Operon: An Encrypted Database for Ownership-Preserving Data Management

Sheng Wang, Yiran Li, Huorong Li, Feifei Li, Chengjin Tian, Le Su, Yanshan Zhang, Yubing Ma, Lie Yan, Yuanyuan Sun, Xuntao Cheng, Xiaolong Xie, Yu Zou
{sh.wang, yiranli.lyr, huorong.lhr, lifeifei, tianchengjin.tcj, le.su, yanshan.z, yubing.myb, yanlie.yl, yuanyuan.sun, xuntao.cxt, changbie.xxl, zouyu.zou}@alibaba-inc.com
Alibaba Group

ABSTRACT

The past decade has witnessed the rapid development of cloud computing and data-centric applications. While these innovations offer numerous attractive features for data processing, they also bring in new issues about the loss of data ownership. Though some encrypted databases have emerged recently, they can not fully address these concerns for the data owner. In this paper, we propose an *ownership-preserving database* (OPDB), a new paradigm that characterizes different roles' responsibilities from nowadays applications and preserves data ownership throughout the entire application. We build *Operon* to follow the OPDB paradigm, which utilizes the trusted execution environment (TEE) and introduces a behavior control list (BCL). Different from access controls that merely handle accessibility permissions, BCL further makes data operation behaviors under control. Besides, we make *Operon* practical for real-world applications, by extending database capabilities towards flexibility, functionality and ease of use. *Operon* is the first database framework with which the data owner exclusively controls its data across different roles' subsystems. We have successfully integrated *Operon* with different TEEs, *i.e.*, Intel SGX and an FPGA-based implementation, and various database services on Alibaba Cloud, *i.e.*, PolarDB and RDS PostgreSQL. The evaluation shows that *Operon* achieves 71% - 97% of the performance of plaintext databases under the TPC-C benchmark while preserving the data ownership.

PVLDB Reference Format:
Sheng Wang, Yiran Li, Huorong Li, Feifei Li, Chengjin Tian, Le Su, Yanshan Zhang, Yubing Ma, Lie Yan, Yuanyuan Sun, Xuntao Cheng, Xiaolong Xie, Yu Zou. *Operon*: An Encrypted Database for Ownership-Preserving Data Management. PVLDB, 15(12): 3332 - 3345, 2022.
doi:10.14778/3554821.3554826

1 INTRODUCTION

For the past few decades, database management systems have been evolving towards superior capabilities (e.g., efficiency, availability and scalability) [3, 11, 22, 42, 45, 61, 66] to cope with the surge of digital activities and data volumes. In contrast, practical techniques for database security have witnessed relatively less significant advances, where access control, file encryption, database audit have become de-facto standards [17, 19] that protect the database from unexpected accesses and external attacks. In conventional settings, a database system shall run in a private domain, and the system owners (as well as administrators) inherently have full access to the data inside. However, recent trends have overturned the assumption and brought new security issues.

The issues mainly come from two fundamental changes. The first one is the widespread adoption of cloud computing. Notice that, although nowadays cloud systems are able to offer many attractive features (*e.g.*, pay-as-you-go model, multi-region availability), they break the private domain assumption. This outsourced infrastructure could be compromised by insiders (*e.g.*, co-tenants or rogue staffs from the cloud). In particular, anyone with privileges (or physical access [35]) on that server can look into the databases and cause data breaches, which is out of customers' control. The second one is that the data-centric revolution has complicated the data management in applications. More specifically, the data needs to flow between different processing components, each of which is probably controlled by a different entity (*e.g.*, internal sub-divisions, business partners, and independent software vendors). Once the data flow into others' subsystems and databases, it is no longer under the control of the original data owner, leading to a contradiction between the utilization and the ownership of sensitive data.

On the surface, the above issues appear merely to be a confidentiality problem of database systems. To this end, several *encrypted database systems* have been built by both academia and industry, while most of them focus on protecting the confidentiality of sensitive data in outsourced databases. They either exploit special cryptographic primitives [13, 29, 49] to support data manipulation directly over ciphertext [12, 38, 50–52] or use trusted execution environments (TEE) [20, 37] to operate on sensitive data in an isolated *enclave* inaccessible from the rest of the host [2, 6, 27, 30, 39, 47, 53, 62, 65]. However, due to functional limitations, only a few of them are commercialized, *e.g.*, SQL Server Always-Encrypted [2]. Moreover, an even more fundamental problem has rarely been considered: existing solutions assume that the authorized endpoint directly interacting with the database is trusted and can touch sensitive data, which is hard to achieve in application subsystems hosted on the cloud or controlled by other entities.

Deep down, the above issues come from the loss of data ownership, where simply extending prior systems with data confidentiality protection is insufficient. Concerning this, we propose a new paradigm for the encrypted database. We name it *ownership-preserving database* (OPDB), with which the data is not even revealed to any subsystems and the data owner exclusively governs data accessibility. In a nutshell, all sensitive data remains in ciphertext wherever

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554826

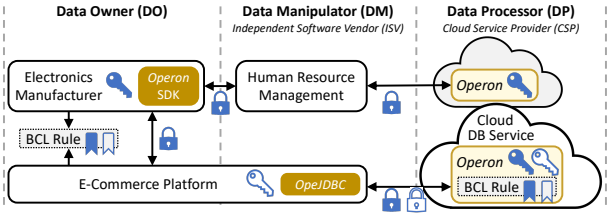


Figure 1: Illustration of supporting the OPDB with *Operon*. Data owners use their keys (differently colored) to protect data against any application processes or databases. *Operon* only performs permitted operations (*i.e.*, by BCL) within TEE.

it appears (*e.g.*, in the memory of application/database server processes or on the disk) and only the data owner can decrypt the data. Compared with the existing data owner - service provider model [33, 34], by adding a separate data manipulator role, OPDB brings data ownership into control. When an entity (*e.g.*, a business partner) needs to process or utilize the sensitive data in its business logic, the data owner only needs to grant it access to necessary operations that are adequate to complete the task using cipher processing capability in an OPDB instance. With OPDB, sensitive data can be securely passed and processed across different entities' subsystems and databases, which significantly reduces the risk of data abuses and leakages throughout the entire process.

This paper presents *Operon*[1], the first framework we built at Alibaba Cloud that follows the OPDB paradigm. From its core, *Operon* utilizes TEE to re-establish the private domain assumption. To prevent application system owners from compromising data ownership, *Operon* introduces a *behavior control list* (BCL), which extends conventional system-oriented resource access control with the control of data operation behaviors, decoupling data ownership and system ownership. To the best of our knowledge, *Operon* is the first database framework with which the data owner exclusively controls the accessibility and behaviors of sensitive data, even when the data has passed through many entities' untrusted subsystems. Figure 1 shows an example of supporting OPDB with *Operon*.

The design and implementation of *Operon* face two challenges. First, since different entities and cloud may have diverse infrastructures, a flexible architecture is necessary to make *Operon* performs as the basis of the OPDB paradigm across all subsystems, and to adapt to different database systems and TEEs. As a solution, *Operon* adopts a modular architecture and acts as a feature enhancement to existing database systems. We have successfully integrated *Operon* with different TEEs, *i.e.*, Intel SGX and an FPGA-based implementation, as well as various database services on Alibaba Cloud, *i.e.*, PolarDB [11] and RDS PostgreSQL. Second, since real-world applications always involve rich database functionalities (*e.g.*, mix-typed expression, connection pool, client driver) beyond basic operational primitives, *Operon* should also provide corresponding functionalities while preserving data ownership. Note that existing encrypted databases mostly focus on cipher processing primitives, short of fully-functional database support. *Operon* uses a set of server-side and client-side co-designs to address this challenge.

In summary, our major contributions are as follows:

- We propose the OPDB paradigm and achieve it using BCL. It is the first database solution with which the data owner can exclusively control its sensitive data across multiple subsystems and entities (Section 3).
- We build *Operon* to implement the above solution with TEE. By co-designing the server-side system and client-side SDK, *Operon* completes and extends database capabilities towards flexibility, functionality, and ease of use (Section 4).
- We analyze the security and privacy aspects of *Operon*, making a solid foundation for data owners to outsource data processing to other entities (Section 5).
- We discuss practical application scenarios and evaluate the performance of *Operon*. The result shows that *Operon* achieves 71% - 97% of the performance of plaintext databases under TPC-C, while preserving the data ownership (Section 6 & 7). *Operon* has commercialized on Alibaba Cloud through *RDS* [16] and *DataTrust* [15] services[2].

2 PRELIMINARIES AND OVERVIEW

In this section, we first brief the preliminaries about trusted execution environments and our threat model. We then provide an overview of *Operon*'s workflow using entities in Figure 1.

2.1 Trusted Execution Environments

A Trusted Execution Environments (TEE) provides a secure area called an *enclave* that helps to secure applications in untrusted environments where the host might conduct malicious or curious actions. Typically relying on special hardware support, TEE implementations are available on various platforms, such as Intel SGX [20, 46], AMD SEV [37], ARM TrustZone [43], RISC-V Keystone [41], as well as chip-based IBM 4758 [26] and FPGA-based AEGIS [56]. Though they are slightly different in either features or security assumptions, *Operon* only relies on their common features, including execution isolation, remote attestation and data sealing.

TEE requires the application to be divided into two memory-isolated parts, *i.e.*, trusted and untrusted, where the *untrusted* part runs as an ordinary process, and interacts with the *trusted* part inside the enclave through the prescribed interfaces (*e.g.*, ECall). Besides execution isolation, TEE also provides remote attestation (RA) that allows the client to verify the authenticity of an enclave and its loaded code/data on a remote host. It also facilitates the establishment of a secure channel between client and enclave, with which the data owner can safely pass over its secret keys. Furthermore, hardware-dependent data sealing enables the enclave to securely persist its internal data on the host, by deriving a sealing key from the hardware identity and loaded binary. Hence, only the original enclave on the original machine can read back its sealed data, preventing the data from malicious access.

For any TEE, a small trusted computing base (TCB) is always preferred, especially for a small loaded library, as it is more robust and can be exhaustively examined. However, this limits the complexity of the supported ciphertext operators. In addition, frequent-and-short invocations of the enclave will result in significant context switch overhead (*e.g.*, Intel SGX ECall is about several microseconds [24]). Hence, the choice of TCB is extremely vital to *Operon*.

