SecuDB: An In-enclave Privacy-preserving and Tamper-resistant Relational Database

Xinying Yang
ByteDance Inc
xinying.yang@bytedance.com

Cong Yue
National University of Singapore
yuecong@comp.nus.edu.sg

Wenhui Zhang
ByteDance Inc
wenhui.zhang@bytedance.com

Yang Liu
ByteDance Inc
liuyang.007@bytedance.com

Beng Chin Ooi
National University of Singapore
ooibc@comp.nus.edu.sg

Jianjun Chen
ByteDance Inc
jianjun.chen@bytedance.com

ABSTRACT

With the escalation in the demand for privacy-preserving and tamper-resistant data management and processing on the public cloud, an increasing number of mainstream databases start to provide always-encrypted and blockchain-like features, including Microsoft SQL Server, MongoDB, and Alibaba PolarDB. The recent progress in Trusted Execution Environment (TEE) technology has enabled the deployment of the complete database engine within TEE. This implementation ensures that data stored in memory, cache, and registers is encrypted, thereby maintaining the confidentiality of information. In this paper, we present SecuDB, a multi-granularity privacy-preserving and tamper-resistant relational database by placing the entire RDBMS in Intel TDX. We propose a novel visibility control mechanism incorporating column masking, log masking, and statistics masking to realize fine-grained privacy preservation and devise an isolated TEE-endorsed temporal table method to support efficient data and query verifiability, without affecting insertion and selection performance. We evaluate SecuDB using Sysbench, TPC-C and TikTok copyright workloads. The results show that compared with a system without an enclave, SecuDB hits 84.7% and 94.7% of the performance when providing coarse-grained and fine-grained privacy preservation, respectively. While the overhead for tamper-resistance is less than 22.6%.

PVLDB Reference Format:
Xinying Yang, Cong Yue, Wenhui Zhang, Yang Liu, Beng Chin Ooi, and Jianjun Chen. SecuDB: An In-enclave Privacy-preserving and Tamper-resistant Relational Database. PVLDB, 17(12): 3906-3919, 2024.
doi:10.14778/3685800.3685815

1 INTRODUCTION

With the migration of sensitive data to the cloud, businesses, institutions, and individuals have endured higher risks associated with data breaches, unauthorized access, and other security threats. Consequently, the focus on both data security and privacy in the cloud has become paramount. For example, enterprises are digitizing their business documents using a database system. It is essential to ensure the contents of confidential business documents are kept secret and cannot be revealed to unauthorized users, attackers, and database administrators. Moreover, the system must ensure the documents are authentic and stored correctly.

Many database systems have been developed by mainstream database service providers with the primary goal of safeguarding data security and preserving data privacy. On the one hand, verifiable databases [3, 7, 75, 77–79] are built to ensure the integrity of the data content. These systems often leverage cryptographic functions and authenticated data structures to summarize database states into a digest. The users can perform client-side verification with the digest and proofs generated by the server to verify that the data has not been tampered with. Moreover, systems can support server-side verification with secure hardware to eliminate the network transmission of proofs and relieve the burden of clients [5, 9, 65, 76]. However, these systems do not protect data privacy. On the other hand, encrypted databases are developed to ensure data privacy [2, 17, 43, 53]. In these systems, the clients employ encryption algorithms such as the Advanced Encryption Standard (AES) [22] to encrypt data before transmitting it to the server and decrypting it upon retrieval. The cryptographic algorithm ensures data privacy and security, but the notable challenge limits its utilization in encrypted databases due to its performance bottleneck and deficient functionality for ubiquitous cipher data-based computation; examples include fully homomorphic encryption (FHE) techniques [16, 28], partially homomorphic encryption [46], and property-preserving encryption methods [10, 54]. Furthermore, systems [2, 4, 56, 71] employ secure hardware to support more operations on encrypted data within the enclave. Once a ciphertext is delivered to the enclave, it first decrypts the data, computes on the plaintext, and then encrypts the data before replying to DBMS. We call this architecture as partially hardware encrypted (P-HE). These systems are designed with the assumption of a limited Enclave Page Cache (EPC) size, making it impractical to run the entire DBMS within the enclave. Consequently, they suffer from significant I/O costs between the enclave and the DBMS. Besides, the DBMS run outside the enclave can be compromised, and therefore, data security cannot be guaranteed.

