request**

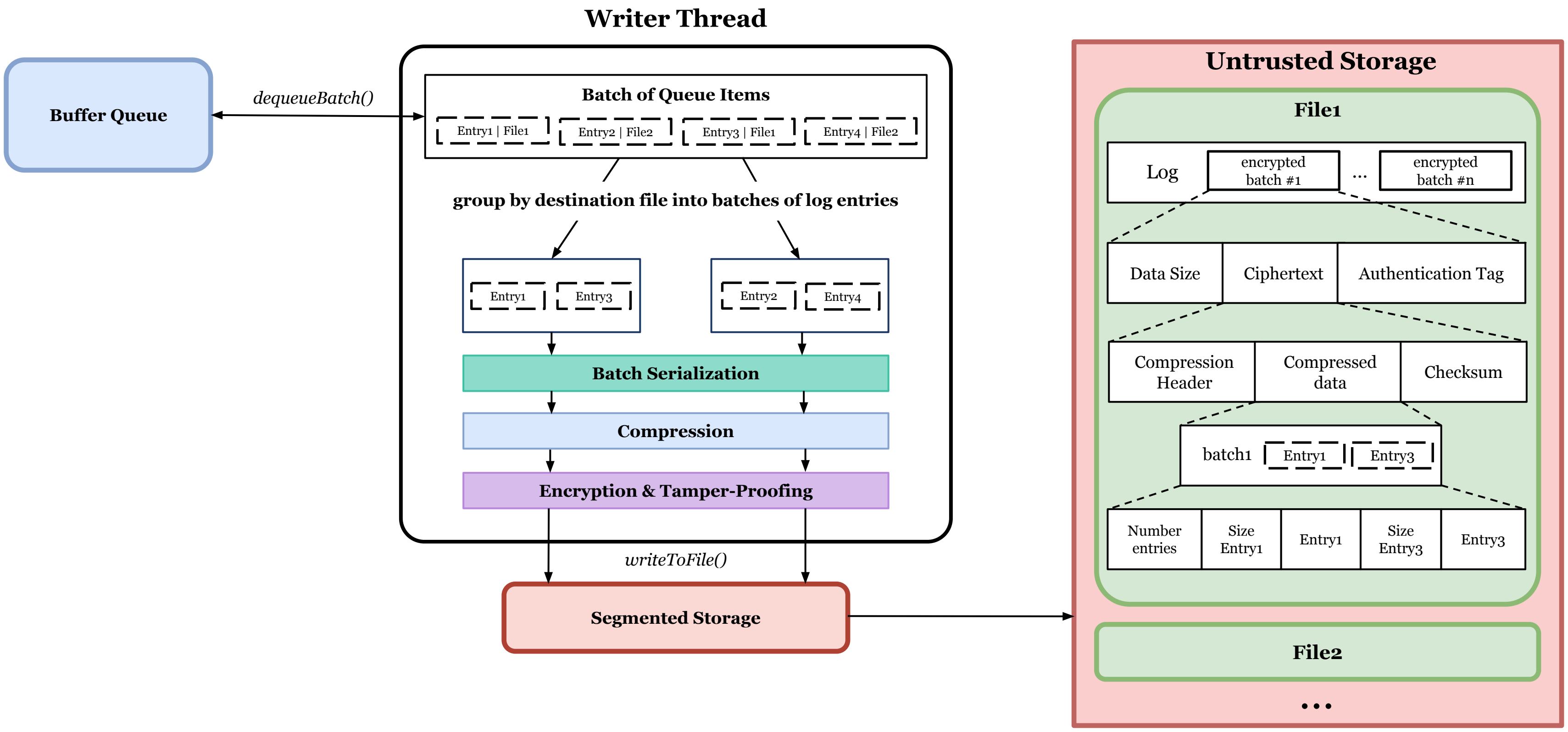
System overview



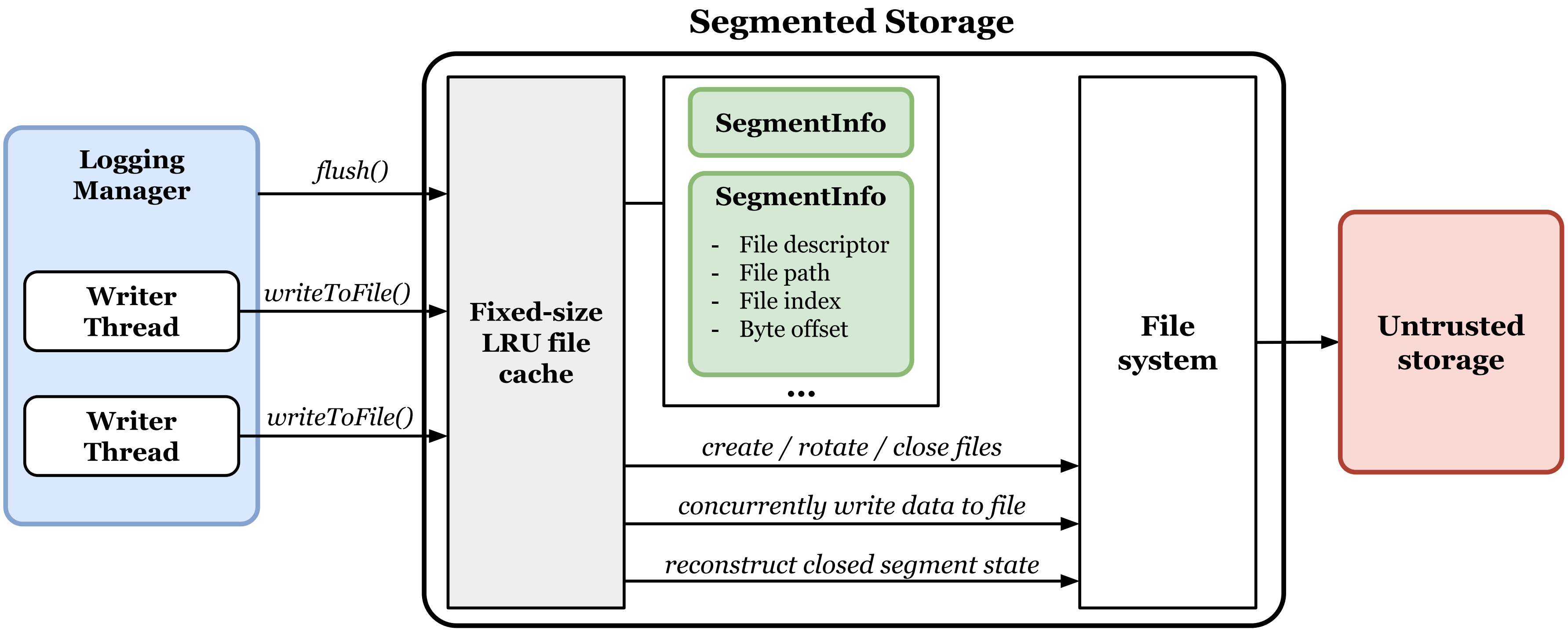
Design details – concurrent buffer queue



Design details – writer threads



Design details – segmented storage



Outline

- ~~Background & Motivation~~
- ~~Design~~
- Implementation
- Evaluation
- Discussion & Future Work

Implementation details

Language: **C++**

- used for high performance and fine-grained control
- system designed as a standalone **C++ library**

Libraries used:

- **AES-GCM (OpenSSL[1])** for authenticated encryption
- **zlib[2]** for compression
- **moodycamel::ConcurrentQueue[3]** (popular C++ library, fast multi-producer, multi-consumer lock-free concurrent queue)

[1] Open SSL library: https://openssl-library.org
[2] zlib compression: https://zlib.net
[3] moodycamel::ConcurrentQueue: https://moodycamel.com/blog/2014/a-fast-general-purpose-lock-free-queue-for-c++