[1]OPE-ron here stands for **O**wnership-**P**reserving **E**ncrypted database.

[2]Currently, *Operon* is only available in China regions.

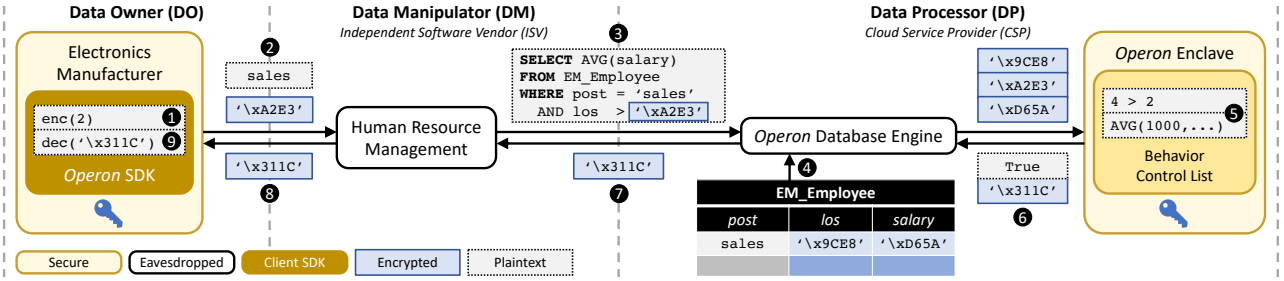


Figure 2: Detailed workflow of an electronics manufacturer using a cloud-based human resource management service (the top half of Figure 1). Data remains in ciphertext except the two secure endpoints: the manufacturer itself and the *Operon* enclave.

2.2 Threat Model

Operon aims to ensure that the data owner can exclusively control the ownership (*i.e.*, confidentiality, accessibility, and behavior) of its sensitive data fields hosted by untrusted entities' subsystems and databases. We target a *strong adversary* (*e.g.*, a third-party service provider) that has privileged access to the OS and database of any machines for all involved subsystems. It could not only monitor the content of resources (*e.g.*, memory, disk and network), but also tamper with the execution logic. However, the adversary cannot access enclaves. More specifically, data and execution inside an enclave are protected for confidentiality and integrity. The communication between the enclave and host (*e.g.*, invocations of ciphertext operators) is still exposed to the adversary. We exclude TEE side-channel attacks [48, 60], since these vulnerabilities are mostly implementation-specific and we can easily switch to more secure TEEs when needed (*e.g.*, Intel SGX leaks memory access pattern but FPGA-based TEEs do not).

Note that we do not pursue strict indistinguishability of ciphertext and corresponding operators, since it usually results in unacceptable performance overhead [27, 47]. Instead, we provide *operational data confidentiality* similar to prior work [2, 4, 52, 57] — what the adversary learns is a function of ciphertext operations that the data owner allows performing. For example, a *compare* with plaintext result will reveal the order information of ciphertext to the adversary. Since only necessary operations are granted, we can significantly reduce the risk of data abuses and leakages. In Section 5.3, we will analyze privacy leakage of operations in detail.

2.3 Examples of *Operon*'s Workflow

We zoom in the top half of Figure 1 to demonstrate how *Operon* utilizes TEE to preserve data ownership against other entities. In this example, an electronics manufacturer (*i.e.*, DO) uses a human resource management system (HRMS) to manage its organization, but worries that the HRMS provider (*i.e.*, DM) might obtain and sell its sensitive human resource data. The HRMS provider further chooses to use a CSP (*i.e.*, DP) to host its application, and also worries that its data might leak from the cloud side. The responsibilities of DO, DM and DP will be discussed formally in Section 3.1.

Figure 2 illustrates *Operon*'s overall workflow for the example. Suppose that the electronics manufacturer wants to find the average salary of all salesmen whose length of service (*i.e.*, `los`) is more than two years. With the `salary` and `los` fields encrypted, the company first prepares the ciphertext "2" using *Operon* SDK (❶), and then sends "`sales`" and ciphertext "`\xA2E3`" to the HRMS (❷). The HRMS then embeds ciphers into the query template "`SELECT AVG(salary) FROM EM_Employee WHERE post = ? AND los > ?`" and outsources the final query to *Operon* hosted by CSP (❸). In *Operon*, most stages of the query execution are as usual, such as query parsing, execution planning, table lookup, and filtering on position attribute (❹). However, part of the expression evaluation is performed in the enclave by comparing the ciphertext (*e.g.*, "`\xA2E3`" and "`\x9CE8`") via a compare operation, and those salary values from qualified records (*e.g.*, "`\xD65A`") are averaged via an average operation (❺). The final result "`\x311C`" is then securely sent back to the company (❻ - ❽), and gets decrypted by *Operon* SDK (❾).

To let *Operon* process ciphertext, DO needs to issue BCL (Section 3.2) and share secret keys to the enclave via a secure channel (Section 2.1). Hence, in the entire workflow, only two endpoints, *i.e.*, DO and *Operon*'s enclave, are able to obtain the plaintext data.

3 DATA OWNERSHIP MANAGEMENT

The *data ownership* discussed in this paper means that the data owner should exclusively control the accessibility and behavior of its data in whichever subsystems the data appears. Existing *encrypted database systems* fail to protect data ownership during the entire business process. Even though they have successfully provided confidentiality protection to sensitive data hosted by third-party entities, existing systems require that the client-side process, which interacts with the server-side encrypted database, can access the plaintext of sensitive data during query preparation and execution [2, 30, 51, 52, 62]. That is to say, the application subsystem (probably hosted on the cloud or controlled by other entities as discussed in Section 1) can still obtain sensitive data.

Preserving data ownership during the entire business process needs additional design restrictions to databases. In this section, we start by presenting the paradigm derived from analyzing the underlying requirements of ownership-preserving data management. It puts a lot of demands on the database beyond data encryption. Then, we show *Operon*'s solution of implementing the paradigm, by leveraging TEE to control the behavior of data processing.

3.1 Ownership-Preserving Database Paradigm

We now propose a new paradigm of an encrypted database named *ownership-preserving database* (OPDB), which by its own can preserve data ownership across all entities' subsystems and databases. The data owner only needs to trust the design and implementation of an OPDB, and no additional protections are required against other subsystem owners or application logic.

3.1.1 OPDB paradigm principles. The key to making the OPDB paradigm effective is to ensure that a system implementing the paradigm always preserves data ownership. Thus, at its core, the OPDB paradigm proposes three principles.

***Principle 1**: An entity can not access the sensitive data content without the data owner's authorization.* This principle guarantees the content of all sensitive data remains inaccessible wherever it appears in other entities' subsystems or databases (*e.g.*, in the memory of application/database processes or on the disk). This principle can be simply met by encrypting the data with its owner's secret key before sending it out to any entity.

***Principle 2**: An entity can only conduct authorized operations on sensitive data without knowing its content.* This principle permits the data owner to authorize other entities to manipulate the data, separating data ownership and application logic. As this principle prevents other entities from knowing the data content, the authorized entity cannot perform operations that reveal the data property knowledge (*e.g.*, relative orders of different data items).

***Principle 3**: An entity can only use authorized operations to learn properties of sensitive data.* This principle enables full-featured data processing by authorizing necessary data properties to other entities (*e.g.*, order property for building tree-based indexes).

With the above principles, we can safely pass and process sensitive data across different entities' subsystems and databases, while granting access to only necessary operations to preserve the data ownership. This significantly reduces the risk of data abuse and leakages throughout the entire business process.

3.1.2 Operators and measures. As above principles suggest, OPDB strictly differentiates operations that leak nothing (Principle 2) and those that return specific properties (Principle 3). We name operations under Principle 2 and Principle 3 as *operator* and *measure*, respectively. The term *measure* is borrowed from quantum computing, where measuring a qubit collapses its state. Likewise, in OPDB, after an operation under Principle 3, the data owner permanently shares the returned data property with the authorized entity.

More specifically, measures may reveal a limited degree of information (*i.e.*, property), but significantly facilitate the application execution. For example, a database may need relative data orders to build tree-based indexes or perform range queries; a business partner may need hashed identity to associate two records. To support various business processes and allow the data owner to grant restricted capabilities to an entity, the measure, as well as the Principle 3 it is designed to achieve, are indispensable for OPDB.

3.1.3 OPDB roles and responsibilities. OPDB paradigm abstracts out three roles: 1) *Data Owner* (DO), who owns the data and wants to utilize data in its business activities. It exclusively controls data accessibility and behaviors. 2) *Data Manipulator* (DM), who determines (alone or joint with DO) the purposes and means of data processing. It develops the software (*i.e.*, subsystem) to manipulate the data. 3) *Data Processor* (DP), who processes data on behalf of DM. It provisions infrastructural resources to run the subsystem and host the OPDB. These roles are similar to those defined in GDPR [59], but are derived from a different perspective: OPDB defines roles based on the rights and dependency over data ownership, while GDPR abstracts from the task separation in data processing.

The three roles cannot be further simplified as the abstraction reflects different responsibilities related to operators and measures. In general, the DO grants the DM operators and measures based on the application logic, and the DP enforces authorization while executing the operators and measures. Without the DO/DM separation, a DO cannot utilize third-party data processing applications while preserving data ownership, which suffers the limitation of existing encrypted databases. According to our threat model (Section 2.2), the DO neither trusts DM nor DP. However, without the DM/DP separation, a DO cannot gain the benefits of enjoying various services (*i.e.*, provided by DM) and trusting only a compact system (*i.e.*, inside TEE enclaves hosted by DP) at the same time. In other words, instead of expanding the trust to a DM, OPDB, by design, keeps it untrusted and ensures the DO is aware of the potential data leakage through the DM's processing logic.

In the real world, there are many examples of the above roles: a DM might be an internal sub-division or a business partner; a DP might be an on-premise datacenter or a cloud service provider. An entity might have multiple roles, *e.g.*, in Figure 1, the e-commerce platform is both DO and DM as it owns platform data and provides transaction services to merchants (*e.g.*, electronics manufacturer).

3.2 Data Behavior Control and Processing

Although leveraging cryptographic data encryption is sufficient to achieve the first principle of OPDB (Section 3.1), the second and third principles lead to additional requirements beyond data encryption: while the former demands for decoupling data ownership and operator processing, the latter depends on a fine-grained control of measure to preserve data ownership across multiple entities.

