GDPRuler: A Trusted GDPR Monitor for Cloud Data Systems

Dimitrios Stavrakakis
TU Munich
Munich, Germany
dimitrios.stavrakakis@tum.de

Masanori Misono
TU Munich
Munich, Germany
masanori.misono@in.tum.de

Julian Pritzi
TU Munich
Munich, Germany
julian.pritzi@tum.de

Harshavardhan Unnibhavi
TU Munich
Munich, Germany
harshavardhan.unnibhavi@tum.de

Nuno Santos
INESC-ID / Instituto
Superior Tecnico, University of Lisbon
Lisbon, Portugal
nuno.m.santos@tecnico.ulisboa.pt

Pramod Bhatotia
TU Munich
Munich, Germany
pramod.bhatotia@tum.de

Abstract

Privacy regulations such as the General Data Protection Regulation (GDPR) impose strict requirements on how personal data is stored, processed, and audited. While key-value stores (KVS) are widely used in latency-sensitive applications, their simple data model and untrusted cloud deployment environments make GDPR compliance particularly challenging. Existing approaches require invasive code modifications, impose high performance overheads, or overlook the integrity of compliance mechanisms themselves.

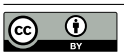
This paper presents GDPRuler, a trusted middleware system that enables verifiable GDPR compliance for KVS on untrusted clouds without modifying their codebase. GDPRuler deploys a *trusted GDPR monitor* inside a Confidential Virtual Machine (CVM), which enforces GDPR policies, manages compliance metadata, and maintains tamper-evident audit logs. A declarative policy language translates core GDPR obligations into enforceable runtime rules. To ensure efficiency, GDPRuler encodes metadata compactly within KV records, builds dedicated metadata indexes for GDPR-specific queries, and logs only compliance-relevant events in a space-efficient format. We implement GDPRuler as a transparent proxy for unmodified Redis and RocksDB deployments. Evaluation with YCSB and GDPR-inspired workloads shows that GDPRuler enforces core compliance guarantees with low overheads: GDPRuler achieves ≈61% of native KVS throughput with the CVM environment contributing 28%-32% of it, metadata storage overhead remains below 20%, and GDPR queries benefit from 13-182× speedup through metadata indexing. By embedding verifiable policy enforcement into a trusted middleware layer, GDPRuler offers a practical path toward GDPR-compliant KVS on untrusted cloud infrastructures.

CCS Concepts

• **Security and privacy → Database and storage security**; **Systems security**; **Software and application security**.

Keywords

Confidential computing; GDPR; Security; Database; Storage; Key-Value Store; Policy Language; Privacy; Audit Logging



This work is licensed under a Creative Commons Attribution 4.0 International License.
CCS '26, The Hague, Netherlands
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3832582

ACM Reference Format:
Dimitrios Stavrakakis, Masanori Misono, Julian Pritzi, Harshavardhan Unnibhavi, Nuno Santos, and Pramod Bhatotia. 2026. GDPRuler: A Trusted GDPR Monitor for Cloud Data Systems. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3830454.3832582

1 Introduction

The rapid move of data processing and storage to the cloud has shifted control of sensitive information from individuals and organizations to untrusted third-party infrastructures, intensifying the concerns over data security, privacy, and anonymity. Across diverse cloud data architectures—from data warehouses to distributed data processing systems—modern privacy regulations such as the General Data Protection Regulation (GDPR) [51] require data controllers to operationalize users' rights—such as access and erasure [37, 38]—and to uphold principles of purpose and storage limitation [39] directly in their storage and processing pipelines. Non-compliance can result in severe financial and reputational penalties [25, 63, 75, 76], motivating systems that can not only protect data but also *record, enforce, and prove* compliance actions through auditable operations on stored data. Achieving this goal in cloud deployments, however, remains difficult: privileged software stacks controlled by cloud operators can observe or tamper with data and audit trails, undermining both confidentiality and the integrity of compliance evidence.

Key-Value Stores (KVS) [7, 28, 69, 71, 85] constitute a core component of modern cloud data architectures, exemplifying the tension between simplicity and compliance. They underpin a wide range of modern data services, from caching layers to IoT pipelines, but their flat key-value (KV) abstraction provides no native support for compliance metadata, secondary indexes, or expressive queries. As a result, mechanisms such as purpose-based access must be implemented above the data layer, through ad hoc application logic that offers little assurance of correctness or completeness. Unlike relational databases, KVS lack schemas and query semantics that would allow them to represent and reason about regulatory state.

A fundamental design question is thus *where and how to enforce privacy-aware compliance*. Embedding GDPR mechanisms directly into KV engines (e.g., Redis [85], RocksDB [71], LevelDB [46]) would tightly couple regulatory policy with system internals, complicating upgrades and breaking compatibility. Delegating enforcement to the application layer scatters compliance checks and forfeits verifiability.

We argue that effective GDPR compliance requires a dedicated, verifiable *trusted middleware layer*—a secure data-management service interposed between applications and unmodified KVS backends, which centralizes policy enforcement, metadata management, and audit logging while preserving the existing KVS API.

However, designing such a layer raises three intertwined challenges spanning *security and privacy*, *policy enforcement*, and *performance*. First, the middleware must remain secure and trustworthy even when deployed on untrusted clouds, preventing privileged software from observing or altering compliance logic or logs. Administrators or compromised hypervisors may gain access to in-memory data, intercept compliance metadata, or tamper with audit logs to conceal non-compliant behavior [86], undermining both confidentiality and the ability to demonstrate lawful processing. Existing approaches that rely on application-level controls [1] or data-engine enforcement [6, 90] assume a trusted cloud stack and leave the integrity of compliance mechanisms (e.g., metadata) and audit logs unprotected.

Second, the system must translate legal requirements into enforceable policies that can be applied to KV operations without relying on schemas or rich query semantics. This translation is challenging because GDPR requirements are written in legal rather than technical language, leaving system builders to decide how principles such as storage or purpose limitation should be represented and enforced. Prior work on compliance-aware relational databases [6, 90] introduces heavy schema extensions and relies on trusted execution environments, but these designs do not generalize to lightweight, high-throughput KVS workloads or to untrusted cloud deployments.

Third, the middleware must preserve high performance while continuously enforcing compliance and maintaining observability. Integrating compliance logic in the data path risks undermining efficiency that makes KVS attractive. Associating metadata with every record increases storage footprint, while enforcing deletion and access rights often requires non-key lookups that degenerate into full scans. Prior analyses show that naïve GDPR implementations can lead to high compliance-query latencies even for small datasets [90]. Moreover, generating tamper-evident audit logs introduces I/O and synchronization overheads that accumulate in high-throughput scenarios, exhausting resources and inflating latency.

Our key insight is that GDPR compliance can be realized as a *verifiable data-management layer* that executes inside a Confidential Virtual Machine (CVM). Running the enforcement logic within this hardware-isolated environment allows the system to mediate all KV operations securely, associate each key with compact compliance metadata, and maintain cryptographically protected audit logs, while requiring no modifications to existing storage engines.

This paper presents GDPRuler, a trusted middleware that enables GDPR-compliant KVS on the untrusted cloud. More precisely, it deploys a *trusted compliance monitor* inside a CVM, providing hardware-backed isolation, integrity, and attestation guarantees for compliance logic and metadata. The monitor intercepts all KV operations, enforces GDPR policies, maintains per-record metadata, and produces verifiable audit trails. To express compliance rules, GDPRuler introduces a declarative policy language that compiles legal requirements (e.g., purpose limitation) into lightweight runtime checks within the CVM. Compact metadata encoding and dedicated indexes accelerate compliance queries, while a tamper-evident logging system records compliance-relevant operations in a secure, space-efficient format. By acting as a transparent proxy, GDPRuler preserves compatibility with existing deployments, allowing organizations to retrofit GDPR compliance without code changes.

We implement GDPRuler as a transparent proxy layer compatible with unmodified Redis and RocksDB deployments. Our evaluation using YCSB and GDPR-inspired workloads demonstrates that GDPRuler enforces key GDPR obligations while maintaining high throughput and modest storage overheads: GDPRuler achieves ≈61% of native throughput on average, with the underlying CVM contributing 28-32% of this overhead. GDPR queries achieve 13-182× improvement through dedicated metadata indexing, while metadata storage overhead remains below 20%, and the tamper-evident logging system reduces throughput by merely 2%. These results confirm that GDPRuler delivers secure, policy-compliant, and auditable data management on untrusted clouds without sacrificing performance.

In summary, the main contributions of this paper are as follows:

- **Trusted middleware architecture:** a data-management layer that decouples compliance enforcement from applications and KV engines, providing compatibility and verifiable GDPR control.
- **Secure enforcement environment:** a CVM-based *trusted compliance monitor* that ensures confidentiality, integrity, and remote attestation on untrusted cloud infrastructures.
- **Declarative policy framework:** a policy language and compiler translating GDPR rules to enforceable constraints in the data path.
- **Efficient metadata and auditing:** compact per-record metadata encoding, compliance-specific indexing, and tamper-evident logging optimized for high-throughput workloads.
- **Implementation and evaluation:** a prototype supporting Redis and RocksDB demonstrating secure, low-overhead GDPR enforcement under realistic workloads.

2 Background

2.1 Confidential Virtual Machines (CVMs)

Confidential computing [26] protects data in use through hardware-assisted Trusted Execution Environments (TEEs), while providing remote attestation capabilities to ensure the system is deployed with the intended configuration in untrusted cloud environments.

Recently, the industry has shifted toward VM-level confidential computing solutions, namely *Confidential Virtual Machines (CVMs)* [24, 47, 73], including AMD SEV-SNP [8], Intel TDX [50], and ARM CCA [13]. Major cloud providers have started offering CVMs as a service [45, 72]. CVMs run the entire guest system within the TEE, enabling the reuse of existing applications while preventing unauthorized access even from the cloud provider.

2.2 GDPR Policy Compliance for KVS

KVS provides a simple data model and high performance, making it the backbone of latency-sensitive, large-scale services. At the same time, organizations are increasingly subject to strict regulatory frameworks that govern how personal data must be handled, including the General Data Protection Regulation (GDPR) in Europe, the California Consumer Privacy Act (CCPA) in the United States, and other emerging data protection acts. Among these, GDPR has become the global reference, imposing strict responsibilities and hefty penalties for non-compliance. Since many KVS-based applications manage personal data directly, ensuring GDPR compliance is both a

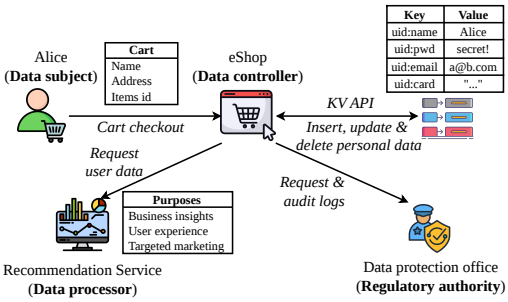


Figure 1: GDPR ecosystem: *The eShop (controller) stores user data in a KVS and uses a recommendation service (processor). Purpose limitation (Art. 5(1)(b)) restricts Alice's data usage to the declared purpose. The controller must provide verifiable compliance and respond to regulatory requests.*

legal and technical necessity. This paper focuses on providing system support for *GDPR-compliant KVS applications*, while preserving the efficiency and compatibility that make KVS attractive in practice.

GDPR entities in KVS applications. GDPR defines distinct actors with specific obligations that naturally map to KVS workflows handling sensitive data (Figure 1). The *data subject* is the individual whose personal data is stored in the KVS. The *data controller* is the organization operating the application and bears the legal responsibility to ensure GDPR compliance. The *data processors* are internal or external services that perform operations on the KVS on behalf of the controller and must declare their intended purpose of data use. Finally, *regulatory authorities* supervise compliance and may request verifiable logs to audit how personal data is processed.

Compliance requirements for KVS. To achieve GDPR compliance all KV operations (e.g., *get*/*put*/*delete*) must respect the policies defined by data subjects, that processors' declared purposes are checked against those policies, and that controllers maintain tamper-evident records of processing activities that regulators can independently audit. Compliance is therefore not an external feature bolted onto the application, but an end-to-end property that must extend across the entire workflow. Meeting this goal demands three key requirements: *(i)* confidential and integrity-protected personal data processing and storage; *(ii)* policy-aware KV operations with verifiable audit logging; and *(iii)* preservation of the performance and simplicity of KVS.