To address the limitations of existing designs and inspired by recent advancements in trusted execution environment technologies [15, 26, 27, 36, 38, 63, 66, 80], such as Intel TDX, AMD SEV-SNP, and ARM CCA, we have conceived and implemented SecuDB. It stands as an in-enclave relational database utilizing full hardware encryption (F-HE), to enable privacy-preserving and verifiable functionalities. This endeavor involves migrating the entire database

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685815

management system (DBMS) into TEE, thus reshaping the prevailing paradigm of partial hardware encryption (P-HE). We invent a column mask-enabled visibility control method for select lists, predicates, logs, and statistics, to realize fine-grained privacy preservation, thereby eliminating the need for conventional client-side encryption and decryption solutions. We isolate TEE endorsement from general queries such as insertion and selection, which significantly outperforms conventional P-HE, and support both data and query-oriented verification using a native temporal table.

SecuDB deploys the entire VeDB [73, 76] engine into TDX with efficient attestation and optimized system tuning. It is deployed in TikTok [13] and Lark [12] to support privacy-preserving and tamper-resistant management of the copyright and grounding, respectively (detailed in Section 3.1). We conduct detailed performance evaluation on SecuDB using two synthesis workloads, Sysbench and TPC-C, and a real-world workload consists of TikTok copyright. We summarize our main contributions as follows:

- We design and develop SecuDB, which, to the best of our knowledge, is the first in-enclave DMBS that serves both multi-granularity privacy-preserving and tamper-resistant functionalities.
- We comprehensively analyze the threat model and database kernel design principle for in-enclave privacy-preserving and tamper-resistant databases.
- We design a novel framework using column mask-enabled visibility control to realize efficient fine-grained privacy preservation in the F-HE system, which reforms client-side encryption in conventional P-HE systems.
- We enhance security and accountability within the isolated environment by extending the attestation and trust chain establishment processes between the Trust Domain (TD) and the operating software components.
- We devise TEE-endorsed advanced temporal tables to support efficient data and query verifiability, significantly enhancing the serviceability of conventional verifiable databases.
- We conduct experiments to evaluate the performance of SecuDB. The experimental results show that SecuDB achieves 84.7% and 94.7% of the baseline throughput when providing coarse-grained and fine-grained privacy preservation, respectively, and incurs 22.6% for tamper-resistance functionalities.

The rest of this paper is organized as follows. Section 2 introduces the background of TEE and databases in TEE. Section 3 outlines system architecture with table and SQL enhancement of SecuDB. Section 4 presents our design of attestation and trust chain in TDX. Section 5 discusses fine-grained privacy-preserving in SecuDB realized by visibility control. Section 6 describes the tamper-resistant framework in our system. We then provide our experimental evaluation in Section 7 before we conclude in Section 8.

2 BACKGROUND

In this section, we first introduce the general principles of Trusted Execution Environment (TEE) and functions in Intel Trust Domain Extensions (TDX) and then present privacy-preserving and tamper-resistant database-related works.

2.1 Trusted Execution Environment

A Trusted Execution Environment (TEE) offers an isolated environment (i.e., secure enclave) for protecting data-in-use. TEE safeguards data and computations against threats, including from a malicious host kernel or hypervisor. TEE has been a prominent focus of academic and industrial research for the past two decades [24, 33, 37, 56, 64, 69, 71, 74], based on various TEE hardware platforms, such as Intel Software Guard Extensions (Intel® SGX) [26, 40], Intel Trust Domain Extensions (Intel TDX) [15, 27], ARM Trustzone [52], Arm Confidential Compute Architecture (ARM CCA) [80], AMD Secure Encrypted Virtualization (SEV) [34], AMD Secure Encrypted Virtualization and Secure Nested Paging (AMD SEV-SNP) [63] and RISC-V Keystone [37].