System configuration options

The key parameters of the system config are:

- Writer batch size
- Number of writer threads
- Encryption usage
- Compression level
- Queue capacity
- Maximum log file size
- Maximum parallelly opened log files

Outline

- ~~Background & Motivation~~
- ~~Design~~
- ~~Implementation~~
- Evaluation
- Discussion & Future Work

Evaluation

- *What is the optimal system configuration for handling heavy workloads?*
- *How well does the system scale with increasing resources / workloads?*
- *What is the effect of compression and encryption on system performance?*
- *How does the system perform under demanding workloads?*
    - throughput
    - client latency
    - write amplification

Evaluation

Hardware environment

- **CPU:** 2x Intel Xenon Gold 6236 (32 cores, 64 threads)
- **Memory:** 320 GiB DDR4-3200 ECC
- **Storage:** Intel S4510 SSD (960GB)
- **OS:** NixOS 24.11, ZFS filesystem
- **NUMA-optimized** execution (pinned to single node)

Evaluation

Synthetic Log Data Generation:

- **4096-byte payloads**
- **controlled compression ratios***

  across different zlib levels

zlib Level	Compression Ratio
0	1.00
1	9.15
2	10.88
3	13.15
4	13.74
5	15.54
6	17.44
7	18.18
8	19.55
9	20.65

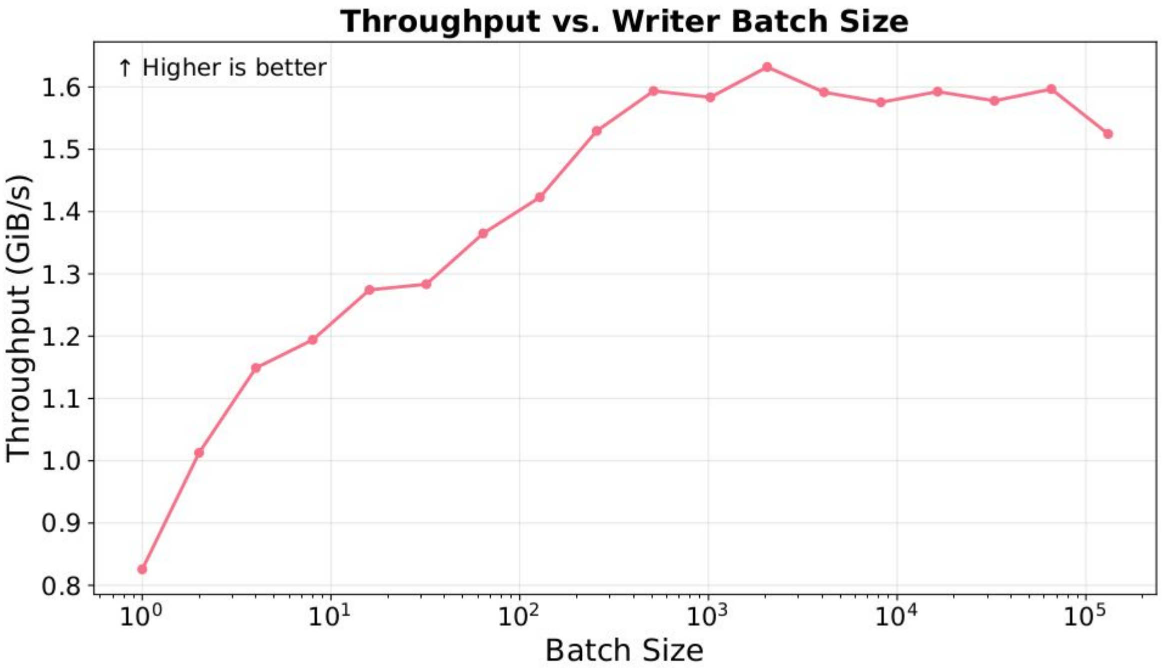
*compression ratio = uncompressed bytes / compressed bytes