3 Data Security & GDPR Compliance Challenges

KVSs are architected for simplicity and performance, fundamentally complicating the verifiable enforcement of GDPR requirements [90]. Meeting these requirements (§ 2.2) in untrusted cloud environments introduces data management challenges across three dimensions: *security*, *policy specification and enforcement*, and *performance*.

3.1 Security in Untrusted Cloud Environments

KVS-based applications in untrusted clouds face GDPR risks as privileged operators can access or tamper with in-memory KV pairs, stored data, and operation logs. Conventional protections (e.g., authentication, access-control, TLS), while effective, assume a trusted OS and hypervisor, assumptions that no longer hold in public clouds. Therefore, achieving GDPR compliance requires end-to-end protection covering volatile (e.g., in-memory data, policy decisions, compliance metadata) and persistent state (e.g., KV pairs and audit logs).

CVMs establish a strong security foundation by offering three core guarantees: confidentiality of volatile state, ensuring data and metadata privacy; integrity protection, which prevents unauthorized modification of data and policies during execution; and remote attestation, allowing for external verification of the deployment. However, placing a KVS in a CVM alone is insufficient for GDPR compliance because the latter depends on the integrity of policy enforcement and metadata management throughout the workflow. Simply encrypting values does not ensure that deletion requests are honored, retention and purpose policies are correctly enforced, or audit logs remain authentic once data leaves the trusted boundary.

The core security challenge is thus to extend CVM guarantees beyond volatile state to persistent storage and logs, so that policy decisions and compliance records can be verified end-to-end. These guarantees must remain general and compatible across heterogeneous KVS without altering their data models. This requires orchestrating KV operations and dataflows so that data, metadata, and policy enforcement execute in a unified, verifiable environment.

3.2 Policy Specification and Enforcement

GDPR defines a broad legal framework for data protection, but not all of its articles are equally relevant to the KVS layer [90]. We focus on requirements technically enforceable within the KVS, such as "purpose limitation" and "storage limitation" (Article 5), "right of access" (Article 15), "right to be forgotten" (Article 17), "right to object" (Article 21), and "records of processing activities" (Article 30). Articles handled at higher application layers—such as breach notification (Articles 33-34) and data portability (Article 20)—are data controller responsibilities that extend beyond the KVS scope.

GDPR obligations relevant to KVS introduce *policy specification challenges*, as GDPR's legal language cannot be easily mapped to technical constraints. Unlike relational databases with structured schemas and rich metadata support, KVS offers limited expressiveness for defining complex policies. For example, determining if user behavior tracking for "improving user experience" resembles the same purpose (Article 5) as "product optimization" requires legal interpretation that KVS systems cannot handle autonomously. This gap between legal requirements and technical implementation creates uncertainty on whether systems truly satisfy the GDPR rules, exposing organizations to compliance risks despite their efforts.

Beyond specification, *policy enforcement introduces its own challenges*. GDPR mandates associating user data with rich compliance metadata that govern how it can be accessed or modified. In practice, ensuring correct and efficient enforcement across KVS operations is non-trivial. As KVS lacks structured schemas or relational integrity, maintaining key-policy associations through updates or cache operations can introduce inconsistencies. Moreover, the granularity of enforcement, whether at the level of individual keys or namespaces, can cause *metadata proliferation*, demanding careful trade-offs between scalability, consistency, and resource efficiency. These factors collectively make it challenging to realize correct and efficient system-level enforcement of regulatory policies with modern KVS.

3.3 Preserving Performance

GDPR mandates rights (e.g., data retention) that force KVS to locate and manage records via GDPR metadata, not simple key lookups.

These metadata-driven queries (e.g., implementing the "right to be forgotten") are inefficient for existing KVS, causing severe latency and I/O overheads. For example, naïve implementations in Redis show latencies of several seconds even on modest datasets [90].

Beyond query processing, compliance mechanisms themselves introduce performance penalties: GDPR's audit logging demands creating metadata-rich entries for every operation on sensitive data, transforming lightweight requests into heavier I/O workloads with multiple physical writes. Structured GDPR audit logs also require more complex serialization and parsing than simple append-only logs, adding CPU overhead that compounds the I/O bottleneck and erodes the latency advantages of KVS architectures.

Addressing these *performance challenges* requires rethinking how GDPR mechanisms integrate with the KVS data path. A practical approach involves representing compliance metadata compactly, indexing it to avoid full KVS scans, and recording audit trails efficiently without disrupting normal KV operations. Equally important is minimizing interference with the critical data path: reads and writes must remain lightweight, while policy enforcement should be optimized through efficient validation pipelines.

Problem statement. To address these challenges, we target the following question: *How can we design a system that enables secure GDPR compliance for KVS on untrusted clouds while maintaining high-performance query processing, providing verifiable policy enforcement, and preserving compatibility with existing deployments?*

4 Overview

As a solution, we present GDPRuler, a trusted middleware system that acts as a GDPR monitor to transparently ensure GDPR compliance for cloud KVS through strict policy enforcement, efficient GDPR metadata management, and verifiable audit logs.

4.1 System Architecture

Figure 2 depicts GDPRuler's architecture, highlighting its trusted boundary (green box) and key components. Interposed between applications and unmodified KVS engines, it runs within a CVM that establishes an isolated, attested environment mediating all KV operations. Within this boundary, it enforces GDPR policies, validates query intents, and maintains tamper-evident compliance logs, while preserving compatibility with existing KVS. GDPRuler's design is guided by three principles tailored to KVS traits:

- **Policy-driven query control.** Since KVS lack schemas or query planners for access controls, GDPRuler introduces a *policy specification language* and *companion API* for data subjects and processors to express policies alongside KV queries. The subjects define how personal data must be managed (e.g., consent), while processors declare their intended purposes. These policies are compiled into compact metadata structures attached to queries, transforming opaque KVS requests into policy-aware operations.
- **Trusted enforcement and verifiable auditing.** GDPRuler enforces compliance within a *trusted monitor* running in a CVM. All compliance decisions remain confined to this protected domain. A *KVS-tailored tamper-evident log* records policy-relevant actions, while *remote-attestation protocols* extend trust to external GDPR entities allowing them to verify that GDPR enforcement and auditing originate from an authentic, untampered deployment.

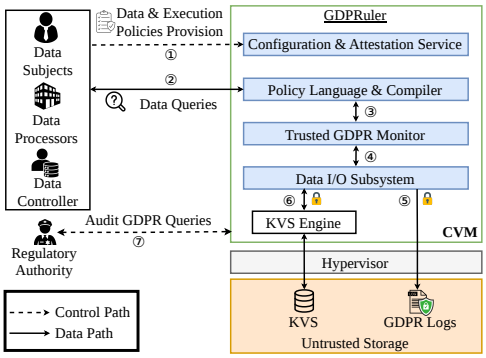


Figure 2: GDPRuler architecture and workflow.

- **Backend-agnostic deployment and extensibility.** GDPRuler operates as a drop-in middleware that can sit atop any KVS. A *unified interface layer* abstracts backend-specific APIs, enabling integration with KVS (e.g., Redis) without re-implementing compliance logic. The design supports future backends through lightweight adapters and performance-oriented optimizations, such as efficient metadata encodings, and metadata indexing (see §5).

Guided by these principles, GDPRuler comprises five modular components. The *Configuration and Attestation Service* anchors system trust by managing entity registration and establishing secure channels via mutual remote attestation, ensuring that only legitimate parties interact with the system and register their default data and execution policies used for subsequent enforcement.

The *Policy Compiler* translates these policies to executable rule sets optimized for KVS workloads, intercepting KV operations and transforming them to GDPR-aware queries carrying compliance metadata. It applies compact encodings and merges default policies with per-query overrides for fine-grained runtime enforcement.

The *Trusted GDPR Monitor* enforces the compiled rules in the CVM, validating access, modification, and retention constraints while managing the tamper-evident log. For data operations, it performs access control and filters results to ensure that the requesting entity's declared policy aligns with subjects' consents, sharing permissions, and retention settings. It maintains auxiliary data structures to optimize GDPR queries and runs a background expiration scanner.

The *Data I/O Subsystem* bridges the monitor and the backend engine, encrypting payloads, authenticating operations, and ensuring the confidentiality and integrity of both data and logs. It generates tamper-evident records for all policy-relevant actions and exposes an authenticated audit trail accessible to regulators.

Finally, the *KVS Engine* executes standard key–value operations over encrypted values without GDPR awareness, ensuring compatibility with unmodified backend software. Both encrypted data files and audit logs reside in untrusted storage outside the CVM.

4.2 System Workflow

1. Setup phase. Before launch, the data controller registers public keys of all legitimate entities—data subjects, processors, controllers, and regulatory authorities—with the *Configuration and Attestation Service*, establishing the system's trust anchors. Entities verify GDPRuler's integrity through remote attestation, while the service validates their cryptographic identities against the registered

Query API	Description
get(k)	Retrieves the *value* associated with *k*.
put(k,v)	Inserts a KV pair and its metadata.
delete(k)	Deletes the specified KV pair.
getm(key_prefix, "(meta)data")	Retrieves *values* or metadata. Results are filtered based on the query policy.
putm(key_prefix, new_metadata)	Updates GDPR metadata for KV pairs that comply with the query policy.
deletem(key_prefix)	Deletes KV pairs that comply with the query policy.
getLogs(optional_key)	Retrieves all GDPR logs (or those of a specified key).

Table 1: GDPRuler's Query API.

Relational predicates (for GDPRuler policy enforcement)	
eq(x,y), *le*(x,y), *lt*(x,y)	$x = y, x \leq y, x < y$
ge(x,y), *gt*(x,y)	$x \geq y, x > y$
Session predicates	
sessionKey(*K*)	Sets and/or represents the session key K
GDPR metadata predicates	
objOwn(*x*), *objOwnIs*(*x*)	Represents the subject (owner) of an object
objExp(*x*), *objExpIs*(*x*)	Represents the expiry time of an object
objPur(*x*), *objPurIs*(*x*)	Represents the purposes for an object or an entity
objOrig(*x*), *objOrigIs*(*x*)	Represents the origin of an object
objShare(*x*), *objShareIs*(*x*)	Represents the sharing information of an object
objObj(*x*), *objObjIs*(*x*)	Represents the objections imposed on an object
GDPR data predicates	
monitor($true/false$)	Enables/disables monitoring
encryption($true/false$)	Enables/disables encryption
Query predicates	
query($put/get/delete$)	Represents a standard query operation
query($putm/getm/deletem$)	Represents a GDPR query operation
query($getLogs$)	Represents a GDPR getlog query operation

Table 2: GDPRuler policy language predicates.

keys. After successful mutual attestation, secure channels are established, and session keys are issued for subsequent interactions. Data subjects and processors then upload default data and execution policies to the *Trusted GDPR Monitor*, securing them inside the CVM (①).

2. Query execution. When an entity issues a query (②), it may include GDPR predicates that act as complementary constraints. The *Policy Compiler* parses these predicates and merges them with the entity's default policy, generates an enforcement rule, and attaches the corresponding metadata to the query. The augmented operation is passed to the *Trusted GDPR Monitor* (③), which validates it against the compiled rules, and forwards it to the *Data I/O Subsystem* (④).

3. Data processing and logging. The *Data I/O Subsystem* creates a tamper-evident log entry for each operation involving personal data or in case of a policy violation (⑤). Then, it encrypts the payload (data, metadata), maps the operation to the appropriate API (e.g., Redis *SET/GET*, RocksDB *Put/Get*), and submits it to the *KVS* (⑥) for execution. Subsequently, it returns the result to the monitor, which performs final validation before delivering it to the requesting entity.

4. Auditing. For regulatory inspection, authorities can request access to GDPR logs (⑦), enabling verification of compliance and detection of possible violations. The *Trusted GDPR Monitor* authenticates the requester based on its registered key.

4.3 Programming Model

GDPRuler exposes a standard *put/get/delete* API, summarized in Table 1. To support diverse backends that may use distinct bindings, GDPRuler provides adapters that map these operations to its unified API. To support metadata management and auditing, required for GDPR compliance (§3.2), GDPRuler provides *compliance-aware operations* (*getm*/*putm*/*deletem*) and a logging primitive (*getLogs*), enabling the management of GDPR-related metadata and audit logs.