In general, different hardware supports different levels of isolation. For example, Intel SGX [26, 40, 69] and ARM Trustzone [52] provide application code and data isolation through Enclaves and Secure World, respectively. Intel TDX [15, 27], ARM CCA [80], AMD SEV [34] and AMD SEV-SNP [63] provides secure execution environments that are completely opaque to privileged, untrusted system software such as host OSes and hypervisors.

SecuDB relies on Intel Trusted Domain Extension to provide a TEE-based virtual machine (VM) environment, which provides execution domain isolation by encryption of memory and registers, integrity measurement, and remote attestation to ensure data confidentiality. Intel TDX VM instances, unlike Intel SGX, do not require additional development of a library OS to support application workloads, thereby conserving engineering resources. Moreover, Intel TDX VMs have the ability to fully utilize all CPU and memory resources available on a physical node. This advantage facilitates the management of large-memory workloads entirely within secure memory, minimizing I/O operations and boosting performance significantly [15, 27, 59].

2.2 Intel Trust Domain Extensions

Intel's TDX [15, 27] provides isolation, confidentiality, and integrity at the VM level. It fortifies the confidentiality of guest VMs against the host system and physical security threats. This is achieved through the isolation of guest register states and the encryption of guest memory via secure page management. TDX module operates in a privileged mode, acting as an intermediary between the host and guest environments to oversee the separation between the two.

Intel TDX delivers two major functionalities. First, it ensures the confidentiality and integrity of memory and CPU states, safeguarding sensitive intellectual property and workload data against threats from the host OS. Secondly, it enables remote attestation, allowing a verifying entity, be it the workload's proprietor or a user of the workload's services, to ascertain that the workload is operational on an Intel-TDX-enabled platform within a trusted domain (TD) before sharing any workload-related data.

Memory Encryption by TDX. TDX introduces a new CPU operating mode and utilizes memory encryption techniques to ensure isolation between two VMs. These VMs are encrypted using different keys, directly managed by the TDX Module [15]. TDX employs two complementary technical mechanisms: the Secure Arbitration Mode (SEAM) CPU mode and the Multi-key Total-Memory Encryption (MKTME) [58].

The SEAM mode employs a set of new instructions such as SEAMCALL, SEAMRET, SEAMOPS, and TDCALL to facilitate interaction between the Trust Domain operating system and the host/VMM [57]. Simultaneously, TDX offers protected physical memory regions for safeguarding the code of the Intel TDX module. Multi-key Total-Memory Encryption (MKTME) [58] engine employs the PCONFIG instruction to allocate and configure memory encryption keys for individual Trust Domain VMs. TDX divides memory into private and shared segments. Despite external hardening being necessary under TDX, shared memory remains used during this process. This shared memory is externally readable, as certain virtualized or semi-virtualized devices require external communication, necessitating the host operating system's ability to access this memory to provide services such as networking.

Attestation of TDX. The attestation process in TDX is designed to provide remote parties with verifiable proof of the TD Guest's security and trustworthiness. By leveraging cryptographic techniques, measurements, and secure hardware, TDX attestation enhances the overall security of virtual environments [15, 57, 58]. The attestation process encompasses two fundamental phases: TDREPORT generation and Quote generation.

The initial step within the TDX guest environment entails invoking TDCALL[TDG.MR.REPORT] to procure the TDREPORT from the TDX module. The TDREPORT represents a predetermined data structure, a product of the TDX module, containing guest-specific information encompassing build and boot measurements, platform security version, and a Message Authentication Code (MAC) designed to safeguard the TDREPORT's integrity. A user-provided 64-Byte REPORTDATA [58] is incorporated into the TDREPORT, typically serving as a nonce supplied by the attestation service to enable unique verification of the TDREPORT.