Access control list (ACL) is a canonical solution that allows an *issuer* to grant the access permission of particular objects he owns (*e.g.*, protected data, network interface cards) to a *subject*. However, this traditional approach is insufficient in our context. With ACL, the issuer (*i.e.*, DO) has to allow the subject (*i.e.*, DM and DP) to access the particular data encryption keys for data decryption. Once such access is granted, the data becomes out of the issuer's control. Instead, *Operon* proposes a *behavior control list* (BCL), which, on top of basic ACL concept, further controls the behavior of data (*e.g.*, cannot be viewed but is only allowed to participate in comparisons). TEE is used to validate the authenticity of the BCL, as well as enforce the defined data behaviors, so that *Operon* can ensure data ownership throughout the entire workflow process.

3.2.1 Ciphertext operators and measures. The BCL controls primitive cipher operations to perform information flow control (IFC) [5, 9, 10, 63]. Formally, we can define a cipher operation FUNC as

$$\mathsf{FUNC}_{\{c,p\}} : A \times B \times \cdots \rightarrow R,$$

where arguments (*i.e.*, A, B, *etc.*) may belong to different DOs, and the subscript c or p respectively denotes FUNC returns R in ciphertext or plaintext. According to OPDB, FUNC is an *operator* if and only if the subscript is c and all arguments have the same DO; otherwise, whenever the subscript is p or the arguments come from multiple DOs, FUNC is a *measure*. Thus, for instance, EQUAL_p represents an equality measure that returns plaintext, while ADD_c can be either a measure or operator based on its argument composition.

Table 1: BCL Specification

Field	Value	Description
ver	<version>	BCL specification version no.
serno	<uuid>	BCL rule serial number.
type	GRANT|REVOKE	Either grant or revoke ops.
i_id	<puk-id>	Issuer's public key ID.
s_id	<puk-id>	Subject's public key ID.
a_src	NULL|ISSU|ISSU_SUBJ	Argument source restriction.
i_dek	[{<min>,<max>},...]	List of issuer's DEK ranges.
s_dek	[{<min>,<max>},...]	List of subject's DEK ranges.
prep	NULL|MASK|FPHASH|...	Arg. preprocessing method.
ops	[NULL|<op>,...]	List of cipher operations.
postp	NULL|MASK|FPHASH|...	Result postprocessing method.
r_dek	NULL|SUBJ|<dek>	Result DEK, NULL for $\texttt{FUNC}_p$.

3.2.2 Behavior control list. *Operon* BCL implements the behavior control by defining and enforcing a list of operational tasks to be included in the BCL. To maximize ownership management flexibility, besides restricting data encryption key (DEK) and cipher operations, participating DOs can also negotiate preprocessing and postprocessing actions on data before and after the operation, respectively. Table 1 provides the abstraction of BCL specification. Notice that the preprocessing and postprocessing actions are required to be *format-preserving* (*e.g.*, format-preserving hashing that generates the hash value of the same type) to retain the operation semantics. To allow authorization under key rotation (Section 5.1), BCL accepts DEK ranges instead of individual DEKs, and uses `-1` and `INF` to denote unrestricted lower and upper boundaries.

One or multiple DOs can utilize BCL to authorize cipher operations. For a single DO, BCL helps the DO to configure for a richer operation set, or achieve a lower leakage by enforcing result encryption or disabling operations (*i.e.*, specifying `REVOKE` for `type`). For multiple DOs, BCL enables a DO to grant measures (*i.e.*, specifying `SUBJ` for `r_dek` or adding $\texttt{FUNC}_p$ to `ops`) to other DOs, while restricts information leakage with preprocessing and postprocessing actions. Figure 3 illustrates a simplified version of how BCL enables the addition of data from multiple DOs within TEE, ignoring the steps to find DEKs and match BCL rules.

3.2.3 Issuing BCL. For security considerations, issuing a new BCL requires both the issuer and the subject to sign it with their private keys. When a BCL only involves a single DO, the DO only needs to sign it once as the issuer. The DO can either manually sign the BCL through command line with system utilities (*e.g.*, OpenSSL), or call the corresponding function provided by SDK. Once issued, BCL is checked and enforced by *Operon*, and will not further involve the issuer DO in subsequent query processing. The detailed BCL issuing procedure and its security analysis are discussed in Section 5.2.

Existing literature has proposed numerous access control models, such as role-based access control (RBAC) and organization-based access control (OrBAC). As they perform batch authorization according to different target classification rules, BCL considers the problem from another perspective by performing operational behavior control to meet ownership requirements. To support those batch access control models, we can extend BCL with a virtual DO that represents multiple actual DOs, which is left as future work.

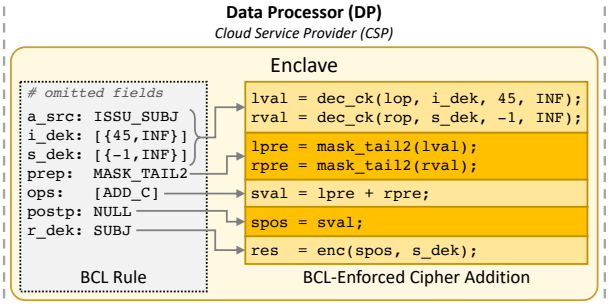


Figure 3: Illustration of how BCL enables the addition of data from two DOs. To preserve data ownership, the last two digits of the operands are masked before addition.

3.2.4 Database-compatible default BCL. The $\texttt{FUNC}_p$ and $\texttt{FUNC}_c$ return results of different data types (*i.e.*, plain or cipher type). To avoid ambiguity, only one of them can be configured as the default cipher operation to database operators. Consequently, targeting to minimize modification on the database system, the default BCL setting of *Operon* makes a trade-off among security, compatibility and the TCB size. For example, given that equi-join utilizes Boolean-typed equality test results to instruct execution engine, disabling $\texttt{EQUAL}_p$ and configuring $\texttt{EQUAL}_c$ as default operation will inevitably lead to a large TCB size and complex logic within TEE. *Operon* thus enables $\texttt{EQUAL}_p$ and configures it to be the default equality operation for all cipher types.

Formally, *Operon* enables a cipher operation $\texttt{FUNC}_p$ and configures it to be the default operation if and only if the operation `FUNC` is involved in control flow decisions. In other words, the default configuration of *Operon* enables condition measures (*e.g.*, $\texttt{EQUAL}_p$, $\texttt{LIKE}_p$) and sets them as default, while leaving the rest of operations as operators (*e.g.*, $\texttt{ADD}_c$) with corresponding measures disabled. To further preserve data ownership, a DO can still explicitly call $\texttt{FUNC}_c$ (*e.g.*, `equal_c(a,b)` that returns `enc_bool` result) and disable the corresponding $\texttt{FUNC}_p$ measure through BCL, at the cost of sacrificing feature richness. In general, BCL allows the DO to configure the privacy level of *Operon*, where the privacy level of other systems (*e.g.*, CryptDB [52]) can be imitated through this configuration. The default configuration of *Operon* follows the convention of enabling server-side filtering [2, 62]. The full operation set, the default system configuration and the corresponding privacy leakage (discussed in Section 5.3) are presented in Table 2.

4 SYSTEM DESIGN AND IMPLEMENTATION

Recall that in the OPDB paradigm, the DP and DM are responsible for provisioning infrastructural resources and developing the software to manipulate the data, respectively. Their demands for database systems are different from the DO. The DP needs a flexible and less-intrusive system architecture to provide a consistent and stable user experience across a variety of environmental specifications (Section 4.1). The DM is most concerned with the database functionalities that facilitate application development. Hence, *Operon* should make its new functionalities (*e.g.*, ciphertext operators and measures) easy to use, while making itself compatible with the way conventional plaintext databases are used (*e.g.*, connection pool). This is achieved by a co-design of server-side and client-side

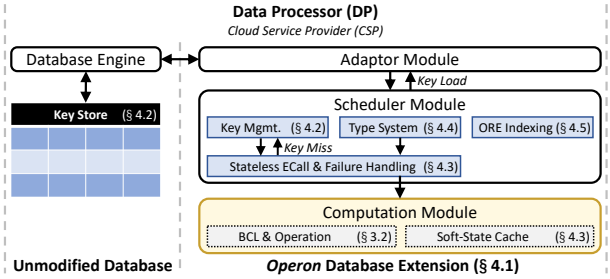


Figure 4: The *Operon* server-side architecture. The components of the *Operon* database extension consist of the extension, the scheduler, and the computation modules.

architectures. From the server side, *Operon* supports built-in key management (Section 4.2), connection pool and parallel processing (Section 4.3), type system for mix-typed cipher expression evaluation (Section 4.4), as well as indexing (Section 4.5). From the client side, *Operon* provides an SDK for explicit data encryption and decryption, as well as a driver that performs automatic handling of ciphertext, implementing the same interface as JDBC (Section 4.6).

4.1 Server-side Architecture

The *Operon*'s architecture reflects a joint consideration from the system's perspective: it protects primitive operations using TEE, and relies on the conventional database system outside the enclave to execute queries and store secured data. By carefully choosing the boundary between the trusted and untrusted parts, *Operon* implements OPDB with restricted TCB size, while achieving both easy product integration and seamless database feature compatibility.

4.1.1 Flexible architecture for easy integration. The naïve solution of putting the whole database into the enclave has two drawbacks: 1) it does not implement the OPDB paradigm; 2) it results in the dilemma of TCB (Section 2.1). Hence, as Figure 4 shows, *Operon*'s server-side architecture consists of three components, *i.e.*, the *adaptor*, the *scheduler*, and the *computation* modules. Only the computation module resides in the enclave, while the adaptor and scheduler modules execute together with unmodified conventional database system (*e.g.*, query parser and execution engine) and may be eavesdropped. This architecture makes *Operon* easy to adapt to various TEE and database implementations. We have built *Operon* with Intel SGX and an FPGA-based TEE, and integrated it into database systems like PolarDB and RDS PostgreSQL on Alibaba Cloud.