Before issuing queries, each GDPR entity specifies a default data handling policy, as shown in the listing below.

```
// Data subject (Alice) default data policy:
{"sessionKey":alice_key, "default_policy": {
  "purpose":[recommendations,orders], "share":[recommender_key],
  "objection":[marketing,analytics], "expTime":[90d],
  "origin":[shop.com/account_creation], "monitor":[true]}}
// Data processor (Recommendation Service) default execution policy.
{"sessionKey":recommender_key,
 "default_policy": {"purpose":[recommendations]}}
```

Entities express queries with GDPRuler's policy specification language (§5.1) using predicates shown in Table 2. If no per-query predicates are specified, the system automatically applies the entity's default policy. The following listing presents query examples.

```
// Data subject: store data using the default policy.
query(put("alice:preferences","data")) ∧ sessionKey(alice_key)
// Data processor: request data for the "recommendations" purpose.
query(get("alice:preferences")) ∧ sessionKey(recommender_key)
  ∧ objPurIs(recommendations)
```

4.4 Deployment and Threat Model

GDPRuler assumes a platform with CVM support to host its trusted components, such as AMD SEV-SNP [8], Intel TDX [50], and ARM CCA [13]. External applications operated by data subjects, processors, controllers, and regulators interact with GDPRuler over mutually attested, encrypted channels established during system setup.

Applications outside GDPRuler are assumed to run within trusted domains (e.g., CVM, on-premises environment). Securing application logic is beyond our scope. The system uses persistent storage for KVS data and compliance logs. We assume this storage provides durability but not rollback protection. Deployments may instantiate a single CVM instance or multiple ones depending on scalability needs.

Under this deployment setting, the trusted computing base (TCB) of GDPRuler comprises the CVM hardware, its firmware, and all software components running inside the CVM. All other system software and infrastructure—including the hypervisor and external storage—is untrusted. GDPRuler inherits the standard adversary model of modern CVMs [4, 8, 10, 23], defending against powerful attackers [33] that can compromise the system software stack or perform physical attacks such as memory probing, while assuming the processor package and on-chip state remain protected.

We do not consider side-channel attacks [5, 9, 29, 48, 49, 104, 105], denial-of-service, or timing manipulation (e.g., clock/frequency tampering [32]). For GDPR logs, we consider adversaries capable of modifying the data or launching rollback and forking attacks [18, 66, 80].

5 Design

5.1 Policy Language and Compiler

The Policy Language and Compiler transforms data policies into GDPR-aware operations. We first describe how policy is specified with our novel declarative language, along with associated metadata, and then detail its compilation process for enforcement.

GDPR metadata. In GDPRuler, each KV pair (object) has associated metadata that keeps the policies necessary for GDPR compliance. This metadata includes its *subject* (*session key*), *origin* (data source), *purpose* (allowed uses), *objections* (prohibited uses), *sharing permissions* (which processors may access the data), *expiration date*, an *encryption* flag, and a *monitor* flag (logging on/off). This metadata

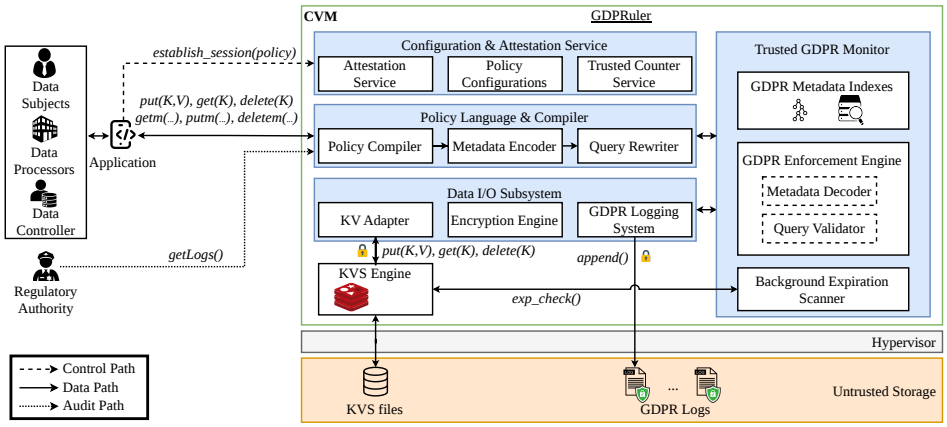


Figure 3: Detailed view of GDPRuler system. GDPRuler consists of *(i)* a policy compiler for GDPRuler's declarative policy language, *(ii)* the Trusted GDPR Monitor, *(iii)* GDPRuler's Data I/O Subsystem, and *(iv)* Configuration and Attestation Services.

is created when a value is inserted and is either provided along with the insertion query or initialized with default policies. Internally, the metadata is prepended to the associated value of a KV pair and stored in the KVS. Besides GDPR metadata associated with KV pairs, GDPRuler preserves metadata for data processors (i.e., session key, data processing purposes), initially provided during the setup phase.

Declarative policy language. Drawing inspiration from declarative language for data policy specification (e.g., [58, 101, 102]), GDPRuler defines a policy language that enables data subjects and processors to specify their data-handling policies along with KV queries in a formal predicate-based syntax (Table 2). These policies use session predicates to specify their session, GDPR metadata and GDPR data predicates to specify GDPR policies, and the query predicate to specify the actual query, with multiple predicates joined with '∧'. For instance, a typical query is formed like the following:

```
query(op) ∧ sessionKey(K) ∧ objExp(x) ∧ objPur(y) ∧ ...
```

GDPRuler policies are categorized into two parts: *data access* and *execution* policies. Data access policies override the data subject's default policy. Specifically, preference type predicates (predicates of the form *objXXX()*, e.g., *objPur(x)*) explicitly specify the policy associated with the KV pair. For example, Alice (data subject) uses

```
query(put("alice:purchase","data")) ∧ sessionKey(alice_key) ∧
   objPur(orders) ∧ objObj(recommendations)
```

to insert "data" with a key "alice:purchase" using "alice_key" as a session key with the purpose of "orders" and objection to "recommendations". Additionally, filter type predicates (predicates of the form *objXXXIs()*) are used to create a filter for the associated metadata (referred to as the *metadata filter*) when performing GDPR queries (*getm/putm/deletem*). For example, the following operations get values whose key prefix is "alice:" and whose purpose is "orders".

```
query(getm("alice:","data"))∧sessionKey(alice_key)∧objPurIs(orders)
```

On the other hand, execution policies control the intent of data processors. With this policy, for instance, the data recommendation service (data processor) may send the following query intended to get a value of "alice: purchase" for recommendation purposes.

```
query(get("alice:purchase")) ∧ sessionKey(recommender_key) ∧
   objPurIs(recommendations)
```

GDPRuler policy enforcement rules. The Trusted GDPR Monitor uses an extended version of our policy language to describe its policy enforcement rules. They are expressed in the form *perm :- condition*. The *condition* is composed of relational predicates (e.g., *eq(x,y)*) and predicates used by data access and execution policies. Multiple relational predicates are joined with '∧' (logical and) or '∨' (logical or) operators. The filter type predicates (predicates of the form *objXXXIs()*) are extended to return metadata values associated with a session key or object (e.g., *objPurIs(K)* returns the purpose of the entity K). The language also defines '&' (intersection) and '||' (union) operations for metadata values. Using this policy language, GDPRuler clearly specifies the GDPR enforcement rules in a declarative manner. For instance, the following enforces the purpose limitation (Article 5):

```
// objPurIs(K): the purposes of the entity with session key K
// objPurIs(k): the allowed purposes of the data object with key k
read :- eq(objPurIs(K) & objPurIs(k), objPurIs(K))
```

The current notation overloads predicates with boolean and metadata semantics; this can be disambiguated by separating them into distinct predicates (i.e., objPurIs(k)/entityPurIs(K)) without affecting enforcement. We detail how predicates form GDPR rules in §5.6.

Policy compiler. The policy compiler transforms high-level GDPR policy expressions into executable enforcement directives. For each query operation, it parses the policy predicates from the request, tokenizes these expressions, and performs semantic analysis to extract predicate attributes and values. It then combines those attributes with default policies to fill the missing values. For GDPR queries, it generates *metadata filters* for later filtering operations.

Metadata encoder. For *put/putm* operations, metadata is stored along with the associated values. GDPRuler utilizes bitmaps for values that represent categories (e.g., purposes), enabling quick bitwise operations for policy validation. The mapping of bits is established during runtime, and all bitmap structures can be resized to accommodate new entries. This ensures the system remains adaptable to changing compliance requirements while minimizing metadata and maintaining efficient processing. The expiration times have 8-byte precision to ensure precise time-based policy enforcement.

Query rewriter. Given the policies from the policy compiler, the query rewriter transforms standard KV queries into GDPR-aware operations. For write operations (e.g., *put*), it prepends the encoded GDPR metadata to the user data. For read operations (e.g., *get*), it propagates the query along with the associated metadata filters to the GDPR Enforcement Engine for subsequent validation.

5.2 Trusted GDPR Monitor

The Trusted GDPR Monitor serves as the central compliance verification component of GDPRuler. It processes GDPR-aware operations and ensures all access complies with specified policies.

GDPR enforcement engine. The GDPR enforcement engine receives operations with GDPR metadata from the query rewriter. For *get/getm* operations, it also retrieves (encoded) GDPR metadata of the associated object, and the *Metadata Decoder* decodes it. Subsequently, its *query validator* validates the policy enforcement rules. After successful validation, it forwards the operations to the Data I/O Subsystem to materialize the KV operation and logging.

The query validator first confirms that the data is not expired and validates that the request initiator either is the data subject (for updates) or has explicitly been granted access via the sharing property (for reads). Then, it checks that declared purposes fall within the subset of allowed purposes, and finally verifies that no query purposes are part of the KV pair's objections.

GDPR metadata indexes. To optimize policy validation, GDPRuler maintains specialized in-memory data structures that index keys based on their GDPR metadata, as shown in Figure 4, avoiding full KVS scans for metadata operations. GDPRuler provides different indexing options (e.g., hashmap, B+ tree, inverted index). Data controllers can enable specific data structures based on anticipated GDPR query patterns and metadata types. For instance, hashmap indexes provide O(1) lookups for exact metadata matches, while B+ trees support efficient range queries, useful for temporal fields.

Background expiration scanner. GDPRuler implements a background expiration scanner to enforce GDPR's data minimization and storage limitation principles (Article 5). The scanner operates asynchronously at configured intervals. It scans metadata and queues delete requests on expired data to the Data I/O Subsystem. While the KVS backend handles actual deletion, which may implement techniques such as tombstoning [34], GDPRuler ensures that expired data becomes unreachable through its API via its GDPR Enforcement Engine. This approach represents a reasonable trade-off for GDPR compliance, while respecting KVS-specific deletion mechanisms.

5.3 Data I/O Subsystem

GDPRuler's GDPR Data I/O Subsystem handles all data encryption, KV backend communication, and tamper-evident audit logging.

KV adapter. The KV adapter bridges GDPRuler's components with the *unmodified* KVS engine. It implements a template-based approach where core operations (e.g., *put*, *get*, *delete*) are mapped to the specific API of different KV backends. This design enables the seamless integration of diverse KV stores by utilizing generic data types (e.g., byte streams) that are compatible with most implementations.

Encryption engine. To ensure confidentiality (Article 32), even if a KVS does not inherently support encryption, GDPRuler offers a built-in encryption engine. During connection establishment, data subjects and regulatory authorities perform authenticated key exchange with GDPRuler using their session keys to derive encryption keys. The encryption engine encrypts user data (values) and log entries and produces a Message Authentication Code (MAC) as part of each object's metadata to validate integrity upon decryption.

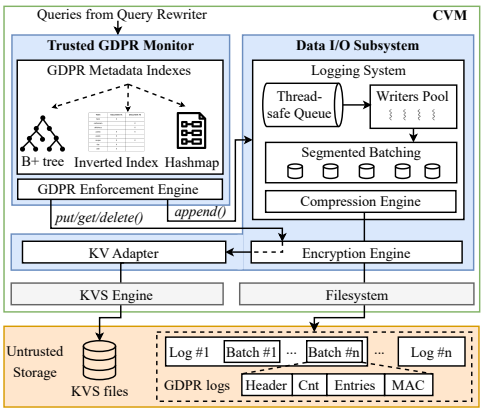


Figure 4: Metadata indexing and data I/O subsystems.