Upon obtaining the TDREPORT, the second phase of the attestation process involves transmitting it to the Quoting Enclave (QE) to generate the Quote. To facilitate remote verification of the TDREPORT, TDX leverages the Intel SGX Quoting Enclave, which verifies the TDREPORT locally and transforms it into a Quote. Subsequently, the Quote is dispatched for remote attestation, where the remote attestation entity parses the Quote and verifies whether the integrity of the boot chain has been compromised.

SecuDB resides within a Trust Domain VM to guarantee the confidentiality of its code and data, and employs integrity measurement and remote attestation mechanisms to verify the integrity of its code and execution environment.

2.3 Trusted and Privacy-preserving Databases

Existing trusted and privacy-preserving databases can be classified into four categories, namely, hardware-enabled encrypted database (H-EDB), software-oriented encrypted database (S-EDB), hardware-enabled verifiable database (H-VDB), and software-oriented verifiable database(S-VDB).

S-EDB systems. Software-enabled algorithms leveraging Property preserving encryption (PPE) to realize fundamental privacy-preserving functionality in database systems [29, 48, 53, 70]. They utilize ciphertext operations to offer ciphertext-based computation. AWS Aurora [62], Microsoft SQL Server [55], MongoDB [43], Alibaba PolarDB [17] all support S-EDB features by client-side column data encryption in their systems. However, besides the high CPU cost of the ciphertext operation complexity, the operations only support a limited scope of DBMS operations.

H-EDB systems. Most hardware-enabled encrypted databases [4, 8, 71] utilize TEE to support secured privacy-preserving features. Microsoft SQL Server [2] and StealthDB [71] are typical P-HE databases that put some supported SQL ciphertext predicates into enclaves. Current commercialized H-EDB systems [2, 17] only support simple operations such as equal, join, like operations [18], which are still far from general and broad SQL operations and functions. Furthermore, P-HE databases cannot support coarse-grained encrypted applications. EnclaveDB [56] puts the entire SQL kernel into the TEE to support stronger security functionality, but it still faces the discussed problems in P-HE database systems.

S-VDB systems. Software-oriented verifiable databases often adopt verifiable computing or authenticated data structures(ADS) to verify the integrity of data. Ledger databases have emerged in recent years [25, 39, 61], to offer blockchain-like tamper-resistant and non-repudiation functionalities in a centralized paradigm. QLDB [7] implements a tamper-evident feature by utilizing ADS on the immutable table composed of current and historical tables. LedgerDB [77] devises a TSA-involved two-way peg protocol to support external auditability. Microsoft SQL Ledger [3] and Oracle blockchain table [47] develop tamper-resistant features on their relational database products.

H-VDB systems. Hardware-enabled verifiable database systems often incorporate the verification of the proof inside the TEE, and therefore, enable server-side verification to relieve the burden of clients and reduce communication costs. Concerto [5] stores incremental hashes of readset and writeset in Intel SGX to verify the integrity of data using a verified memory approach. VeritasDB [65] protects the data with a Merkle B-tree, and uses Intel SGX to perform the integrity checks based on the Merkle B-tree. VeDB [76] utilizes a TEE-native monotonic counter and trusted timestamp as a verifiable endorsement. TrustedDB [9] uses TEE to perform operations, whose trusted zone involves large portions of the DBMS engine. Corda [30] and Hyperledger Fabric [1] implement TEE-assisted oracles as tamper-proofs in a decentralized permissioned blockchain framework.

3 OVERVIEW

In this section, we present an overview of SecuDB, an in-enclave relational database that supports coarse-grained and fine-grained privacy-preserving and tamper-resistant features. We first discuss the use cases, threat model, and design methodology of SecuDB, followed by presenting the system architecture and data model enhancements.

3.1 Use Cases

SecuDB is designed to efficiently support the data privacy and integrity requirements in the increasingly prevalent outsourced collaboration and cloud services. The preservation of privacy can be fine-grained or coarse-grained based on the granularity of protection. While data integrity can be further categorized into data-oriented and query-oriented based on the target of protection.