The computation module is the key to re-establish the private domain assumption. Since the module runs inside the enclave, it is protected against memory and execution attacks. *Operon* only puts minimal functionalities into this module, including the BCL, data operators, and measures, as well as required cryptographic primitives and caches. Benefiting from its simplicity, all the operations in this module can complete without nested interactions with the scheduler. Apart from the security guarantee, the simplicity of the computation module also improves the architectural flexibility. For example, to support an FPGA-based TEE plugged on PCIe, we only need to implement the computation module's interface with the arguments complying with the flat memory model. As the computation module code remains the same for different TEEs, *Operon* ensures consistent behavior on various specifications.

The adaptor and the scheduler improve the architectural flexibility from another angle, *i.e.*, ease for adapting to various databases. As different systems may have different methods of extending their functionality, the adaptor is responsible for connecting databases to the scheduler, and shielding the scheduler from system-specific functions, such as query execution and logging. Similarly, The scheduler abstracts the operations provided by *Operon*, and shields the extension module from implementation details (*e.g.*, the parameter of ECall). Furthermore, apart from behaving as a wrapper of the computation module, the scheduler also executes those operations that do not involve the enclave (*e.g.*, deducing operation signatures).

4.1.2 Database feature compatibility. *Operon* is designed to operate as a database extension. Apart from allowing easy integration, with the help of default BCL (Section 3.2), *Operon* also leads to a stable user experience, which is critical for cloud products. The extension only routes primitive cipher operations to *Operon*, keeping the rest of the database execution workflow intact. For functions requiring to access internal database tables (*e.g.*, key management in Section 4.2), the *Operon* extension interacts with the tables through a standard SQL interface. Subsequently, existing products can add *Operon* support without in-depth feature compatibility inspection.

More specifically, the majority of database features (*e.g.*, replication and recovery) are inherently compatible with *Operon*, as they perform data-content-independent manipulations. For clauses such as `JOIN`, `ORDER BY` and `GROUP BY`, benefit from the fact that the execution engine accomplishes them with primitive operations, *Operon* supports them without further mechanism beyond default BCL. To co-exist with high-availability deployment, *Operon* operates on read-only replicas by forwarding the write SQL, *i.e.*, MEK sealing (Section 4.2), to read-write replicas. Only those features involving cipher operations may require additional efforts, *e.g.*, *Operon* implements parallel processing and connection pool with stateless computation (Section 4.3) and adapts to ETL tools using client SDK and *OpeJDBC* (Section 4.6).

4.2 Key Management

The server-side *Operon* incorporates a built-in key management system, which (by taking full advantage of TEE) removes the need to trust and access additional key management services.

4.2.1 Key hierarchy. Similar to prior systems [2, 36], *Operon* adopts a two-layer key hierarchy: the lower layer data encryption key (DEK) encrypts data, and the upper layer master encryption key (MEK) further encrypts the DEKs. The benefit of a two-layer key hierarchy is twofold: it reduces the key management cost for DOs and also enables fine-grained ciphertext management.

Figure 5 shows this key management system. *Operon* indexes keys using various types of IDs, which are kept in key store tables and cached within TEE. While the MEK ID is globally unique and chosen randomly, the DEK ID and Cipher Context (CC) ID are locally unique, and increased monotonically under a specified MEK ID or a database instance, respectively. The cipher context performs as a shorthand for the full cryptographic metadata (*e.g.*, MEK ID and DEK ID to retrieve the DEK, cryptography algorithm), making it sufficient to include only CC ID in cipher header. By default, *Operon* associates a unique DEK for each encrypted table column, while

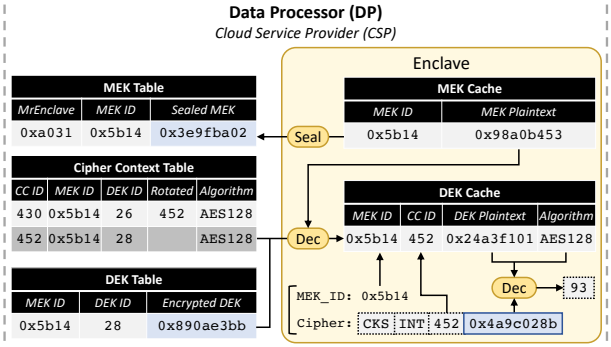


Figure 5: The *Operon* key management system. The three tables are internal database tables with specific fields (*i.e.*, "*Sealed MEK*" and "*Encrypted DEK*") being encrypted. The two caches are of plaintext form in the enclave.

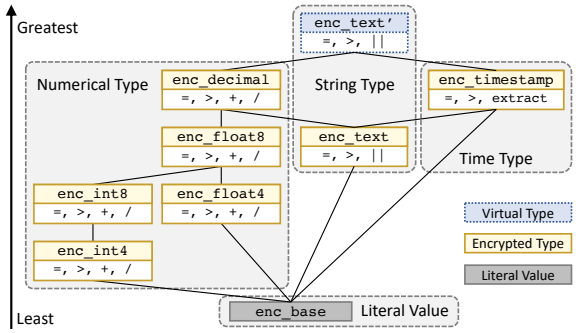


Figure 6: The *Operon*'s lattice of encrypted types. The box below each type name examples several operation overrides.

also allowing the DO to specify different granularities (*i.e.*, from table cell to database level). Due to the ambiguity of CC ID when migrating data among instances, *Operon* will reallocate CC IDs to the targeting instance and update cipher headers accordingly.

The security of the above design roots in the hardware-dependent sealing that is used to encrypt the MEK, which is secure according to the design of TEE. The security of the MEK provision, the DEK transfer from DO to the enclave, and the append-only key rotation, are analyzed in Section 5.1.

4.2.2 Cipher format. *Operon* adopts a compact cipher format. To reduce ciphertext expansion, *Operon* uses *CC ID* to index cryptographic metadata instead of including them explicitly, which typically saves over 16 bytes. Apart from the CC ID (*e.g.*, 452 in Figure 5), the cipher header also includes a checksum (*e.g.*, CKS) and a data type (*e.g.*, INT) fields. They are used to protect the ciphertext from unintentional corruption or misoperation, and guide the type system (Section 4.4) to decide the function overload, respectively.

4.3 Stateless Computation

The enclave inherently requires some state information to process ciphertext (*e.g.*, keys, interaction with callers), and this limits its usability. To address this, *Operon* provides stateless computation inside the computation module, which facilitates rich features, such as connection pool and parallel processing. The stateless computation permits cipher operations to be invoked like those over plaintext. Recall that the ECall parameters and ciphertexts contain sufficient metadata to perform corresponding operations. This makes the computation module be executed stateless naturally.

4.3.1 Failing-fast. The first step of making the computation module stateless is to eliminate stateful context-dependent operations. Although the computation module already operates without nested interactions, the enclave isolation still makes error handling challenging. For example, when the result buffer overflows, as the scheduler cannot access the memory allocated by the computation module, the computation module should switch to the scheduler to reallocate the buffer. Such context switches not only complicate the design, but also increase the attacking surface. Thus, instead of handling errors inside the enclave, *Operon* lets the computation module fail fast and carry out necessary exception messages as if the enclave is not isolated. To throw exceptions with arbitrary messages, *Operon* allocates a message buffer in the scheduler before ECall. When the buffer itself overflows, the computation module will utilize the same mechanism and fill the buffer with a fixed-length buffer doubling exception. Receiving the exception message, the scheduler is responsible for handling the exception and retrying with new parameters (*e.g.*, with a larger result buffer).

4.3.2 Caching soft-states. A fully stateless computation module requires the scheduler to prepare all the necessary information for each possible operation, including the keys and BCL rules. Supporting such a fully stateless computation will result in a high overhead as the enclave is only used for primitive-level operations here. Instead, *Operon* keeps caches (Section 4.2), and makes them cache soft-states inside the enclave. The soft-state is widely used in the domain of network protocol [14]. Different from the hard-state that is being maintained carefully, the use of soft-state improves efficiency, but is not necessary for computation. These soft-states can be discarded or generated as will. In *Operon*, all the states in the enclave, including keys and BCL rules, are soft-states. When cache access misses, *Operon* loads the cache by throwing an exception.

4.3.3 Connection pool and parallel processing. The stateless mode decouples the database connection from secrets (*i.e.*, keys and BCL rules) and operations. Although running in separate contexts, different computation module instances still execute the same binary on the same hardware platform, which generates the same sealing key (Section 2.1). Thus, any computation module instance is able to fetch and unseal a required MEK when it fails to find it in its cache. This allows the dynamic launch of computation module instances.

The stateless computation also enables connection pools and parallel processing. After a DO provisions its MEK to *Operon*, the subsequent connections created by the connection pool can obtain the MEK by unsealing corresponding table entry. Similarly, as parallel processing is helpful for long-running queries, *Operon* can inherit parallel capabilities from the integrated database easily.

4.4 *Operon* Type System

Conventional databases rely on a combination of implicit type conversions, default rules, priorities, *etc.*, to evaluate mixed expressions of different data types [18, 31]. Supporting mixed type expressions not only improves ease of use, but is also necessary to support

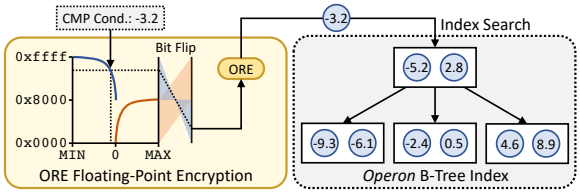


Figure 7: Illustration of the *Operon* B-tree index search on floating-point-typed column with the ORE acceleration.

various queries. For example, the aggregation operation usually returns a data type with higher precision than its operands (*e.g.*, the result of performing `sum` over integers is a long value). If the database only accepts operations among exactly matched types, writing a valid expression while ensuring the type correctness of each operand will become extremely complicated.

Operon proposes a type system that is specially designed for encrypted types. The implicit type cast has two main drawbacks when applying to encrypted types: firstly, each type cast operation introduces an ECall cost; and secondly, the type cast leaks information on failure. Different from conventional databases, *Operon* explicitly specifies all the possible function signature overrides according to its type system. At its core, the type system uses a lattice of encrypted types, as shown in Figure 6. The line represents the transitive relation of "greater than" (*e.g.*, `enc_float4` < `enc_decimal`). Note that the greatest element of the lattice is `enc_text'`, which is introduced to avoid loops between `enc_text` and other types. The operation set of `enc_text'` is the same as that of `enc_text`.