GDPR logging system. To address the GDPR auditing requirements (Article 30/33), GDPRuler incorporates a secure, tamper-evident logging system (Figure 4) that ensures the confidentiality, integrity, and freshness properties of GDPR logs. Log entries are batched and compressed to optimize write throughput and reduce write amplification. Specifically, they are enqueued in a thread-safe queue, and processed by dedicated writer threads. Processed entries are grouped as a batch based on a hash of the affected key. The batch contains a metadata header, a monotonically incremented software-based trusted counter protected within the CVM for freshness verification, a series of GDPR entries encrypted along with the counter, and a MAC for integrity checks. The batched log is written to append-only files, and log files are automatically rotated based on configurable size thresholds.

For log access requests from regulatory authorities, GDPRuler decrypts the identified files per-batch, verifies integrity (via MAC), and confirms monotonically increasing counter values within the expected range, thereby preventing replay attacks.

5.4 Configuration and Attestation Service

The Configuration and Attestation Service establishes trust with GDPRuler, manages policies, and maintains session security.

Attestation service. GDPRuler's attestation service enables external entities to ensure the integrity of a GDPRuler and KV engine before establishing secure communication channels. They verify the platform configuration, guest VM measurements, and GDPRuler binaries through hardware-generated attestation reports [10]. Upon successful verification, GDPRuler validates applications against entity keys registered by the data controller, ensuring that only authorized, legitimate applications can establish connections with it.

Policy configurations. The policy configuration component manages default GDPR policies, which are provided by each entity after successful mutual attestation. The default policies are associated with the entity's session key and stored within the CVM memory.

Trusted counter service. The trusted counter service provides monotonically increasing counters to ensure the freshness of the audit logs. It maintains a software-based counter for each log file that increments with each appended batch. During log retrieval, GDPRuler verifies that the log's counter value matches the expected sequence. Any deviation from the expected values indicates tampering or rollback attacks. The counters are periodically checkpointed to encrypted storage to survive system restarts.

Algorithm 1: Unified GDPR-Aware Operation Framework

Input : $operation$,key/key_prefix,$value$,$session_key$,$query_policy$
Output : Compliance-validated result or denial of operation

Stage 1: Policy Compilation
$default_policy \leftarrow$ `get_session_policy`($session_key$);
$policy \leftarrow$ `compile_query_policy`($default_policy$, $query_policy$);
Stage 2: Data Retrieval and Metadata Extraction
$encrypted_value(s) \leftarrow$ `kv_store.get`(key or $key_prefix*$);
$value(s) \leftarrow$ `decrypt`($encrypted_value(s)$);
$gdpr_metadata \leftarrow$ `extract_metadata`($value(s)$);
Stage 3: Access Validation and Compliance Checking
foreach $entry \in gdpr_metadata$ **do**
  $compliance \leftarrow$ `validate`($entry$, $policy$, $session_key$);
  if $\neg compliance$ **then**
    `monitor_query`(key, $operation$, $compliance$);
    Skip or deny this $entry$;
  end
end
Stage 4: Operation Execution and Audit Logging For Each Value
if $operation \in \{put, putm\}$ **then**
  $enhanced_value \leftarrow$ `rewriter.create_new_value`($policy$, $value$);
  if $gdpr_metadata.encrypt$ **then**
    $enhanced_value$ = `encrypt`($enhanced_value$);
  end
  $result \leftarrow$ `kv_store.put`(key, $enhanced_value$);
else if $operation \in \{get, getm\}$ **then**
  $result \leftarrow$ `filter_and_aggregate`($compliant_entries$);
else if $operation \in \{delete, deletem\}$ **then**
  $result \leftarrow$ `kv_store.delete`($compliant_keys$);
if $gdpr_metadata.monitor \vee violationsdetected$ **then**
  `monitor_query`(key, $operation$, $result$, $compliance$);
end
return $result$;

5.5 Algorithms

Algorithm 1 illustrates how GDPRuler converts standard KV operations to GDPR-compliant operations via a four-stage pipeline: *(i)* policy compilation, and merging default and query-specific predicates, *(ii)* object metadata retrieval, *(iii)* policy validation for compliance checks, and *(iv)* operation execution with audit logging. Detailed algorithms for all operations are covered in our tech report [96].

Standard KV queries. For *get* operations, GDPRuler first compiles policies from the requesting entity and any operation-specific overrides (Stage 1), retrieves and decrypts the KV pair and extracts embedded GDPR metadata (Stage 2), validates access permissions (Stage 3), and logs the access if required by the data's privacy settings (Stage 4). Failed access attempts are always logged for audit purposes.

put operations execute Stages 1 and 2 of the pipeline to check if a key exists and fetch its current metadata. To reduce KVS stress, GDPRuler preserves an in-memory cache associating a subset of keys with their metadata. For existing keys, Stage 3 validates modification permissions based on their current metadata. In Stage 4, the query rewriter encodes metadata directly within the value, creating a payload that embeds GDPR properties. The operation is then submitted to the KVS and gets conditionally logged.

delete operations follow a similar pipeline to *get/put*. GDPRuler ensures that deletion rights are properly enforced while maintaining comprehensive audit trails.

GDPR queries. For *getm* operations, GDPRuler performs bulk retrieval using key prefix matching (Stage 2), then applies a two-stage filtering: (1) filters entries matching specified criteria, and (2) performs individual GDPR compliance validation. In Stage 4, every data access is logged, if required, creating detailed audit trails.

GDPR Articles	Policy enforcement rule
#5 Purpose limitation	**read**:- eq(objPurIs(K) & objPurIs(k), objPurIs(K))
#5 Storage limitation	**read**:- le(time, objExpIs(k))
#13,#14 Information to be provided	**read_metadata**:- eq(objOwnIs(k), objOwnIs(K))
#15 Right of access by users	**read(_metadata)**:- eq(objOwnIs(K), objOwnIs(k)) **update(_metadata)**:- eq(objOwnIs(K), objOwnIs(k))
#17 Right to be forgotten	**delete**:- eq(objOwnIs(K), objOwnIs(k))
#21 Right to object	**update**:- gt(objPurIs(K) & objPurIs(k),0) **read**:- eq(objPurIs(K) & objObjIs(k),0)
#22 Automated individual decision-making	Derived from policies in #5 and #21
#25 Data protection by design and default	**read**:- eq(objOwnIs(K), objOwnIs(k)) ∨ gt(objOwnIs(K) || objShareIs(k), 0) **update**:- eq(objOwnIs(K), objOwnIs(k)) **delete**:- eq(objOwnIs(K), objOwnIs(k))
#28 Do not grant unlimited access to data	Access control based on session keys and ownership verification
#30 Records of processing activity	**read**:- eq(monitor(k), true) **update**:- eq(monitor(k), true) **delete**:- eq(monitor(k), true)
#31 Cooperation with supervisory authorities	GDPRuler's mutual attestation protocol, and tamper-evident logging scheme
#32 Security of processing	Encryption of data and metadata, and TEE security properties
#33 Personal data breach	**read**:- eq(K, K_{reg}) ∧ query(getLogs())

Table 3: GDPR articles related to KVS [90] and their mapping to GDPRuler's policy enforcement rules. K: the session key of the querying entity, k: the requested data object, K_{reg}: the session key of a regulatory authority.

The *putm* operation executes Stages 1 and 2 to compile policies and perform bulk retrieval. In Stage 3, the system applies GDPR metadata filters to identify entries matching the specified criteria, then validates modification permissions for each entry. Stage 4 leverages the query rewriter to create enhanced values.

The *deletem* operation follows the unified framework adapted for bulk deletion. After policy compilation (Stage 1) and bulk retrieval (Stage 2), Stage 3 validates that the requesting entity has deletion rights for each matching pair strictly based on ownership. In Stage 4, GDPRuler performs bulk deletion of compliant KV pairs and logs each deletion with comprehensive audit information.

Regulatory accesses. The *getLogs* operation requires explicit regulator authentication and follows a specialized retrieval path. The operation validates regulatory authority credentials through their registered keys, temporarily pauses ongoing logging to ensure consistency in the retrieved audit trail, retrieves log files either for specific keys (using hash-based partitioning) or the entirety of the log files, and internally performs integrity and freshness verification and reports any tampering detections to the authorities.

5.6 GDPR Policy Predicates

We next present how GDPRuler expresses GDPR articles [90] in an e-commerce scenario (Figure 1): customer Alice (data subject), RecommendationService and AnalyticsService (processors), and an online retailer (controller). Alice's default policy allows recommendations and orders, objects to marketing and analytics, sets 90-day retention, and shares her data with RecommendationService, which in turn declares its recommendation purpose in its execution policy, as shown in §4.3. Table 3 summarizes GDPRuler's enforcement rules.

Purpose limitation (Article 5). When RecommendationService attempts to read Alice's shopping preferences, GDPRuler validates that the declared purpose "recommendations" is included in Alice's allowed purposes, does not conflict with her objections, and sharing

Attack vector	Mitigation
System level attacks	
Read the GDPRuler's memory	CVM protection
Read user data in the query request	Data encryption
Load forged GDPRuler	Attestation
DMA to the GDPRuler's memory	CVM protection
Manipulate user request	Data encryption
Rollback KV data	Undetected (out of scope)
GDPR-specific attacks	
Access other KV instance's memory	CVM protection
Read the GDPR logs	Log encryption, access control
Tamper with GDPR logs	Detection via integrity checks
Rollback GDPR logs	Detection via freshness checks
Read KV data	Data encryption
Tamper with KV data	Detection via integrity checks
Malicious request for access to KV data	Policy check, access control
Fake user identity	Mutual attestation (optional)

Table 4: Attack vectors and GDPRuler's mitigations.

is permitted. The access succeeds, and since Alice's monitoring flag is enabled, the operation is logged for auditing.

```
query(get("alice:preferences")) ∧ sessionKey(recommender_key)
  ∧ objPurIs(recommendations)
```

In contrast, when AnalyticsService attempts to access the same data with purpose "analytics", GDPRuler detects this in Alice's objections, denies access, and logs the violation attempt.

```
query(get("alice:preferences")) ∧ sessionKey(analytics_key)
  ∧ objPurIs(analytics)
```

Right of access (Article 15). Alice exercises her right to retrieve all her personal data stored in the retailer's system. GDPRuler locates all entries where Alice is the subject (owner), and returns them.

```
query(getm("alice:","data"))∧sessionKey(alice_key)∧objOwnIs(alice)
```

Right to object (Article 21). After receiving targeted recommendations, Alice decides she no longer wants this processing. She exercises her right to object by updating her metadata across all entries. GDPRuler updates the objections bitmap in all affected entries while preserving the underlying data values.

```
query(putm("alice:")) ∧ sessionKey(alice_key) ∧ objOwnIs(alice)
  ∧ objObjections(marketing,analytics,recommendations)
```

Right to be forgotten (Article 17). Alice requests complete deletion of her personal data. GDPRuler validates her ownership, locates all her entries, executes the deletion, and creates a comprehensive audit log documenting the deletion operation.

```
query(deletem("alice:")) ∧ sessionKey(alice_key) ∧ objOwnIs(alice)
```

Records of processing activities (Article 30). The DataProtectionAuthority audits processing activities for Alice's preferences. GDPRuler authenticates the authority, retrieves the logs, performs integrity and freshness checks, and returns a formatted audit trail.

```
query(getLogs("alice:preferences")) ∧ sessionKey(regulator_key)
```

6 Security Analysis

Our security analysis comprehensively details potential attack vectors and their mitigations (Table 4), including both system-level and GDPR-specific threats. Further, we formally verify GDPRuler's attestation and logging protocols using the Tamarin Prover [30, 68] under the Dolev-Yao [33] attacker model, detailed in our tech report [96].

For system-level attacks, GDPRuler leverages CVM protection to prevent unauthorized memory access, including DMA. Data encryption and the respective integrity checks protect query data and prevent data manipulation during transit. The attestation mechanism detects forged GDPRuler instances, ensuring only legitimate deployments can process sensitive data. Note that GDPRuler does not protect against rollback attacks on database data, as such attacks require an intrusive design in the format in which databases store their data (e.g., versioning), which contradicts the compatibility goal of GDPRuler. To prevent time-manipulation attacks, GDPRuler uses secure time sources provided by CVMs, e.g., AMD's secureTSC [64].