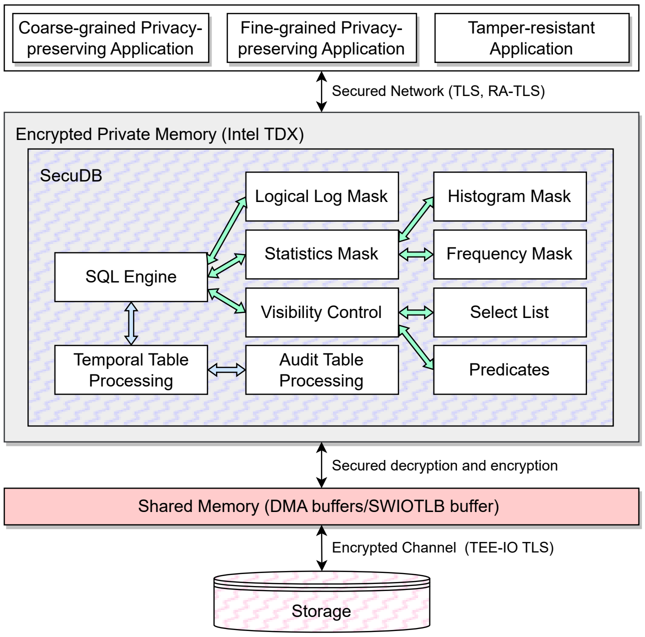

Coarse-grained Privacy-preserving Application
Fine-grained Privacy-preserving Application
Tamper-resistant Application
Secured Network (TLS, RA-TLS)
Encrypted Private Memory (Intel TDX)
SecuDB
Logical Log Mask
Histogram Mask
Statistics Mask
Frequency Mask
SQL Engine
Visibility Control
Select List
Temporal Table Processing
Audit Table Processing
Predicates
Secured decryption and encryption
Shared Memory (DMA buffers/SWIOTLB buffer)
Encrypted Channel (TEE-IO TLS)
Storage

Figure 1: System Architecture of SecuDB. Coarse-grained privacy-preserving is realized by the native in-enclave architecture. Fine-grained privacy-preserving is designed by visibility control and data masking of columns, logs, and statistics. Tamper-resistance leverages an advanced temporal table paired with an audit table practically.

Coarse-grained privacy-preserving. The coarse-grained privacy-preserving method protects all data from unauthorized access. A typical customer requirement for coarse-grained privacy-preserving is the Volcengine [11, 73] (ByteDance public cloud platform) database product. An in-enclave database natively supports the coarse-grained privacy-preserving use case.

Fine-grained privacy-preserving. To fulfill flexible data privacy demands in collaboration, the fined-grained approach is utilized to protect privacy at the column level. An example of fine-grained privacy-preserving customer cases is the employee data table containing sensitive information such as salary that is stored in Lark [12]. The DBMS has to guarantee that no users other than human resource roles, including database administrators, can view the contents. Another customer case is Bytedance information system platform, which builds regulation-compliance infrastructure to support multi-level sensitive column data that is realized by SecuDB fine-grained privacy-preserving feature.

Data-oriented tamper-resistance. Data verifiability is another requirement of database customers. It is useful for a database service provider to prove a certain piece of data is never tampered with (blockchain-like) to its user or an external auditor, especially for mission-critical cases. For example, TikTok [13] leverages the tamper-resistant techniques in SecuDB to notarize the copyright authorship and royalty trail of the artworks on its platform. Another use case is Bytedance digital signature service, which can adopt SecuDB to realize the tamper-resistant functionality with easier deployment and better performance compared to their previous permissioned blockchain solution.

Query-oriented tamper-resistance. Query-oriented tamper-resistance refers to a verifiable query result set. Tamper-proof is expected to verify the result set generated by the database engine has never been tampered with. For example, Volcengine [11] users may not trust the results retrieved from a third-party outsourced database. To facilitate trust, the cloud database service provider needs to prove its integrity.