Optimal system configuration

Example: writer batch size

Workload traits:

- asynchronous log producers: **16**
- entries per producer: **2.000.000**
- entry size: **~4 KiB**
- total Input Data Volume: **~125 GiB**
- producer batch size: **4096**
- queue capacity: **~2.000.000**
- encryption: **enabled**
- compression level: **4, balanced fast**



Optimal system configuration

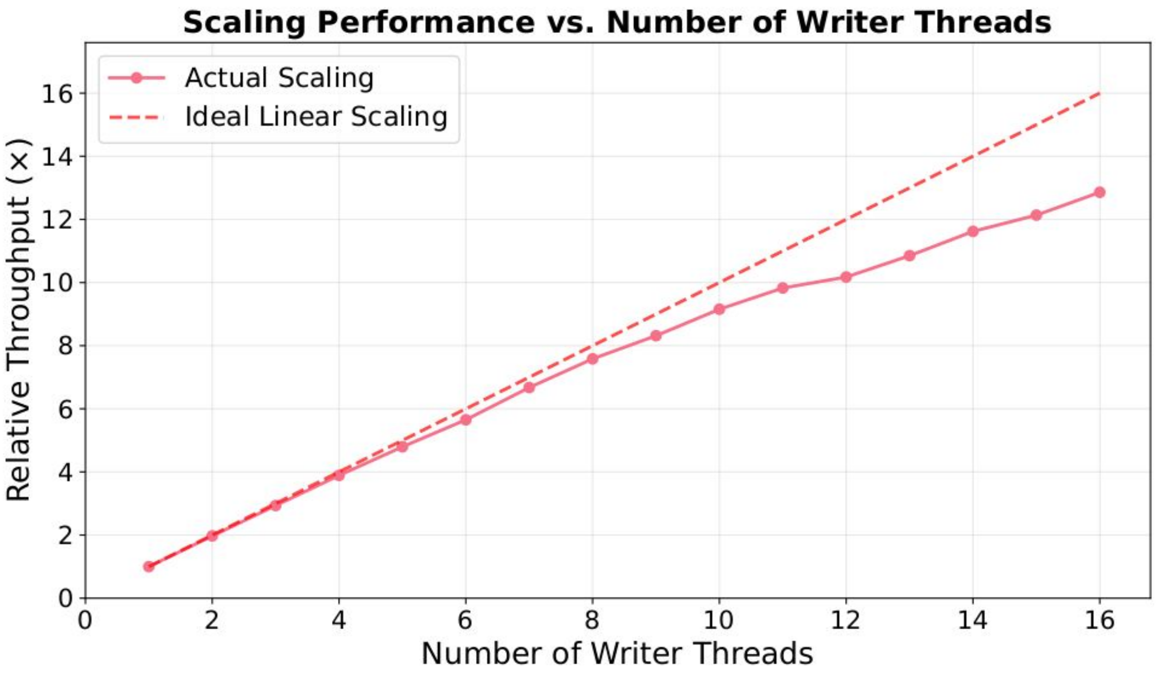
Empirical best-performing parameters:

- writer batch size: 2048 entries
- queue capacity: 2.000.000 (internally rounded to 2.097.152)
- compression level: 1, very fast – least aggressive
- …

System scalability

Workload traits:

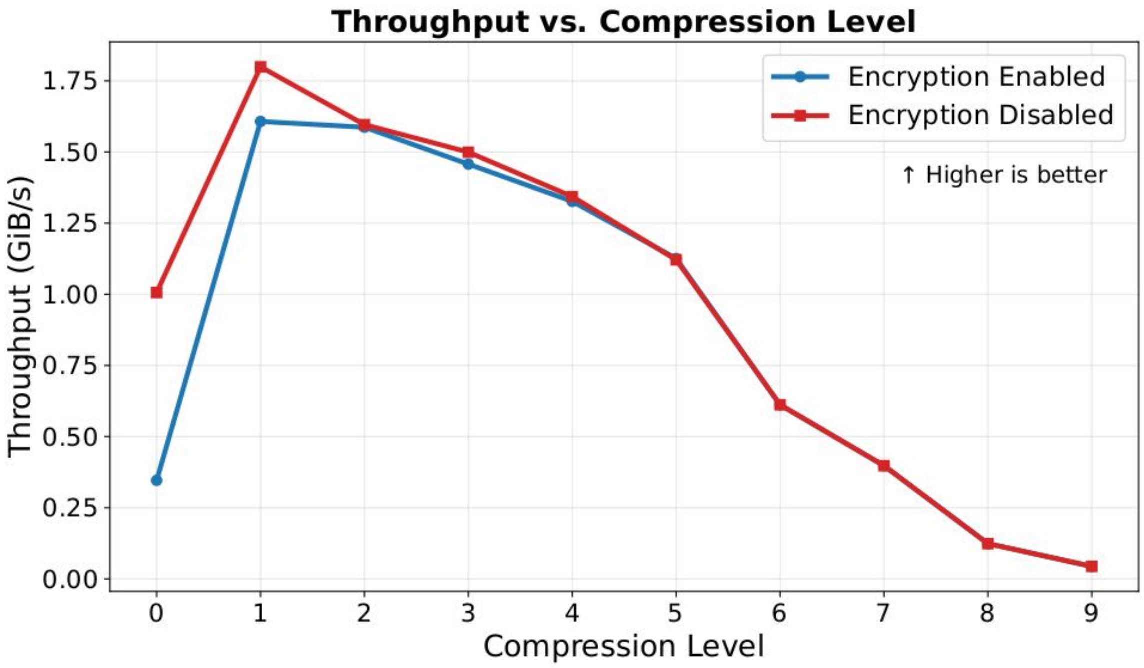
- number of log producers scaled **1:1** with writers
- entries per producer: **2.000.000**
- entry size: **~4 KiB**
- writer batch size: **2048**
- producer batch size: **4096**
- queue capacity: **~2.000.000**
- encryption: **enabled**
- compression level: **4, balanced fast**



Encryption & compression overhead

Workload traits:

- asynchronous log producers: **16**
- entries per producer: **300.000**
- entry size: **~4 KiB**
- total Input Data Volume: **~18.69 GiB**
- writer batch size: **2048**
- producer batch size: **4096**
- queue capacity: **~2.000.000**



System performance

Workload traits:

- asynchronous log producers: **16**
- entries per producer: **2.000.000**
- entry size: **~4 KiB**
- total Input Data Volume: **~125 GiB**
- writer batch size: **2048**
- producer batch size: **4096**
- queue capacity: **~2.000.000**
- encryption: **enabled**
- compression level: **1, very fast**

Metric	**Value**
Execution Time	59.95 seconds
Throughput (Entries)	~533,711 entries/sec
Throughput (Data)	~2.08 GiB/sec
Latency	Med.: 54.7ms, Avg.: 55.9ms, Max: 182.7ms
Write Amplification	0.109
Final Storage Footprint	~13.62 GiB for ~124.6 GiB input data

Outline

- ~~Background & Motivation~~
- ~~Design~~
- ~~Implementation~~
- ~~Evaluation~~
- Discussion & Future Work

Discussion & Future Work

Current Limitations:

- No direct log export functionality
- Static encryption key & IV
- Incomplete freshness properties
- Infrequent disk flushes (trade-off for performance)

System remains a prototype – solid foundation for secure & performant logging

Conclusion

Designed and implemented a **secure, tamper-evident, performant** and **modular** logging system for GDPR compliance.

Impact:

- Enables verifiable audit trails with minimal integration effort
- Supports GDPR accountability with high performance
- Lays groundwork for future improvements (e.g. key management, export)

Demonstrates feasibility of combining regulatory compliance and security with high performance

Complete implementation: https://github.com/chriskari/bachelor-thesis

Quod erat demonstrandum.