For GDPR-specific attacks, CVM-provided isolation prevents access to other KV instances' memory, while log encryption and access control protect GDPR logs from unauthorized access. Integrity checks detect data tampering, and freshness checks prevent rollback attacks on logs. Lastly, GDPR policy checks and access control mechanisms prevent malicious requests for unauthorized database access, and mutual attestation prevents fake user identity attacks.

Variant	GDPR	Encryption	CVM
Native KVS **(deployed w/o VM)**	✗	✗	✗
Native GDPRuler (w/o Encr)	✓	✗	✗
Native GDPRuler	✓	✓	✗
CVM KVS	✗	✗	✓
GDPRuler (w/o Encr)	✓	✗	✓
GDPRuler	✓	✓	✓

Table 5: Benchmarking variants.

	Workload	Access pattern	Request distribution
YCSB [19]	A	50% Read, 50% Update	Zipfian
	B	95% Read, 5% Update	Zipfian
	C	100% Read	Zipfian
	D	95% Read, 5% Insert	Latest
	F	100% Read-Modify-Write	Zipfian
GDPR [90]	Controller	50% Insert, 25% Update metadata, 25% Delete	Uniform
	Customer	20% Read, 20% Read metadata, 20% Update, 20% Update metadata, 20% Delete	Zipfian
	Processor	80% Read, 20% Read by metadata	Uniform

Table 6: Benchmarking workloads and their characteristics.

7 Evaluation

We evaluate GDPRuler by analyzing its GDPR compliance performance (§7.1), storage efficiency (§7.2), and I/O subsystem (§7.3).

Prototype. We implement GDPRuler's policy compiler in `Python` and runtime components in `C++`. The KV adapter supports Redis [85] and RocksDB [71], communicating via Unix domain sockets, and uses `OpenSSL` [78] for cryptographic operations. GDPRuler's logging system features a lossless `zlib` [43, 108] compression layer, which supports various compression levels (0-9), balancing between CPU consumption and compression efficiency [40, 61, 94].

Testbed. We perform our experiments on an AMD SEV-SNP server with an AMD EPYC 7713P CPU (64 cores) and 1024 GB of DRAM. The server runs NixOS 25.05 with an SNP-enabled Linux kernel (v6.16.11). CVMs run Ubuntu 22.04 OS with a Linux kernel version v6.5.0.

Baseline and variants. We conduct experiments with the variants described in Table 5. "GDPR" employs GDPRuler, "Encryption" includes data encryption, and "CVM" deploys the system in CVMs. *Importantly, with Native KVS (deployed outside a VM), we show the worst-case overheads for GDPRuler.*

Workloads. We use YCSB workloads (A-D, & F) [19, 27] to evaluate end-to-end KVS performance (we exclude workload E due to the lack of range queries in our prototype). Each client uses a crafted default policy, controlling encryption and logging percentage.

We also evaluate GDPRuler with GDPR-specific workloads (Table 6) proposed by GDPRBench [90]. These workloads include: *(i) Controller*, representing data controllers inserting KV pairs and performing administrative metadata updates and deletions, *(ii) Customer*,

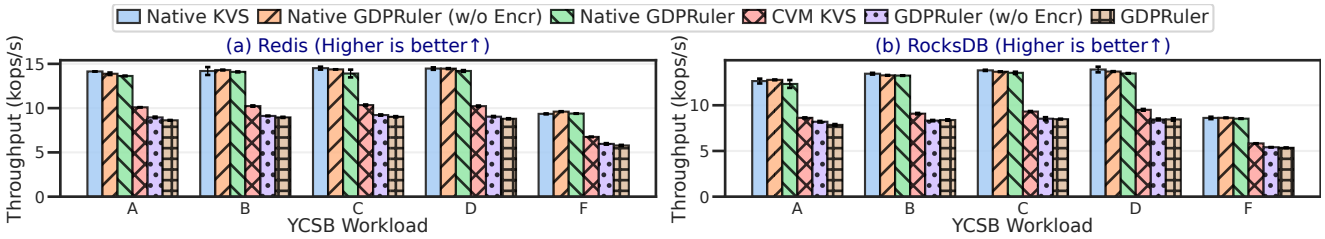


Figure 5: Workload analysis for Redis and RocksDB: GDPRuler's performance with different YCSB workloads.

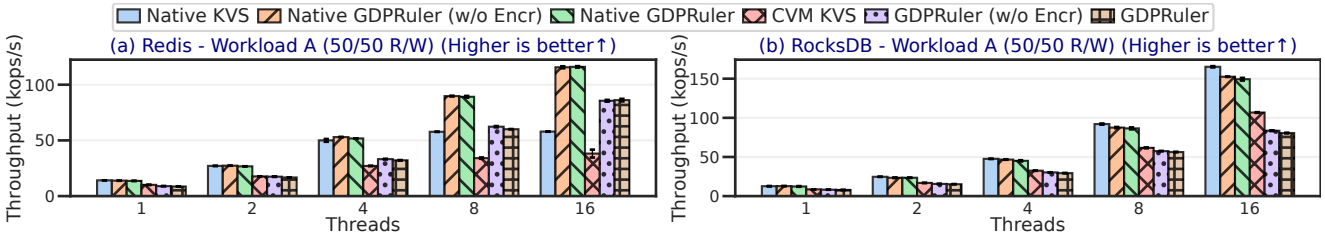


Figure 6: Scalability analysis for Redis and RocksDB: GDPRuler's scalability with YCSB's write-heavy workload A. Other YCSB workloads are covered in our extended tech report [96].

representing data subjects exercising GDPR rights by reading their data, updating privacy preferences, and requesting deletions, and *(iii) Processor*, representing data processors reading data with occasional metadata-based queries.We omit the *Regulator* workload of GDPRBench as it was not publicly available from the authors.

In every run, we preload the KVS per YCSB guidelines [20]. Unless otherwise specified, we report the average across 3 runs.

7.1 Performance under GDPR Compliance

We evaluate GDPRuler across three dimensions: *(i)* KVS workload performance, *(ii)* scalability, and *(iii)* GDPR workloads performance.

KVS workload analysis. For each YCSB workload, we execute 1M queries over 100K distinct keys. We set the logging to be 0% to show the effect of processing the GDPR metadata and performing their filtering in the hot data path. For variants employing GDPRuler, we ensure all queries have valid policies, maintaining consistent data retrieval and insertion across experiments.

Figure 5 presents the end-to-end throughput across all YCSB workloads with a single client (1 thread). In this setup, GDPRuler achieves 61.8% and 61.7% of the Native KVS throughput on average across all workloads for Redis and RocksDB, respectively. The read-heavy Workload C reaches 61.4-62.3% of baseline, while the respective range for the write-heavy Workload A is 61-62.1%. The performance degradation mainly stems from write operations requiring metadata encoding and policy validation before KV insertion, while read operations primarily perform metadata extraction and filtering. We further observe that the deployment of the KVS engine in a CVM introduces 28.5-32.2% overhead compared to native execution, mainly due to memory encryption and CVM I/O costs. Notably, adding GDPRuler's GDPR compliance on top of the CVM introduces only an additional 8.6% overhead, while the encryption (AES-GCM) contributes minimally to this overhead (<2.6%).

Scalability analysis. To evaluate GDPRuler's scalability, we vary the number of concurrent clients. Figure 6 presents the scalability results for Redis and RocksDB for Workload A. Additional results for the rest of the YCSB workloads are provided in our tech report [96].

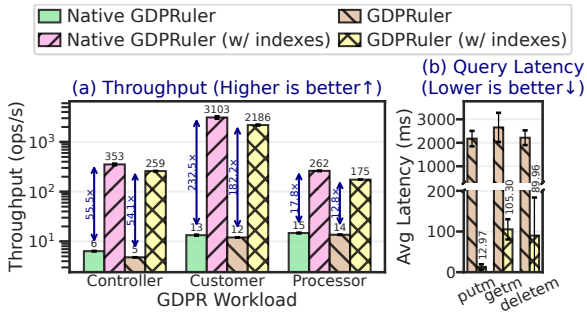


Figure 7: GDPRuler's performance for GDPR workloads and average latency of GDPR queries with Redis backend. RocksDB results are covered in our tech report [96].

As thread count increases from 1 to 16, GDPRuler demonstrates good scalability with the throughput gap between the baseline and GDPRuler remaining relatively constant (≈50%). For instance, GDPRuler with RocksDB achieves 80.6 K ops/sec for Workload A, while the baseline 165.1 K ops/sec, when handling 16 concurrent clients. However, for Redis deployments with more than 8 clients, GDPRuler using Unix sockets achieves comparable or better throughput than native Redis with TCP connections. Specifically, GDPRuler achieves 86.1 K ops/sec for Workload A, while the baseline reaches only 57.9 K ops/sec. Our analysis indicates Redis's *single-threaded* architecture creates a bottleneck at the TCP layer, which GDPRuler's connection handling bypasses. This phenomenon does not occur with our RocksDB server, which properly utilizes multiple threads and shows expected overhead patterns.

GDPR workloads. We evaluate GDPRuler's performance on GDPR-specific workloads (Table 6) that use metadata-based operations (*putm*, *getm*, *deletem*). We preload 100K KV pairs, then execute GDPR workloads consisting of 1K operations with 8 connecting clients and encryption enabled, similarly to prior work [90]. We run GDPRuler with (1) no indexing, and (2) "user" and "purpose" metadata indexing, which are the most frequently queried fields in GDPR workloads. We distribute KV pairs among 128 users and 128 data sources with round-robin assignment of purposes and objections. Each KV pair is

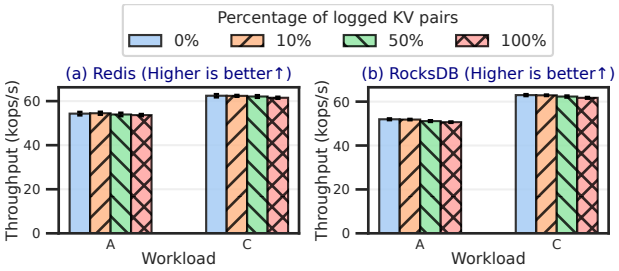


Figure 8: GDPRuler's throughput with varying logged KV pair percentages for Redis and RocksDB.

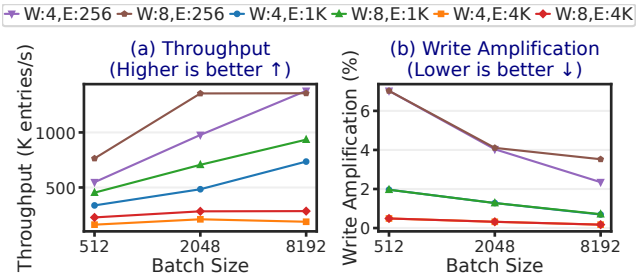


Figure 9: Logging layer performance analysis for various writer threads (W) and entry sizes (E).

configured to be shared with 10 users to reflect realistic multi-user access patterns and reduce query failure rates. We ensure that query operations specify valid purposes, maintaining consistency with realistic GDPR-compliant access patterns. We present the results of Redis here; RocksDB exhibits similar trends and is detailed in our extended tech report [96] due to space constraints.

Figure 7(a) demonstrates the throughput of GDPRuler on Controller, Customer, and Processor workloads for Redis. Without indexes, GDPRuler must perform full KVS scans to locate entries matching metadata criteria, resulting in significant performance drops as low as 4.8, 12.0, and 13.6 ops/sec for the three workloads, respectively. This low throughput shows that even a relatively small subset of GDPR queries—only 1K operations across 100K pairs—can become detrimental without proper indexing support. With GDPR indexes enabled, performance improves by 12.8-182.2×, depending on the workload. Specifically, Customer workloads that access all personal data of a specific subject benefit the most through indexed ownership lookups (182.2×). Similarly, Processor workloads performing purpose-based filtering achieve significantly faster execution (12.8×). These improvements highlight that GDPR metadata indexing effectively eliminates the need for multiple full-KVS scans. Note that, similar to previous experiments, CVM deployment maintains 26.6-42.8% overhead compared to native execution.