When matching function signatures, the type system first finds the least upper bound `enc_m` of argument types. Then, starting from the `enc_m`, the type system returns the matched type if it finds a registered signature; otherwise, it repeats by checking the least type `enc_m'` that satisfies `enc_m` < `enc_m'`. Finally, if the check of the greatest element `enc_text'` remains unmatched, the matching process fails. For example, `enc_int4` < `enc_text` matches at `enc_decimal`, while `enc_int4` || `enc_text` matches at `enc_text'`.

Operon implements the above type system by converting the lattice to its topological order. For n arguments and m encrypted types, the overhead of matching the function signature is $O(nm)$. The type system is also extensible: adding a new encrypted type only requires updating the lattice and its topological order.

4.5 Indexing

Due to the cost of ECall and cryptographic primitives, the overheads of ciphertext operations are much higher than plaintext operations, making the index way more important for *Operon* than for conventional databases. Existing encrypted databases leverage the original database index by using ciphertext operations to directly build indexes on ciphertexts [2]. This approach applies well to the hash index, where an index look-up only involves a single ECall to get the hash value of the condition, and a few ECalls to compare slot entries with the condition. However, for the tree-based index, each node requires multiple ciphertext comparison measures against a set of keys, which degrades the performance significantly.

Operon improves the tree-based index performance by replacing the ECall-based comparison measure with the ECall-less order-revealing encryption (ORE) measure. To achieve this, *Operon* adopts an extended version of the CLWW ORE algorithm [13]. The original CLWW takes a byte array as input, and uses special encryption and comparison algorithms to preserve the order of ciphertexts. Observing that the signed integer and floating-point numbers are of fixed lengths, we flip specific plaintext bits before CLWW encryption, making the byte-order consistent with the represented value. Figure 7 shows how *Operon* looks up a B-tree index.

Since the tree structure already leaks the order information, introducing CLWW will not further compromise data ownership. In fact, the use of ORE measures in tree-based indexes is similar to the use of hash measures in hash indexes, where hash measures also reduces ECalls. Note that *Operon* maintains a separate DEK for each column, and hence the use of ORE measures on different columns will not leak extra information beyond in-column orders. *Operon* requires the DO to issue BCL for ORE measures before it can create CLWW indexes.

4.6 Client-side SDK and *OpeJDBC*

Operon provides an *Operon* SDK for DO and DM to interact with a database. The SDK implements client-side functionalities for key management (Section 4.2), caches DEKs locally, encrypts data for a query and decrypts ciphertext results as well. However, migrating a legacy application to *Operon* using the SDK requires considerable efforts from both the DO and DM. DO needs to specify correct DEKs for thousands of tables, while DM needs to modify a huge codebase to adjust data types and process ciphertext.

To this end, *Operon* provides *OpeJDBC*, which analyzes the query statement and performs automatic data encryption/decryption by calling the SDK. Note that *OpeJDBC* is designed for cases that the DO also controls the caller applications' run-time environments. *OpeJDBC* uses the same interface as conventional JDBC, which greatly reduces the migration overhead. *OpeJDBC* achieves automatic encryption and decryption through query analysis. On each new query, *OpeJDBC* parses the query to an abstract syntax tree (AST), acquires from the *Operon* server the function signatures and table schema to label the AST, and replaces plaintexts with corresponding ciphertexts if necessary. Similar to the SDK, *OpeJDBC* also caches query analysis results and metadata to improve efficiency.

5 SECURITY & PRIVACY ANALYSIS

Here we discuss the overall security and privacy of *Operon* from a system perspective and further explain some of our design choices.

5.1 Key-related Security

With the carefully designed key hierarchy and its management system, the key-related security aspect are mainly the following:

5.1.1 MEK provision. The security of this process is guaranteed through the RA process between the DO and the TEE that validates the enclave's authenticity. The DO then encrypts the MEK using the enclave's public key and sends it to the server side. This encrypted MEK can only be decrypted (and cached) within the enclave.

5.1.2 DEK rotation. Standard key rotation technique applies here. However, we stress that deprecated DEKs are not erased. As data may get encrypted/decrypted at and copied to various locations at various times, maintaining those deprecated keys are vital to

Table 2: *Operon* Cipher Operations & Privacy Leakage

Operand Type	Primitive	Operation	Return Type	Leakage
enc_{int,float,decimal}[3]	+, −, ×, ÷, %, EXP	**Operator**	**enc_{int,float,decimal}**	**None.**
		Measure	int,float,decimal	Arithmetic operation result.
enc_{int,float,decimal}	SUM, AVG, MIN, MAX	**Operator**	**enc_{int,float,decimal}**	**None.**
		Measure	int,float,decimal	Aggregation result.
enc_text	SUBSTRING, ‖	**Operator**	**enc_text**	**None.**
		Measure	string	String manipulation result.
enc_timestamp	EXTRACT_YEAR	**Operator**	**enc_int**	**None.**
		Measure	int	Year of the timestamp.
enc_bool	NOT,AND,OR	**Operator**	**enc_bool**	**None.**
		Measure	bool	Boolean operation result.
enc_{int,float,decimal,text,timestamp}	=, ≠, >, <, ≥, ≤	Operator	enc_bool	None.
		Measure	**bool**	**Operands order.**
enc_text	LIKE	Operator	enc_bool	None.
		Measure	**bool**	**String match result.**
enc_{int,float,decimal,text,timestamp}	HASH	**Measure**	**bytes**	**Hashed value.**
All encrypted types	COUNT	**Measure**	**int**	**Number of input items.**

[3] For simplicity, we omit size suffixes (*i.e.*, 4 and 8) of int and float data types.

avoid complicated consistency problems. The security of DEKs, even those deprecated ones, is protected through MEK encryption. Key rotation is vulnerable to rollback attacks, same as all systems that support this functionality. *Operon* adopts classical approaches (*e.g.*, client-side or trusted counter [44]) to solve the freshness issue.

5.2 BCL Security

When multiple users and their data are involved in a specific workload, *Operon* requires both the issuer and subject to sign the BCL (*i.e.*, the subject must be aware of being granted with certain capabilities), to prevent potential forged BCL attack that may result in data leakage. Without such a dual-signature (*i.e.*, only the issuer signs the BCL), an adversary (*e.g.*, a malicious DBA) could forge data and DEKs, and grant a victim access to these DEKs by issuing a BCL without alerting him. Subsequently, the adversary assumes the role of the victim (DBA could easily do this), processes the queries, and encrypts results using the forged DEKs over the victim's data (as the victim has been granted access to use these DEKs). In this manner, it can obtain the processing results. Although the adversary DOES NOT have access to the victim's data (*i.e.*, no victim's DEK access), the leakage comes from the results of crafted query (*e.g.*, equality comparison) to infer the victim's data.

For easy verification of the BCL signatures, the public key information and MEK IDs of system users are stored in a key pair table. To further protect the integrity of the stored key information, it is signed using TEE's enclave-embedded private key. Users could verify the signature to assure the authenticity of the public keys used to generate the BCLs.

5.3 Privacy Leakage

Operon allows the users to choose a balance between the data privacy and system usability, through the careful design and separation of primitives into either *operator* or *measure* as defined in Section 3.1. Table 2 provides a (non-exhaustive) list of primitives that are supported in *Operon* and their corresponding privacy leakage based on whether they are executed as *operators* or *measures*. In *Operon*, most of the primitives, except COUNT and HASH, could be used in both modes, and those in bold indicate the default setting in *Operon*. The privacy consideration for not defining the *operator* version for COUNT is that this aggregation function is typically applied after a filtering condition (*e.g.*, comparisons and/or LIKE) with a bool return value. This non-encrypted bool already leaks the privacy of the comparison result, rendering an *operator* COUNT meaningless. For HASH, the reason is straightforward: an encrypted hash result defeats the purpose of using hash operation in the first place, that is to make equality comparisons efficient compared to throwing encrypted values into the enclave.

The leakage induced by *measure* inevitably degrades the overall security strength. For example, a HASH measure leaks the hashed values of plaintext data. In the event of a small plaintext domain, an adversary may brute force the original value. A more complicated attack may also apply to encrypted values if their relevant orders are leaked through the compare measure, and subsequently recover the plaintext value. However, these privacy leakage and induced security degradation are not caused by the design of *Operon*, but rather inherited from the use of various measures. The users could still adhere to a high level of security, *e.g.*, by disabling the hash measure and putting encrypted values into enclave for comparison. However, these approaches greatly degrade the overall performance and system usability. Nevertheless, *Operon* provides a flexible design that allows the users to configure according to their requirements and guarantees the corresponding security and privacy level.

6 APPLICATION

Operon preserves data ownership while compatible with conventional databases. In this section, we list several application scenarios covering different stages of the software lifecycle, including development, deployment, operation, and maintenance.

Legacy application migration. Due to the lack of maintainers, legacy but critical applications are inherently hard to migrate. Even worse, since operating on encrypted data is essentially different

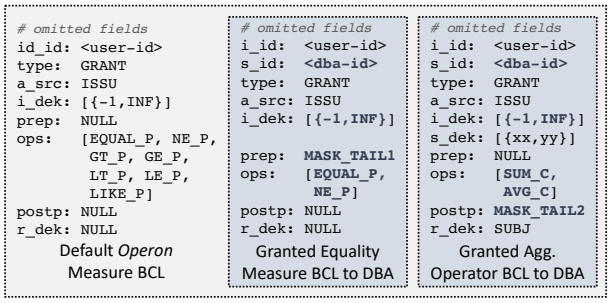


Figure 8: An example of granting DBA diagnosis operations using BCL. The DBA can only perform approximated equality measures and aggregation operators.

from that on plaintext data, existing encrypted databases introduce limited operations and non-standard interfaces, which typically require modifying or even refactoring to migrate. *Operon* provides applications with *OpeJDBC* (Section 4.6), which complies with standard JDBC interfaces. *OpeJDBC* automatically encrypts input plaintext data before querying and decrypt ciphertext results transparently. The application only needs to replace the original JDBC driver with *OpeJDBC* and embed MEK into the connection info. Typically, this only involves several lines of modification (in configuration file), easing the migration of legacy applications.

Data-centric application deployment. In data-centric applications, data protection is crucial for data-driven decision-making. However, for an external offline deployment of such an application, data has to be shipped together, and hence a malicious user may be able to steal the data by, *e.g.*, diskcopy or memory dump. In *Operon*, as data is always encrypted outside the enclave both at rest and on the fly, even a veteran attacker can learn nothing from either database storage or runtime memory, which is guaranteed by TEE. Combined with the BCL to restrict the allowed actions during user's joint analysis, DO and ISV can ensure that the ownership of the shipped data will not be compromised.