Figure 7(b) presents the latency for GDPR operations averaged across the three workloads, justifying the poor performance in the non-indexed implementation. GDPR queries show significant latency reductions with indexing, from 2,178-2,654 ms to just 13-105 ms on average for Redis. Precisely, *putm* operations improve by 168×, while *getm* operations achieve 25× speedup. This massive decrease is reasonable if we consider that, with indexes, operations transform from full KVS scans examining every KV pair to efficient index lookups followed by targeted KV pair fetching and metadata validation. We further notice that even standard operations (*put, get, delete*) benefit from reduced contention, showing 12-45% latency improvements for Redis despite not directly using GDPR indexes.

7.2 Storage Efficiency

We analyze the GDPRuler's storage layer characteristics across four dimensions: *(i)* the *end-to-end impact* of logging on application throughput, *(ii)* the *isolated throughput* of the logging subsystem itself, *(iii)* the *storage overhead* introduced by embedding GDPR metadata within KV records, and *(iv)* the *audit log storage requirements* under varying logging configurations and compression settings.

Logging layer impact on applications. To quantify the impact of GDPR audit logging, we run a write-heavy (Workload A) and a read-heavy (Workload C) workload with 8 client threads with varying logging percentages. For these experiments, we configure GDPRuler with compression level 3 and encryption enabled.

Figure 8 highlights that GDPRuler's asynchronous logging architecture incurs minimal performance overhead, with throughput degradation under 2% even at 100% logging across workloads and KVS. The low overhead stems from GDPRuler's design decisions: asynchronous logging decouples it from the critical path, batched log writes amortize I/O costs across multiple entries, and efficient compression reduces the volume of data written to storage. These results show that GDPR audit logging can be achieved in production systems without sacrificing performance, validating GDPRuler's approach of treating logging as an off-critical-path operation.

Logging layer performance. To further understand GDPRuler's logging system parameters, we evaluate the logging layer using a microbenchmark with 16 producer threads, varying logging threads (4 or 8), batch size (512-8192 entries), and log entry size (256 bytes, 1KB, 4KB), processing 10GB total in each configuration. We use random-generated payloads in a format similar to the one used in YCSB workloads and disable compression to write an equivalent amount of data in each setup. We use Zipfian-distributed keys (θ=0.99) across 100K files to simulate realistic patterns, with files written to a dedicated ext4 filesystem on a separate block device.

Figure 9(a) presents the throughput of GDPRuler's logging system. We observe that batching effectiveness is inversely proportional to entry size: smaller entries (256 bytes) benefit significantly from batching, achieving substantial throughput gains as batch size increases, while larger entries (4KB) have diminishing returns. Further, the 8-writer configuration outperforms the 4-writer setup for most configurations, demonstrating effective parallelization. At batch size 2048, the 8-writer configuration achieves 1354, 705 and 283 K entries/sec for 256-byte, 1KB, and 4KB entries, respectively.

The throughput metric (K entries/s) reveals that the system achieves consistent and efficient I/O performance across different entry sizes, as larger entries inherently carry larger payloads. This validates that the logging system's I/O pipeline handles varying data volumes effectively. The results also demonstrate an interaction between CPU resources and batching: with 8 writers, the system reaches a plateau shown in batch sizes 2048 and 8192. In contrast, the resource-constrained configuration (4 threads) continues to benefit from larger batch sizes, as it compensates for limited CPU parallelism by amortizing per-batch processing overhead.

Figure 9(b) shows write amplification, measuring the ratio of actual bytes written to storage versus the original payload size. Write

KVS	KVS (MB)	GDPRuler KVS (MB)	Compr. level	GDPR logs (MB)		
				10%	50%	100%
Redis	103.27	112.42	0	44.47	221.11	463.20
			3	7.18	43.02	87.97
			6	6.80	41.62	84.45
RocksDB	144.40	173.06	0	44.44	221.07	463.17
			3	7.02	43.03	88.03
			6	6.80	41.63	84.44

Table 7: Storage requirements analysis for GDPRuler.

amplification remains consistently low (below 7%) across all configurations, validating that GDPRuler's logging format introduces minimal metadata overhead. The slight increase in write amplification for smaller entry sizes (256 bytes showing 6-7% overhead) reflects the relatively higher cost of per-entry headers, while larger entries (4KB showing < 1% overhead) amortize this cost.

KVS metadata storage overhead. Table 7 quantifies the storage overhead introduced by GDPR metadata. Redis KVS consumes 103.27 MB for 100K KV pairs with 1KB values, while GDPRuler requires 112.42 MB for KVS files, an 8.9% increase. RocksDB shows a similar pattern with 19.8% metadata overhead (144.40 MB vs. 173.06 MB). This overhead stems from embedding compliance metadata—purposes, objections, ownership, retention periods, and sharing permissions—in each value and is inversely proportional to value size; larger values dilute the relative metadata cost.

Audit log storage requirements. Table 7 presents GDPR log storage requirements with varying log percentages and compression levels. Without compression (level 0), logging all operations (100%) generates 463.20 MB of logs for 100K entries with 1M operations, yielding a 4.3:1 ratio of logs to original KVS size (103.27 MB). This substantial volume reflects comprehensive audit trail requirements where each operation generates a metadata-rich log entry. At 50% logging—representing realistic selective monitoring involving only sensitive data—uncompressed logs consume 221.11 MB (almost twice the KVS size), while conservative 10% logging generates 44.47 MB of audit trail. However, when compression (level 3) is used, it reduces log storage by 5× overall, reaching ≈88 MB for 100% logging. At 50%, logs are 43 MB (≈40% of KVS size). Level 6 offers negligible extra savings (≈2.1%) but increases CPU overhead [40, 61, 94].

7.3 Data I/O Subsystem Analysis

To contextualize GDPRuler's performance results and isolate the baseline costs of confidential computing from GDPR compliance, we evaluate the I/O behavior of CVMs.

Storage I/O performance. Figure 10 shows FIO benchmark [42] results comparing 4KB IOPS across native execution, standard VM, and CVM (SNP) configurations. We observe that sequential reads degrade to 61% and 57% of the native variant while random reads drop even further, to just 27% of native throughput for both VM and CVM configurations. In contrast, both sequential and random writes maintain near-native performance (95-99% of the native variant) for both VM and CVM configurations. We further observe that enabling halt polling [62] (SNP-poll) does not result in any significant differences from standard SNP configurations across all workloads. The cost of memory encryption remains modest (8-15%), demonstrating that virtualization dominates performance impact. These measurements align with previous CVM characterizations [73, 84]. Importantly, GDPRuler's logging system operates in sequential write patterns

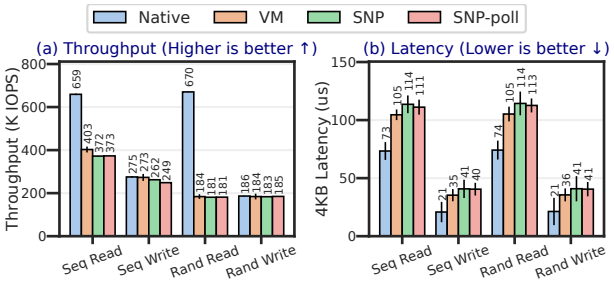


Figure 10: FIO performance analysis.

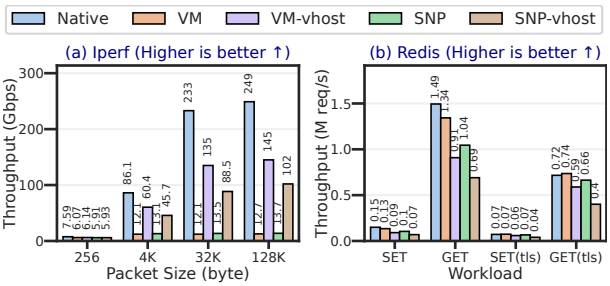


Figure 11: Iperf and Redis performance (multiqueue=8).

where CVMs maintain 95% of native performance, validating the almost negligible <2% logging overhead (§7.2).

Network I/O characteristics. Figure 11 presents network performance evaluation using iperf3 [52] (a) and Redis (b) with a virtio-net device (8 queues, 8 kernel vhost threads on host, 8 interrupt vectors in guest). vhost optimization [65] (VM-vhost, SNP-vhost) significantly improves throughput for large messages: for 128KB transfers, VM-vhost achieves 145 Gbps, and SNP-vhost reaches 102 Gbps, compared to 12-14 Gbps for standard VM/SNP configurations. However, CVMs show up to a 60% performance drop compared to native TCP, particularly as packet size increases. Redis workloads exhibit opposite behavior. For GET queries, a standard VM achieves 1.34 M req/s while VM-vhost delivers only 0.91 M req/s and SNP-vhost 0.69 M req/s. While vhost with multiqueue excels at iperf's bulk transfers via parallel processing, Redis's request-response pattern generates excessive software interrupt requests (softIRQs) and context switches from multiple host threads. This causes interrupt overhead that outweighs the benefits of parallelization, making standard paravirtualization a better choice for KVS workloads.

8 Related work

Policy compliance. A significant line of work explores policy-based compliance systems to address the requirements of privacy regulations. Systems such as Qapla [67], K9db [6], and Deep Enforcement [55] focus on policy-driven schema transformations and access control to ensure policy compliance. Other efforts, mainly for relational databases, formalize and enforce data policies using policy languages [17, 22, 79, 88, 101, 103], optimize for performance [36], or propose storage-centric policy compliance solutions [53, 58, 102]. On the GDPR frontier, prior work examines compliance challenges in the cloud [74, 91, 92, 97], its impact on storage systems [89], and proposed techniques to enforce GDPR regulations [21, 87, 90]. Unlike these approaches, GDPRuler is an end-to-end system for GDPR compliance that couples a rich policy language with verifiable execution for KVS.

Compliance plugins. Prior studies that investigate compliance frameworks for existing systems either provide transparent runtime policy enforcement for legacy SQL databases [1, 57, 59], enforce policies at the programming language layer [54], develop specialized auditing frameworks [2], or design enforcement systems for web applications [41, 99, 106] or storage systems [35, 58, 101]. These solutions either require intrusive application changes, such as at the database layer [6, 90], placing the burden on developers to build compliance mechanisms [1], or lack the security guarantees needed for compliance [41]. Moreover, they fail to provide such compliance in a verifiable manner on untrusted cloud infrastructures. Compared to them, GDPRuler enables GDPR compliance for KVS without requiring changes to the application or KVS engines.

Secure data management systems. A line of work focuses on secure cloud database and storage systems, including systems that ensure data security at the database level [12, 81] and support relational queries over encrypted data [11, 60, 77, 82, 100]. Hardware-based solutions [3, 15, 16, 31, 56, 70, 83, 93, 95, 98, 107] leverage TEEs to protect data during processing and at rest, while others extend security guarantees to distributed cloud environments [14, 44]. In contrast, GDPRuler focuses on GDPR compliance, addressing purpose limitation, data subject rights, and auditable logging, while preserving strong security guarantees for both data and metadata and ensuring compatibility with existing KV store deployments.

9 Conclusion

In this paper, we present GDPRuler, an end-to-end system enabling GDPR compliance for KVS without codebase modification. GDPRuler balances compliance and system performance using a comprehensive policy language, a secure runtime monitor, and an efficient metadata management system. Deploying the enforcement engine within a CVM ensures private data and metadata protection throughout the processing pipeline. Our evaluation shows that GDPRuler achieves GDPR compliance with reasonable performance overhead, minimizes metadata storage via optimized encoding, and provides tamper-evident logging and efficient GDPR queries, demonstrating its suitability for cloud adoption.

Extended version. A detailed tech report is available at [96].

Acknowledgments

This work was supported in part by an ERC Starting Grant (ID: 101077577) and the Chips Joint Undertaking (JU), European Union (EU) HORIZON-JU-IA, under grant agreement No. 101140087 (SMARTY), the Intel Trustworthy Data Center of the Future (TDCoF), Google Research Grants, Portuguese national funds through Fundação para a Ciência e a Tecnologia, I.P. (FCT) under projects UID/50021/2025 (DOI: https://doi.org/10.54499/UID/50021/2025) and UID/PRR/50021/2025 (DOI: https://doi.org/10.54499/UID/PRR/50021/2025), IAPMEI under grant C6632206063-00466847 (PT Smart Retail), and the European Union's Horizon Europe Research and Innovation Programme under Grant Agreement No. 101189689.

Ethical Considerations

The research team attests that we considered the ethics of this research. All experiments and evaluations were executed exclusively on local computing environments, ensuring that no live systems were affected. The research did not require the collection of personal or sensitive data. The findings of this research are intended to establish GDPR compliance and ensure data privacy and security for data management systems deployed on untrusted cloud infrastructures, and are intended for constructive and beneficial purposes.

Open Science

We openly share our research artifact, including the GDPRuler framework (Github link) and its associated security protocol proofs (Github link). Our artifact includes GDPRuler's code, its policy language and compiler, its formal verification proofs, and is accompanied by the respective README files. It also incorporates scripts that generate the workloads used in GDPRuler's evaluation and others that provide guidance for its configuration, deployment, and evaluation on AMD SEV-SNP machines.

References

[1] Archita Agarwal, Marilyn George, Aaron Jeyaraj, and Malte Schwarzkopf. 2021. Retrofitting GDPR compliance onto legacy databases. *Proc. VLDB Endow.* 15, 4 (Dec. 2021), 958–970. doi:10.14778/3503585.3503603

[2] Rakesh Agrawal, Roberto Bayardo, Christos Faloutsos, Jerry Kiernan, Ralf Rantzau, and Ramakrishnan Srikant. 2004. Auditing compliance with a Hippocratic database. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30* (Toronto, Canada) *(VLDB '04)*. VLDB Endowment, 516–527.

[3] Adil Ahmad, Kyungtae Kim, Muhammad Sarfaraz, and Byoungyoung Lee. 2018. OBLIVIATE: A Data Oblivious Filesystem for Intel SGX. doi:10.14722/ndss.2018.23296

[4] Erdem Aktas, Cfir Cohen, Josh Eads, James Forshaw, and Felix Wilhelm. 2024. Intel Trust Domain Extensions (TDX) Security Review.

[5] A K M Mubashwir Alam and Keke Chen. 2023. Making Your Program Oblivious: A Comparative Study for Side-channel-Safe Confidential Computing. In *Proceedings of the IEEE 16th International Conference on Cloud Computing*. Institute of Electrical and Electronics Engineers.

[6] Kinan Dak Albab, Ishan Sharma, Justus Adam, Benjamin Kilimnik, Aaron Jeyaraj, Raj Paul, Artem Agvanian, Leonhard Spiegelberg, and Malte Schwarzkopf. 2023. K9db: Privacy-Compliant Storage For Web Applications By Construction. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 99–116. https://www.usenix.org/conference/osdi23/presentation/albab

[7] Amazon. 2025. Amazon DynamoDB. https://aws.amazon.com/dynamodb/.

[8] AMD. 2020. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf

[9] AMD. 2022. TECHNICAL GUIDANCE FOR MITIGATING EFFECTS OF CIPHERTEXT VISIBILITY UNDER AMD SEV REVISION 5.10.22. https://www.amd.com/system/files/documents/221404394-a_security_wp_final.pdf

[10] AMD. 2023. SEV Secure Nested Paging Firmware ABI Specification Revision: 1.55. https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/specifications/56860.pdf

[11] Panagiotis Antonopoulos, Arvind Arasu, Kunal D. Singh, Ken Eguro, Nitish Gupta, Rajat Jain, Raghav Kaushik, Hanuma Kodavalla, Donald Kossmann, Nikolas Ogg, Ravi Ramamurthy, Jakub Szymaszek, Jeffrey Trimmer, Kapil Vaswani, Ramarathnam Venkatesan, and Mike Zwilling. 2020. Azure SQL Database Always Encrypted. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1511–1525. doi:10.1145/3318464.3386141

[12] Arvind Arasu, Ken Eguro, Raghav Kaushik, Donald Kossmann, Pingfan Meng, Vineet Pandey, and Ravi Ramamurthy. 2017. Concerto: A High Concurrency Key-Value Store with Integrity. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 251–266. doi:10.1145/3035918.3064030

[13] ARM. 2025. Arm Confidential Compute Architecture. https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture

[14] Maurice Bailleu, Dimitra Giantsidi, Vasilis Gavrielatos, Do Le Quoc, Vijay Nagarajan, and Pramod Bhatotia. 2021. Avocado: A Secure In-Memory Distributed Storage System. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 65–79. https://www.usenix.org/conference/atc21/presentation/bailleu

[15] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. 2019. SPEICHER: Securing LSM-based Key-Value Stores using Shielded Execution. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*. USENIX Association, Boston, MA, 173–190. https://www.usenix.org/conference/fast19/presentation/bailleu

[16] Sumeet Bajaj and Radu Sion. 2014. TrustedDB: A Trusted Hardware-Based Database with Privacy and Data Confidentiality. *IEEE Transactions on Knowledge and Data Engineering* 26, 3 (2014), 752–765. doi:10.1109/TKDE.2013.38

[17] Chinmayi Baramashetru, Silvia Lizeth Tapia Tarifa, Olaf Owe, and Nils Gruschka. 2022. *A Policy Language to Capture Compliance of Data Protection Requirements*. 289–309. doi:10.1007/978-3-031-07727-2_16

[18] Marcus Brandenburger, Christian Cachin, Matthias Lorenz, and Rüdiger Kapitza. 2017. Rollback and Forking Detection for Trusted Execution Environments using Lightweight Collective Memory. arXiv:1701.00981 [cs.DC] https://arxiv.org/abs/1701.00981

[19] Brian Frank Cooper. 2025. YCSB. https://github.com/brianfrankcooper/YCSB.

[20] Brian Frank Cooper. 2025. YCSB - Running a Workload. https://github.com/brianfrankcooper/YCSB/wiki/Running-a-Workload#step-5-load-the-data.

[21] Vishal Chakraborty, Stacy Ann-Elvy, Sharad Mehrotra, Faisal Nawab, Mohammad Sadoghi, Shantanu Sharma, Nalini Venkatsubhramanian, and Farhan Saeed. 2023. Data-CASE: Grounding Data Regulations for Compliant Data Processing Systems. arXiv:2308.07501 [cs.DB] https://arxiv.org/abs/2308.07501

[22] Haobin Hiroki Chen, Hongbo Chen, Mingshen Sun, Chenghong Wang, and XiaoFeng Wang. 2025. Picachv: Formally Verified Data Use Policy Enforcement for Secure Data Analytics. arXiv:2501.10560 [cs.CR] https://arxiv.org/abs/2501.10560

[23] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2023. Intel TDX Demystified: A Top-Down Approach.

[24] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2024. Intel TDX Demystified: A Top-Down Approach. *ACM Comput. Surv.* (mar 2024). doi:10.1145/3652597

[25] CMS: law-tax-future. 2025. GDPR Enforcement Tracker. https://www.enforcementtracker.com/.

[26] Confidential Computing Consortium. 2022. Confidential Computing: Hardware-Based Trusted Execution for Applications and Data: November 2022, V1.3. https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/CCC_outreach_whitepaper_updated_November_2022.pdf

[27] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud computing (SoCC)*.

[28] DB-Engines. 2025. DB-Engines Ranking of Key-value Stores. https://db-engines.com/en/ranking/key-value+store.

[29] Sen Deng, Mengyuan Li, Yining Tang, Shuai Wang, Shoumeng Yan, and Yinqian Zhang. 2023. CipherH: Automated Detection of Ciphertext Side-channel Vulnerabilities in Cryptographic Implementations. In *Proceedings of the 32nd USENIX Security Symposium*. USENIX Association.

[30] Tamarin Developers. [n. d.]. Tamarin Prover. https://tamarin-prover.com/

[31] Bernard Dickens III, Haryadi S. Gunawi, Ariel J. Feldman, and Henry Hoffmann. 2018. StrongBox: Confidentiality, Integrity, and Performance Using Stream Ciphers for Full Drive Encryption. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[32] Tom Dohrmann. 2025. mushroom: Run integrity protected workloads in a hardware based Trusted Execution Environment. https://github.com/Freax13/mushroom?tab=readme-ov-file#time

[33] Danny Dolev and Andrew Yao. 1983. On The Security of Public Key Protocols. *IEEE Transactions on Information Theory* 29, 2 (1983), 198–208. doi:10.1109/TIT.1983.1056650

[34] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. RocksDB: Evolution of Development Priorities in a Key-value Store Serving Large-scale Applications. *ACM Trans. Storage* 17, 4, Article 26 (2021), 32 pages. https://doi.org/10.1145/3483840

[35] Eslam Elnikety, Aastha Mehta, Anjo Vahldiek-oberwagner, Deepak Garg, and Peter Druschel. 2016. Thoth : Comprehensive Policy Compliance in Data Retrieval Systems. In *Proceedings of the USENIX Security Symposium*. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/elnikety

[36] Eslam Elnogomi, Deepak Garg, and Peter Druschel. 2018. Shai: Enforcing Data-Specific Policies with Near-Zero Runtime Overhead. (01 2018). doi:10.48550/arXiv.1801.04565

[37] EU. 2025. Art. 17 GDPR Right to erasure. https://gdpr-info.eu/art-17-gdpr/.

[38] EU. 2025. Art. 18 GDPR Right to restriction of processing. https://gdpr-info.eu/art-18-gdpr/.

[39] EU. 2025. Art. 5 Principles relating to processing of personal data. https://gdpr-info.eu/art-5-gdpr/.

[40] Euccas. [n. d.]. Understanding zlib. https://www.euccas.me/zlib/

[41] Mafalda Ferreira, Tiago Brito, José Fragoso Santos, and Nuno Santos. 2023. RuleKeeper: GDPR-Aware Personal Data Compliance for Web Frameworks. In *2023 IEEE Symposium on Security and Privacy (SP)*. 2817–2834. doi:10.1109/SP46215.2023.10179395

[42] Fio Developers. 2025. Flexible I/O Tester. https://github.com/axboe/fio.

[43] Jean-loup Gailly and Mark Adler. 2004. zlib compression library. (12 2004).

[44] Dimitra Giantsidi, Maurice Bailleu, Natacha Crooks, and Pramod Bhatotia. 2022. Treaty: Secure Distributed Transactions. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 14–27. doi:10.1109/DSN53405.2022.00015

[45] Google. 2019. Google Cloud Confidential Computing. https://cloud.google.com/confidential-computing/

[46] Google. 2025. LevelDB: a fast key-value storage library. https://github.com/google/leveldb.

[47] Haoyang Huang, Fengwei Zhang, Shoumeng Yan, Tao Wei, and Zhengyu He. 2024. SoK: A Comparison Study of Arm TrustZone and CCA. In *Proceedings of the 2024 International Symposium on Secure and Private Execution Environment Design*. doi:10.1109/SEED61283.2024.00021

[48] Intel. 2018. Host Firmware Speculative Execution Side Channel Mitigation. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/host-firmware-speculative-side-channel-mitigation.html

[49] Intel. 2022. Guidelines for Mitigating Timing Side Channels Against Cryptographic Implementations. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/mitigate-timing-side-channel-crypto-implementation.html

[50] Intel. [n. d.]. Intel® Trust Domain Extensions (Intel TDX). https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html

[51] Intersoft consulting. 2025. General Data Protection Regulation (GDPR). https://gdpr-info.eu/.