Database diagnosis. Database administrator usually needs to locate problems based on specific inputs and outputs. However, as existing encrypted databases strictly prevent anyone except the owner to access the data, diagnosis on those databases is difficult or even impossible. By using *Operon*'s ownership-preserving functionality regulated by BCL, the data owner can grant the DBA the permission to access desensitized data. The benefits are three-fold: 1) the DBA can perform SQL queries, observe results, and locate problems just like on normal database; 2) the doubly-signed BCL enables the owner to clearly know what the DBA might learn from the data; and 3) as the data are dynamically desensitized, the owner can specify proper desensitization rules based on the data security level and minimum requirements of the DBA. Figure 8 shows an BCL example of granting DBA diagnosis operations.

7 EVALUATION

We evaluate *Operon* using both micro- and macro-benchmarks: Sysbench [40] and OLTP-Bench [23] (containing benchmarks for TPC-C [21] and SmallBank[1]). *Operon* has been implemented on two TEE platforms: Intel SGX and an FPGA-based TEE developed by Alibaba Cloud (denoted as FPGA hereafter). We evaluate and compare both of them.

Table 3: SQL Templates of Sysbench Microbenchmark

Operation	SQL template
Point Query	`SELECT c FROM sbtest WHERE id = ?`
Range Query	`SELECT c FROM sbtest WHERE id BETWEEN ? AND ?`
Update	`UPDATE sbtest SET c = ? WHERE id = ?`
Insert	`INSERT INTO sbtest (id, k, c, pad) VALUES (?, ?, ?, ?)`

Table 4: TEE-related Overheads on SGX and FPGA

TEE	Enclave Init.		ECall		Encryption		Decryption	
	Time	Pct.	Time	Pct.	Time	Pct.	Time	Pct.
SGX	33 µs	0.11%	4.4 µs	47%	1.8 µs	0.01%	1.8 µs	33%
FPGA	785 ms	2.61%	67 µs	50%	39 µs	1.0%	24 µs	36%

7.1 Setup

7.1.1 Hardware specification. We run the evaluations on a client-server setup. For the SGX version of *Operon*, we use ApsaraDB RDS for PostgreSQL (with *Operon* feature [16]) of `pg.x8t.3xlarge.2c` instance type (24 vCPU, 192 GB of memory and one 2TB PL1 ESSD, cloud SSD with maximum IOPS of 50K) as the database server and an ECS instance with the type of `ecs.c7.4xlarge` (16 vCPU, 32 GB of memory and one 40 GB ESSD) as the client server where benchmarks are deployed and executed. All these resources are available in the Alibaba Cloud.

For the FPGA version, we use two servers of the same specification (16 vCPU, 32 GB of memory and one 40 GB ESSD) for the database server and client, except that the server is equipped with the FPGA card (containing two soft RISC-V IP cores running at 180 MHz and a cryptography algorithm acceleration logic).

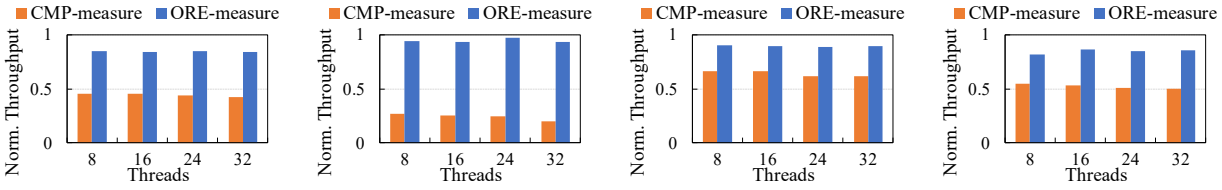
7.1.2 Benchmark configuration. We use Sysbench to evaluate the performance of *Operon* for point and range look-up queries, updates and inserts. Table 3 shows the detailed test SQL templates. We use Sysbench to load 32 tables, each containing a million records. All columns in these 32 tables are encrypted when we evaluate *Operon*.

We use OLTP-Bench [23] to run the TPC-C and SmallBank benchmarks on *Operon*. To achieve peak performance, we run the benchmark on four client instances at the same time with 128 client drivers. We use 256 warehouses for TPC-C, 8 million accounts for SmallBank. In TPC-C and SmallBank, we do not encrypt the ID columns which are non-sensitive sequence numbers.

We evaluate the version of *Operon* with PostgreSQL 13 [32]. If not otherwise specified, *Operon* is configured to use the randomized AES-CBC [25] mode for data encryption in all tests.

7.2 Microbenchmark

7.2.1 TEE implementations. We run the OLTP_read_write benchmark of Sysbench to evaluate the overhead of TEE-related operations in *Operon* (*i.e.*, enclave initialization, ECall, encryption, and decryption). Table 4 shows the time consumption of a single operation and the percentage of the corresponding operation's total time



(a) Point query. (b) Range query. (c) Update. (d) Insert.

Figure 9: Normalized index performance using Sysbench with varying number of client threads.

consumption in the OLTP_read_write Sysbench test. We can see that ECall and decryption take most of the execution time on both platforms, as the benchmark contains many comparisons. Each comparison needs to trigger ECall and perform data decryption within the enclave. The FPGA implementation has a similar breakdown at a slower execution, compared with Intel SGX. In the rest of this paper, we only report results achieved on the SGX platform.

7.2.2 Index performance. We evaluate the overall throughput of queries on encrypted records using indexes with and without ORE acceleration (Section 4.5), denoted as `ORE-` and `CMP-measure`. Figure 9 shows the throughput achieved with an increasing number of client threads, normalized to those of querying plaintext records using plaintext indexes. Figure 9b shows that using ciphertext as index entry without the ORE acceleration introduces high overheads (73.4% slower than the plaintext index). With the ORE acceleration, *Operon* achieves similar throughputs with those of the plaintext index (84.5% and 96.7% for point and range queries, respectively). ORE also improves the throughputs of update and insert (Figure 9c and 9d), as both these operations involve comparisons in the index.

7.3 Macrobenchmarks

Since TEE-related overheads introduced by the encrypted database can not be easily ignored, existing works (e.g., AE [2], EnclaveDB [53], and StealthDB [62]) show that the encrypted databases they built incur about 30% - 70% penalty to the unencrypted ones, and the penalty usually increases as the proportion of encrypted data increases. In this test, we evaluate *Operon* by using two different encryption settings in the TPC-C workloads to show the impact of different encrypted column settings. We first adopt the same setting as AE's evaluation [2] (denoted as `6-Col`), *i.e.*, encrypting name columns (*i.e.*, `C_FIRST` and `C_LAST`) and address columns (*i.e.*, `C_STREET_1`, `C_STREET_2`, `C_CITY`, and `C_STATE`) of the `Customer` table. We further encrypt all 58 columns except the ID columns (denoted as `58-Col`) as a comparison. RND and DET encryptions are used in the evaluation.

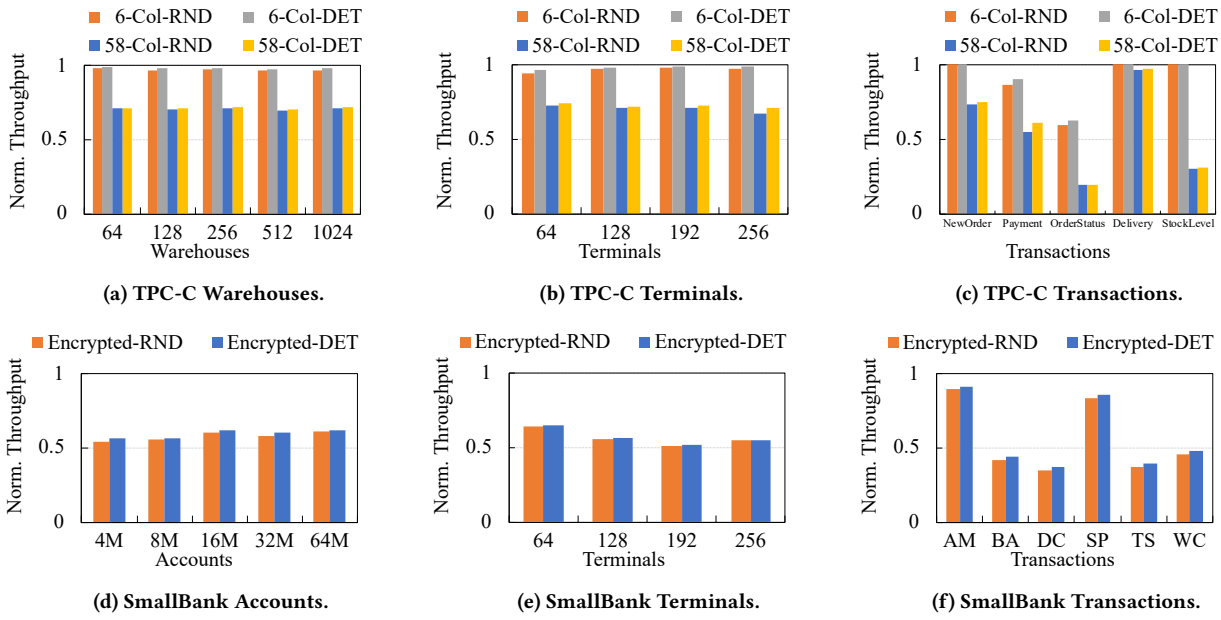
Figure 10a-10c show the results of the TPC-C benchmark, normalized to the plaintext TPC-C baseline achieved in the same database setup. For the `6-Col` setting, we can see that the performance of *Operon* with various warehouses and terminals are on par with the plaintext TPC-C results. Specifically, *Operon* with RND encryption achieves up to 97.7% and 97.9% throughput with different warehouses and different terminals, respectively. Only the `Payment` and `Order Status` transactions (account for 43% and 4% in the workload, respectively) involve operations on the six encrypted columns, which have limited performance impact in this configuration. Hence, the performance penalty is negligible. For the `58-Col` setting, *Operon* with RND encryption achieves up to 71.0% and 72.4% performance of the plaintext TPC-C benchmark. The performance of RND encryption is similar to DET encryption.