[52] iPerf Developers. [n. d.]. The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/

[53] Zsolt István, Soujanya Ponnapalli, and Vijay Chidambaram. 2021. Software-defined data protection: low overhead policy compliance at the storage layer is within reach! *Proc. VLDB Endow.* 14, 7 (March 2021), 1167–1174. doi:10.14778/3450980.3450986

[54] Farzane Karami, David Basin, and Einar Broch Johnsen. 2022. DPL: A Language for GDPR Enforcement. In *2022 IEEE 35th Computer Security Foundations Symposium (CSF)*. 112–129. doi:10.1109/CSF54842.2022.9919687

[55] Ety Khaitzin, Julian James Stephen, Maya Anderson, Hani Jamjoom, Ronen Kat, Arjun Natarajan, Roger Raphael, Roee Shlomo, and Tomer Solomon. 2019. Deep Enforcement: Policy-based Data Transformations for Data in the Cloud. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. USENIX Association, Renton, WA. https://www.usenix.org/conference/hotcloud19/presentation/khaitzin

[56] Igjae Kim, J. Hyun Kim, Minu Chung, HyunGon Moon, and Sam H. Noh. 2022. A Log-Structured Merge Tree-aware Message Authentication Scheme for Persistent Key-Value Stores. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*. USENIX Association, Santa Clara, CA, 363–380. https://www.usenix.org/conference/fast22/presentation/kim-igjae

[57] David Klein, Benny Rolle, Thomas Barber, Manuel Karl, and Martin Johns. 2023. General Data Protection Runtime: Enforcing Transparent GDPR Compliance for Existing Applications. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 3343–3357. doi:10.1145/3576915.3616604

[58] Robert Krahn, Bohdan Trach, Anjo Vahldiek-Oberwagner, Thomas Knauth, Pramod Bhatotia, and Christof Fetzer. 2018. Pesos: policy enhanced secure object store. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 25, 17 pages. doi:10.1145/3190508.3190518

[59] Tim Kraska, Michael Stonebraker, Michael Brodie, Sacha Servan-Schreiber, and Daniel Weitzner. 2019. *SchengenDB: A Data Protection Database Proposal*. 24–38. doi:10.1007/978-3-030-33752-0_2

[60] Mingyu Li, Xuyang Zhao, Le Chen, Cheng Tan, Huorong Li, Sheng Wang, Zeyu Mi, Yubin Xia, Feifei Li, and Haibo Chen. 2023. Encrypted Databases Made Secure Yet Maintainable. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 117–133. https://www.usenix.org/conference/osdi23/presentation/li-mingyu

[61] Genome Research Limited. [n. d.]. A comparison of Zlib implementations. https://www.htslib.org/benchmarks/zlib.html

[62] Linux Kernel Developers. 2025. Guest Halt Polling. https://docs.kernel.org/virt/guest-halt-polling.html.

[63] Luke Jones. 2025. Google Bard is Not Coming to Europe Because of Concerns it Breaches GDPR Rules. https://winbuzzer.com/2023/05/12/google-bard-is-not-coming-to-europe-because-of-concerns-it-breaches-gdpr-rules-xcxwbn/.

[64] LWN. 2024. Add Secure TSC support for SNP guests . https://lwn.net/Articles/979296/

[65] Eugenio P. Martín. 2019. Deep Dive into Virtio-networking and Vhost-net. https://www.redhat.com/en/blog/deep-dive-virtio-networking-and-vhost-net.

[66] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. 2017. ROTE: Rollback Protection for Trusted Execution. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1289–1306. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/matetic

[67] Aastha Mehta, Eslam Elnikety, Katura Harvey, Deepak Garg, and Peter Druschel. 2017. Qapla: Policy compliance for database-backed systems. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1463–1479. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/mehta

[68] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Proceedings of the 25th International Conference on Computer Aided Verification*. Springer-Verlag. doi:10.1007/978-3-642-39799-8_48

[69] Memcached Developers. 2025. Memcached. https://memcached.org/.

[70] Ines Messadi, Shivananda Neumann, Nico Weichbrodt, Lennart Almstedt, Mohammad Mahhouk, and Rüdiger Kapitza. 2021. Precursor: A Fast, Client-Centric and Trusted Key-Value Store Using RDMA and Intel SGX. In *Proceedings of the 22nd International Middleware Conference* (Québec city, Canada) *(Middleware '21)*. Association for Computing Machinery, New York, NY, USA, 1–13. doi:10.1145/3464298.3476129

[71] Meta Platforms. [n. d.]. RocksDB | A persistent key-value store for fast storage environments. https://rocksdb.org/.

[72] Microsoft. 2019. Azure Confidential Computing. https://azure.microsoft.com/en-us/solutions/confidential-compute

[73] Masanori Misono, Dimitrios Stavrakakis, Nuno Santos, and Pramod Bhatotia. 2024. Confidential VMs Explained: An Empirical Analysis of AMD SEV-SNP and Intel TDX. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 3 (2024). doi:10.1145/3700418

[74] Jayashree Mohan, Melissa Wasserman, and Vijay Chidambaram. 2019. Analyzing GDPR Compliance Through the Lens of Privacy Policy. arXiv:1906.12038 [cs.CY] https://arxiv.org/abs/1906.12038

[75] noyb. 2025. noyb: We enforce your right to privacy. https://noyb.eu/en.

[76] noyb. 2025. noyb win: € 1.2 billion fine against Meta over EU-US data transfers. https://noyb.eu/en/edpb-decision-facebooks-eu-us-data-transfers-stop-transfers-fine-and-repatriation.

[77] Mitsuhiro Okada, Takayuki Suzuki, Naoya Nishio, Hasitha Muthumala Waidyasooriya, and Masanori Hariyama. 2022. FPGA-Accelerated Searchable Encrypted Database Management Systems for Cloud Services. *IEEE Transactions on Cloud Computing* 10, 2 (2022), 1373–1385. doi:10.1109/TCC.2020.2969655

[78] OpenSSL. [n. d.]. OpenSSL library. https://openssl-library.org/

[79] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos Keromytis, and Steve Bellovin. 2014. Blind Seer: A Scalable Private DBMS. In *2014 IEEE Symposium on Security and Privacy*. 359–374. doi:10.1109/SP.2014.30

[80] Bryan Parno, Jacob R. Lorch, John R. Douceur, James Mickens, and Jonathan M. McCune. 2011. Memoir: Practical State Continuity for Protected Modules. In *2011 IEEE Symposium on Security and Privacy*. 379–394. doi:10.1109/SP.2011.38

[81] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. 2019. Arx: an encrypted database using semantically secure encryption. *Proc. VLDB Endow.* 12, 11 (July 2019), 1664–1678. doi:10.14778/3342263.3342641

[82] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2012. CryptDB: processing queries on an encrypted database. *Commun. ACM* 55, 9 (Sept. 2012), 103–111. doi:10.1145/2330667.2330691

[83] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A Secure Database Using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*. 264–278. doi:10.1109/SP.2018.00025

[84] Lina Qiu, Rebecca Taft, Alexander Shraer, and George Kollios. 2024. The Price of Privacy: A Performance Study of Confidential Virtual Machines for Database Systems. In *Proceedings of the 20th International Workshop on Data Management on New Hardware* (Santiago, AA, Chile) *(DaMoN '24)*. Association for Computing Machinery, New York, NY, USA, Article 2, 8 pages. doi:10.1145/3662010.3663440

[85] Redis. 2025. Redis - The Real-time Data Platform. https://redis.io/.

[86] Francisco Rocha and Miguel Correia. 2011. Lucy in the Sky without Diamonds: Stealing Confidential Data in the Cloud. In *Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops*. doi:10.1109/DSNW.2011.5958798

[87] Malte Schwarzkopf, Eddie Kohler, M. Frans Kaashoek, and Robert Morris. 2022. Position: GDPR Compliance by Construction. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer-Verlag, Berlin, Heidelberg, 39–53. doi:10.1007/978-3-030-33752-0_3

[88] Shayak Sen, Saikat Guha, Anupam Datta, Sriram K. Rajamani, Janice Tsai, and Jeannette M. Wing. 2014. Bootstrapping Privacy Compliance in Big Data Systems. In *2014 IEEE Symposium on Security and Privacy*. 327–342. doi:10.1109/SP.2014.28

[89] Aashaka Shah, Vinay Banakar, Supreeth Shastri, Melissa Wasserman, and Vijay Chidambaram. 2019. Analyzing the Impact of GDPR on Storage Systems. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*. USENIX Association, Renton, WA. https://www.usenix.org/conference/hotstorage19/presentation/banakar

[90] Supreeth Shastri, Vinay Banakar, Melissa Wasserman, Arun Kumar, and Vijay Chidambaram. 2020. Understanding and benchmarking the impact of GDPR on database systems. *Proc. VLDB Endow.* 13, 7 (March 2020), 1064–1077. doi:10.14778/3384345.3384354

[91] Supreeth Shastri, Melissa Wasserman, and Vijay Chidambaram. 2019. GDPR Anti-Patterns: How Design and Operation of Modern Cloud-scale Systems Conflict with GDPR. arXiv:1911.00498 [cs.CY] https://arxiv.org/abs/1911.00498

[92] Supreeth Shastri, Melissa Wasserman, and Vijay Chidambaram. 2019. The Seven Sins of Personal-Data Processing Systems under GDPR. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. USENIX Association, Renton, WA. https://www.usenix.org/conference/hotcloud19/presentation/shastri

[93] Chenkai Shen and Lei Fan. 2023. Pldb: Protecting LSM-based Key-Value Store using Trusted Execution Environment. In *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 762–771. doi:10.1109/TrustCom60117.2023.00111

[94] Juho Snellman. [n. d.]. Updated zlib benchmarks. https://www.snellman.net/blog/archive/2015-06-05-updated-zlib-benchmarks/

[95] Dimitrios Stavrakakis, Dimitra Giantsidi, Maurice Bailleu, Philip Sändig, Shady Issa, and Pramod Bhatotia. 2023. Anchor: A Library for Building Secure Persistent Memory Systems. *Proc. ACM Manag. Data* 1, 4, Article 231 (Dec. 2023), 31 pages. doi:10.1145/3626718

[96] Dimitrios Stavrakakis, Masanori Misono, Julian Pritzi, Harshavardhan Unnibhavi, Nuno Santos, and Pramod Bhatotia. 2026. Policy-Compliant Cloud Storage Systems. arXiv:2606.05423 [cs.CR] https://arxiv.org/abs/2606.05423

[97] Chen Sun, Evan Jacobs, Daniel Lehmann, Andrew Crouse, and Supreeth Shastri. 2023. GDPRxiv: Establishing the State of the Art in GDPR Enforcement. *Proceedings on Privacy Enhancing Technologies* (2023).

[98] Yuanyuan Sun, Sheng Wang, Huorong Li, and Feifei Li. 2021. Building Enclave-Native Storage Engines for Practical Encrypted Databases. *Proc. VLDB Endow.* 14, 6 (Feb. 2021), 1019–1032. doi:10.14778/3447689.3447705

[99] Lillian Tsai, Hannah Gross, Eddie Kohler, Frans Kaashoek, and Malte Schwarzkopf. 2023. Edna: Disguising and Revealing User Data in Web Applications. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 434–450. doi:10.1145/3600006.3613146

[100] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. 2013. Processing analytical queries over encrypted data. *Proc. VLDB Endow.* 6, 5 (March 2013), 289–300. doi:10.14778/2535573.2488336

[101] Harshavardhan Unnibhavi, David Cerdeira, Antonio Barbalace, Nuno Santos, and Pramod Bhatotia. 2022. Secure and Policy-Compliant Query Processing on Heterogeneous Computational Storage Architectures. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD)*.

[102] Anjo Vahldiek-Oberwagner, Eslam Elnikety, Aastha Mehta, Deepak Garg, Peter Druschel, Rodrigo Rodrigues, Johannes Gehrke, and Ansley Post. 2015. Guardat: enforcing data policies at the storage layer. In *Proceedings of the Tenth European Conference on Computer Systems* (Bordeaux, France) *(EuroSys '15)*. Association for Computing Machinery, New York, NY, USA, Article 13, 16 pages. doi:10.1145/2741948.2741958

[103] Frank Wang, James Mickens, Nickolai Zeldovich, and Vinod Vaikuntanathan. 2016. Sieve: Cryptographically Enforced Access Control for User Data in Untrusted Clouds. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[104] Wubing Wang, Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. 2023. PwrLeak: Exploiting Power Reporting Interface for Side-Channel Attacks on AMD SEV. In *Proceedings of the 20th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer-Verlag.

[105] Jan Wichelmann, Anna Pätschke, Luca Wilke, and Thomas Eisenbarth. 2023. Cipherfix: Mitigating Ciphertext Side-Channel Attacks in Software. In *Proceedings of the 32nd USENIX Security Symposium*. USENIX Association.

[106] Wen Zhang, Eric Sheng, Michael Chang, Aurojit Panda, Mooly Sagiv, and Scott Shenker. 2022. Blockaid: Data Access Policy Enforcement for Web Applications. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 701–718. https://www.usenix.org/conference/osdi22/presentation/zhang

[107] Wenchao Zhou, Yifan Cai, Yanqing Peng, Sheng Wang, Ke Ma, and Feifei Li. 2021. VeriDB: An SGX-based Verifiable Database. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2182–2194. doi:10.1145/3448016.3457308

[108] zlib. [n. d.]. zlib: A Massively Spiffy Yet Delicately Unobtrusive Compression Library. https://zlib.net/