Figure 10c shows the throughputs of the five transactions in TPC-C. Only the `Payment` and `Order Status` transactions contain the encryption operations when we test using the `6-Col` setting, and *Operon* achieves 86.3% and 59.1% throughput compared with the plaintext TPC-C in these two transactions, respectively. For the `58-Col` setting, we find that the performance penalty is negligible for the `Delivery` transaction, while the penalty increases by more than 70% for the `Order Status` and `Stock Level` transactions. The former contains complex update and query SQLs with two client encryption operations, thus the encryption overhead is negligible compared with the transaction overhead. The latter only contains simple query SQLs, increasing the proportion of the encryption overhead significantly and making the overall performance sensitive to the number of columns encrypted.

Figure 10d-10f present the results of the SmallBank benchmark. Since each table of SmallBank only contains two columns, we only encrypt the non-ID column. While using the RND encryption, *Operon* achieves more than 53.9% and 50.8% throughput compared with the original plaintext SmallBank with different accounts and terminals settings, respectively. Note that the transactions of SmallBank are mainly composed of simple select SQLs, which leads to the performance of *Operon* on SmallBank to be similar to that under point query mentioned above (*i.e.*, the `CMP-measure` of Figure 9a). Figure 10f shows the performance with various transaction types. For clarity, we use "AM", "BA", "DC", "SP", "TC", "WC" to denote the transaction names of `Amalgamate`, `Balance`, `Deposit Checking`, `Send Payment`, `Transact Savings`, `Write Check`, respectively. *Operon* is 34.9% - 89.0% as fast as the plaintext result.

8 RELATED WORK

Privacy protection against databases. Data privacy has been considered a "particularly vital problem" since the Database-as-a-Service (DBaaS) model was proposed [34]. Observing much of the data processing can be done on ciphertexts, Hacigümüş *et al.* made the first attempt to address the issue of data privacy in relational databases. They use the DBaaS model to characterize the user and the service provider [33]. In the following two decades, the above model forms the basis for encrypted databases [2, 4, 28, 58, 62]. To address recent changes discussed in Section 1, we extend the DBaaS model to an OPDB model that categorizes more roles in



(a) TPC-C Warehouses. **(b) TPC-C Terminals.** **(c) TPC-C Transactions.**

(d) SmallBank Accounts. **(e) SmallBank Terminals.** **(f) SmallBank Transactions.**

Figure 10: Normalized TPC-C and SmallBank performance.

real applications. Therefore, data owners can not only protect data privacy against database hosts, but also control data accessibility and behavior throughout the entire application path.

Encrypted databases with TEEs. Earlier attempts on building encrypted databases, such as TrustedDB [7] and Cipherbase [4] use TEEs connected on PCI-X or PCIe. Although the appearance of CPU-embedded TEE (*e.g.*, Intel SGX) has greatly improved the performance, we found that PCIe-connected TEEs with physical resource isolation still have their application scenarios. Using more enclave resources, Haven [8] and EdgelessDB [30] explore a straightforward approach to utilize TEE to protect the whole database system. However, due to the difficulty of restricting DBA permissions, they add only limited security enhancement than traditional transparent data encryption (TDE). SQL Server Always Encrypted [2] and StealthDB [62] puts ciphertext operations into the enclave, which is somehow similar to what *Operon* does. SQL Server AE supports comparison and string pattern matching operations that return boolean results in plaintext (similar to measures in *Operon*). StealthDB provides various encrypted types and operations on PostgreSQL [62], but it needs the client to encrypt the plain SQL as a whole. EnclaveDB [53] puts the entire query engine in an enclave to achieve stronger security protection (*e.g.*, integrity and freshness), but it also needs the client to be fully trusted. In summary, most TEE-based encrypted databases assume a trusted client that can touch plaintext of sensitive data. *Operon* solves this issue by categorizing more types of roles (*i.e.*, DO/DM/DP).

Encrypted databases with PPEs. Property-preserving encryption (PPE) is an alternative building block for encrypted databases. They support a subset of ciphertext operations at the various cost of computation complexity or information leakage [13, 29, 49, 55]. Symmetria [54] presents two symmetric homomorphic encryption schemes for efficient addition and multiplication, respectively. CryptDB [52] achieves a good balance between information leakage and functionality with the SQL-aware encryption strategy. Arx [51] introduces ArxRange and ArxEq primitives that achieve semantic security. MONOMI [58] and KafeDB [64] further improve functionality by splitting execution between server and client, following DBaaS paradigm [33]. However, they all assume a trusted proxy or client front, which is difficult to fulfill in real applications.

9 CONCLUSION AND FUTURE WORK

In this paper, we propose the paradigm of *ownership-preserving database* (OPDB), which characterizes the demand of nowadays applications for decoupling data ownership and system ownership throughout complex business processes. We present *Operon*, an encrypted database framework that utilizes TEE to protect sensitive data and follows the OPDB paradigm. In particular, *Operon* implements the behavior control list (BCL) to preserve the data ownership by taking the operation behavior into consideration. To reduce the overhead of migrating legacy applications, *Operon* also enables conventional database functionalities including connection pool, mixed-type expressions, client driver, *etc.* The current implementation of *Operon* supports different TEEs and database products on Alibaba Cloud, and achieves 71% - 97% of the performance of plaintext databases under the TPC-C benchmark. As a database enhancement, *Operon* has served several customers in both cloud and on-premise deployment to protect their sensitive data in application subsystems and databases controlled by others.

Operon is still being improved from various aspects. One ongoing work is to utilize both system-level and cryptographic optimizations to reduce the ciphertext operation overhead. We also face the challenge of query optimization that adapts to different databases and TEE implementations. Other improvements include extending *Operon* with various access control models, index information leakage reduction, built-in ownership risk analysis tool, *etc.*

REFERENCES

[1] Mohammad Alomari, Michael Cahill, Alan Fekete, and Uwe Rohm. 2008. The Cost of Serializability on Platforms That Use Snapshot Isolation. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering (ICDE '08)*. IEEE Computer Society, USA, 576–585. https://doi.org/10.1109/ICDE.2008.4497466

[2] 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 International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1511–1525. https://doi.org/10.1145/3318464.3386141

[3] Panagiotis Antonopoulos, Alex Budovski, Cristian Diaconu, Alejandro Hernandez Saenz, Jack Hu, Hanuma Kodavalla, Donald Kossmann, Sandeep Lingam, Umar Farooq Minhas, Naveen Prakash, Vijendra Purohit, Hugh Qu, Chaitanya Sreenivas Ravella, Krystyna Reisteter, Sheetal Shrotri, Dixin Tang, and Vikram Wakade. 2019. Socrates: The New SQL Server in the Cloud. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1743–1756. https://doi.org/10.1145/3299869.3314047

[4] Arvind Arasu, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, and Ravi Ramamurthy. 2015. Transaction processing on confidential data using cipherbase. In *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering (ICDE '15)*. IEEE Computer Society, USA, 435–446.

[5] Eugene Bagdasaryan, Griffin Berlstein, Jason Waterman, Eleanor Birrell, Nate Foster, Fred B. Schneider, and Deborah Estrin. 2019. Ancile: Enhancing Privacy for Ubiquitous Computing with Use-Based Privacy. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society* (London, United Kingdom) *(WPES '19)*. Association for Computing Machinery, New York, NY, USA, 111–124. https://doi.org/10.1145/3338498.3358642

[6] 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 *Proceedings of the 17th USENIX Conference on File and Storage Technologies* (Boston, MA, USA) *(FAST '19)*. USENIX Association, USA, 173–190.

[7] Sumeet Bajaj and Radu Sion. 2011. TrustedDB: a trusted hardware based database with privacy and data confidentiality. In *Proceedings of the 2011 International Conference on Management of Data* (Athens, Greece) *(SIGMOD '11)*, Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis (Eds.). Association for Computing Machinery, New York, NY, USA, 205–216. https://doi.org/10.1145/1989323.1989346

[8] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding Applications from an Untrusted Cloud with Haven. *ACM Transactions on Computer Systems (TOCS)* 33, 3, Article 8 (Aug. 2015), 26 pages. https://doi.org/10.1145/2799647

[9] Eleanor Birrell, Anders Gjerdrum, Robbert van Renesse, Håvard Johansen, Dag Johansen, and Fred B. Schneider. 2018. SGX Enforcement of Use-Based Privacy. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society* (Toronto, Canada) *(WPES '18)*. Association for Computing Machinery, New York, NY, USA, 155–167. https://doi.org/10.1145/3267323.3268954

[10] Lukas Burkhalter, Nicolas Küchler, Alexander Viand, Hossein Shafagh, and Anwar Hithnawi. 2021. Zeph: Cryptographic enforcement of end-to-end data privacy. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI '21)*. USENIX Association, USA, 387–404.

[11] Wei Cao, Yingqiang Zhang, Xinjun Yang, Feifei Li, Sheng Wang, Qingda Hu, Xuntao Cheng, Zongzhi Chen, Zhenjun Liu, Jing Fang, Bo Wang, Yuhui Wang, Haiqing Sun, Ze Yang, Zhushi Cheng, Sen Chen, Jian Wu, Wei Hu, Jianwei Zhao, Yusong Gao, Songlu Cai, Yunyang Zhang, and Jiawang Tong. 2021. PolarDB Serverless: A Cloud Native Database for Disaggregated Data Centers. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2477–2489. https://doi.org/10.1145/3448016.3457560

[12] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. 2013. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Annual cryptology conference*. Springer, 353–373.

[13] Nathan Chenette, Kevin Lewi, Stephen A Weis, and David J Wu. 2016. Practical order-revealing encryption with limited leakage. In *International conference on fast software encryption*. Springer, 474–493.

[14] David D. Clark. 1988. The Design Philosophy of the DARPA Internet Protocols. In *Symposium Proceedings on Communications Architectures and Protocols* (Stanford, California, USA) *(SIGCOMM '88)*. Association for Computing Machinery, New York, NY, USA, 106–114. https://doi.org/10.1145/52324.52336

[15] Alibaba Cloud. 2022. Alibaba Cloud DataTrust Service. Retrieved March 1, 2022 from https://dp.alibaba.com/product/datatrust

[16] Alibaba Cloud. 2022. Alibba Cloud ApsaraDB RDS for PostgreSQL with Operon. Retrieved March 1, 2022 from https://help.aliyun.com/document_detail/144156.html

[17] Microsoft Corporation. 2022. Row-Level Security. Retrieved March 1, 2022 from https://docs.microsoft.com/en-us/sql/relational-databases/security/row-level-security

[18] Oracle Corporation. 2022. MySQL Reference Manual 12.3 Type Conversion in Expression Evaluation. Retrieved March 1, 2022 from https://dev.mysql.com/doc/refman/8.0/en/type-conversion.html

[19] Oracle Corporation. 2022. Virtual Private Database. Retrieved March 1, 2022 from https://www.oracle.com/database/technologies/virtual-private-db.html

[20] Victor Costan and Srinivas Devadas. 2016. Intel sgx explained. *IACR Cryptol. ePrint Arch.* 2016, 86 (2016), 1–118.

[21] Transaction Processing Performance Council. 2010. TPC Benchmark C. Retrieved March 1, 2022 from http://www.tpc.org/tpcc

[22] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 215–226. https://doi.org/10.1145/2882903.2903741

[23] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proceedings of the VLDB Endowment* 7, 4 (Dec. 2013), 277–288. https://doi.org/10.14778/2732240.2732246

[24] Tu Dinh Ngoc, Bao Bui, Stella Bitchebe, Alain Tchana, Valerio Schiavoni, Pascal Felber, and Daniel Hagimont. 2019. Everything You Should Know About Intel SGX Performance on Virtualized Systems. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, 1, Article 5 (March 2019), 21 pages. https://doi.org/10.1145/3322205.3311076

[25] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. 2001. Advanced Encryption Standard (AES). https://doi.org/10.6028/NIST.FIPS.197

[26] Joan G. Dyer, Mark Lindemann, Ronald Perez, Reiner Sailer, Leendert van Doorn, Sean W. Smith, and Steve Weingart. 2001. Building the IBM 4758 Secure Coprocessor. *Computer* 34, 10 (Oct. 2001), 57–66. https://doi.org/10.1109/2.955100

[27] Saba Eskandarian and Matei Zaharia. 2019. ObliDB: Oblivious Query Processing for Secure Databases. *Proceedings of the VLDB Endowment* 13, 2 (Oct. 2019), 169–183. https://doi.org/10.14778/3364324.3364331

[28] Benny Fuhry, HA Jayanth Jain, and Florian Kerschbaum. 2021. Encdbdb: Searchable encrypted, fast, compressed, in-memory database using enclaves. In *Proceedings of the 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '21)*. IEEE Computer Society, USA, 438–450.

[29] Craig Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing* (Bethesda, MD, USA) *(STOC '09)*. Association for Computing Machinery, New York, NY, USA, 169–178. https://doi.org/10.1145/1536414.1536440

[30] Edgeless Systems GmbH. 2022. EdgelessDB Official Website. Retrieved March 1, 2022 from https://www.edgeless.systems/products/edgelessdb

[31] The PostgreSQL Global Development Group. 2022. PostgreSQL Documentation Chapter 10 Type Conversion. Retrieved March 1, 2022 from https://www.postgresql.org/docs/current/typeconv.html

[32] The PostgreSQL Global Development Group. 2022. PostgreSQL Official Website. Retrieved March 1, 2022 from https://www.postgresql.org

[33] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. 2002. Executing SQL over Encrypted Data in the Database-Service-Provider Model. In *Proceedings of the 2002 International Conference on Management of Data* (Madison, Wisconsin) *(SIGMOD '02)*. Association for Computing Machinery, New York, NY, USA, 216–227. https://doi.org/10.1145/564691.564717

[34] Hakan Hacigumus, Bala Iyer, and Sharad Mehrotra. 2002. Providing database as a service. In *Proceedings of the 2002 IEEE 18th International Conference on Data Engineering (ICDE '02)*. IEEE Computer Society, USA, 29–38.

[35] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. 2009. Lest We Remember: Cold-Boot Attacks on Encryption Keys. *Communications of the ACM (CACM)* 52, 5 (May 2009), 91–98.

[36] Amazon Web Services Inc. 2022. AWS KMS key hierarchy. Retrieved March 1, 2022 from https://docs.aws.amazon.com/kms/latest/cryptographic-details/key-hierarchy.html

[37] Advanced Micro Devices Incorporated. 2005. Secure Virtual Machine Architecture Reference Manual.

[38] Yuval Ishai, Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. 2016. Private large-scale databases with distributed searchable symmetric encryption. In *Cryptographers' Track at the RSA Conference*. Springer, 90–107.

[39] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. 2019. ShieldStore: Shielded In-Memory Key-Value Storage with SGX. In *Proceedings of the Fourteenth EuroSys Conference 2019* (Dresden, Germany) *(EuroSys '19)*. Association for Computing Machinery, New York, NY, USA, Article 14, 15 pages. https://doi.org/10.1145/3302424.3303951

[40] Alexey Kopytov. 2022. SysBench. Retrieved March 1, 2022 from https://github.com/akopytov/sysbench

[41] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An Open Framework for Architecting Trusted Execution Environments. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 38, 16 pages. https://doi.org/10.1145/3342195.3387532

[42] Feifei Li. 2019. Cloud-Native Database Systems at Alibaba: Opportunities and Challenges. *Proceedings of the VLDB Endowment* 12, 12 (Aug. 2019), 2263–2272. https://doi.org/10.14778/3352063.3352141

[43] Arm Limited. 2022. TrustZone. Retrieved March 1, 2022 from https://www.arm.com/technologies/trustzone-for-cortex-a

[44] 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 *Proceedings of the 26th USENIX Conference on Security Symposium* (Vancouver, BC, Canada) *(SEC '17)*. USENIX Association, USA, 1289–1306.

[45] Yoshinori Matsunobu, Siying Dong, and Herman Lee. 2020. MyRocks: LSM-Tree Database Storage Engine Serving Facebook's Social Graph. *Proceedings of the VLDB Endowment* 13, 12 (Aug. 2020), 3217–3230. https://doi.org/10.14778/3415478.3415546

[46] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. Innovative Instructions and Software Model for Isolated Execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy* (Tel-Aviv, Israel) *(HASP '13)*. Association for Computing Machinery, New York, NY, USA, Article 10, 1 pages. https://doi.org/10.1145/2487726.2488368

[47] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. 2018. Oblix: An efficient oblivious search index. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy (S&P '18)*. IEEE Computer Society, USA, 279–296.

[48] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. 2020. A survey of published attacks on Intel SGX. *arXiv preprint arXiv:2006.13598* (2020).

[49] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques* (Prague, Czech Republic) *(EUROCRYPT '99)*. Springer-Verlag, Berlin, Heidelberg, 223–238.

[50] 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 *Proceedings of the 2014 IEEE Symposium on Security and Privacy (S&P '14)*. IEEE Computer Society, USA, 359–374. https://doi.org/10.1109/SP.2014.30

[51] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. 2019. Arx: An Encrypted Database Using Semantically Secure Encryption. *Proceedings of the VLDB Endowment* 12, 11 (July 2019), 1664–1678. https://doi.org/10.14778/3342263.3342641

[52] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (Cascais, Portugal) *(SOSP '11)*. Association for Computing Machinery, New York, NY, USA, 85–100. https://doi.org/10.1145/2043556.2043566

[53] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A secure database using SGX. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy (S&P '18)*. IEEE Computer Society, USA, 264–278.

[54] Savvas Savvides, Darshika Khandelwal, and Patrick Eugster. 2020. Efficient Confidentiality-Preserving Data Analytics over Symmetrically Encrypted Datasets. *Proceedings of the VLDB Endowment* 13, 8 (April 2020), 1290–1303. https://doi.org/10.14778/3389133.3389144

[55] Dawn Song, David A. Wagner, and Adrian Perrig. 2000. Practical Techniques for Searches on Encrypted Data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P '00)*. IEEE Computer Society, USA, 44–55.

[56] G. Edward Suh, Charles W. O'Donnell, and Srinivas Devadas. 2007. Aegis: A Single-Chip Secure Processor. *IEEE Design & Test of Computers* 24, 6 (Nov. 2007), 570–580. https://doi.org/10.1109/MDT.2007.179

[57] Yuanyuan Sun, Sheng Wang, Huorong Li, and Feifei Li. 2021. Building Enclave-Native Storage Engines for Practical Encrypted Databases. *Proceedings of the VLDB Endowment* 14, 6 (Feb. 2021), 1019–1032. https://doi.org/10.14778/3447689.3447705

[58] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. 2013. Processing Analytical Queries over Encrypted Data. *Proceedings of the VLDB Endowment* 6, 5 (March 2013), 289–300. https://doi.org/10.14778/2535573.2488336

[59] European Union. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the Europeran Union* 59 (2016), 1–88.

[60] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. 2021. CacheOut: Leaking data on Intel CPUs via cache evictions. In *Proceedings of the 2021 IEEE Symposium on Security and Privacy (S&P '21)*. IEEE Computer Society, USA, 339–354.

[61] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. In *Proceedings of the 2017 International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1041–1052. https://doi.org/10.1145/3035918.3056101

[62] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. 2019. StealthDB: a Scalable Encrypted Database with Full SQL Query Support. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019), 370–388.

[63] Mohammad H Yarmand, Kamran Sartipi, and Douglas G Down. 2013. Behavior-based access control for distributed healthcare systems. *Journal of Computer Security* 21, 1 (2013), 1–39.

[64] Zheguang Zhao, Seny Kamara, Tarik Moataz, and Stan Zdonik. 2021. Encrypted Databases: From Theory to Systems. In *11th Conference on Innovative Data Systems Research (CIDR '21)*.

[65] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA, USA) *(NSDI '17)*. USENIX Association, USA, 283–298.

[66] Jingyu Zhou, Meng Xu, Alexander Shraer, Bala Namasivayam, Alex Miller, Evan Tschannen, Steve Atherton, Andrew J. Beamon, Rusty Sears, John Leach, Dave Rosenthal, Xin Dong, Will Wilson, Ben Collins, David Scherer, Alec Grieser, Young Liu, Alvin Moore, Bhaskar Muppana, Xiaoge Su, and Vishesh Yadav. 2021. FoundationDB: A Distributed Unbundled Transactional Key Value Store. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2653–2666. https://doi.org/10.1145/3448016.3457559