3.2 Threat Model

In this paper, we assume a powerful adversary who possesses control over the entire software stack on the database server, except for the code encapsulated within secure enclaves. This adversary encompasses various potential threats, including untrusted system administrators, database administrators, and external attackers who may access and tamper with server-side data residing in memory, on disk, and during network transmissions. In this context, the system must guarantee that the transactions and queries issued by the clients are executed correctly. In the meanwhile, sensitive data and information cannot be revealed or deduced by any unauthorized operation whenever the data is in use or at rest. In addition, any authorized modification to the data should be non-repudiated. In case of a malicious action conducted by authorized users or database administrators, the system can always identify the malicious user. In this way, the system protects the data against insider threats. However, denial-of-service and side-channel attacks are outside of the scope of the paper.

3.3 Design Methodology

3.3.1 *Privacy-preserving is simplified to visibility control.*

Conventional H-EDB and S-EDB systems utilized an end-to-end encryption methodology to realize database column privacy. In particular, a P-HE database attested supported operations, e.g., equal predicate, in the enclave, while most DBMS engines run outside the Trusted Computing Base (TCB). Thus, the end-to-end encryption method is unavoidable to keep the data as secrets. However, we discover that in a F-HE architecture, the client-side encryption is no more a constraint since TCB covers the entire DBMS. An attacker can not obtain the data in any memory, CPU, or I/O phase as in our threat model analysis. Hence, column-level privacy preservation is simplified as preventing unauthorized users from viewing the data. We present a new catalog table to store secret data ownership and visibility. In addition, we implement access control for all appearances of data that can be observed from a client, e.g., through the select list, predicates, logical logs, statistics, etc (Section 5). Unlike conventional database column-level privilege controls [32, 42, 50, 60] which allow highly privileged roles such as DBA to access user data, SecuDB introduces an additional visibility control method for secret columns, to disallow any retrieval by a highly privileged user who has access right.

3.3.2 *Verifiability is simplified to TEE-endorsed temporal table.*

S-VDB systems utilize temporal or immutable tables [23, 41, 49], whose records are computed into a representative digest and built with provable ADS, for verifying an entity indeed exists in a set. H-VDB, however, does not need to build such entangled verifiable

Table 1: SQL Enhancement in SecuDB, where sn is the serial number in temporal table, ? is the parameter passed to the SQL statement [44], c1 and c2 are user defined columns, and user1 is the ID of the user.

Features Support	Data Operations	SQL and Procedures
Fine-grained privacy-preserving	Define a fine-grained privacy-preserving column	CREATE TABLE T(c1 INT SECRET, c2 VARCHAR);
Fine-grained privacy-preserving	Control column visibility to another user	GRANT/DENY/REVOKE VIEWER ON schema.table.column TO user1;
Fine-grained privacy-preserving	Transfer visibility ownership to another user	GRANT OWNER schema.table.column TO user1;
Fine-grained privacy-preserving	Undo unexpected ownership transfers	REVOKE OWNER schema.table.column;
Data-oriented tamper-resistance	Define data-oriented tamper-resistant table	CREATE TABLE T(c1 INT) IMMUTABLE YES WITH TYPE TEE;
Data and query tamper-resistance	Verify temporal data or query resultset	SELECT FROM T WITH SIGNATURE;
Data-oriented tamper-resistance	Erase obsolete data in advanced temporal table	DELETE FROM T WHERE sn <= ?;
Data-oriented tamper-resistance	Hide violated data in advanced temporal table	DELETE FROM T WHERE sn = ?;

data structures. In conventional H-VDB systems with a P-HE architecture [76], auxiliary TEE parameters have to be involved such as monotonic counter, trusted timestamp [19] to support auditability, and TEE-endorsed data has also to be recorded into DBMS. Nevertheless, we find that no auxiliary TEE parameters are needed in a F-HE database, even the TEE-endorsed signature recording, due to the F-HE TCB. Therefore, the verification of data integrity is simplified by providing verification API, which executes select statements on the temporal table and returns the TEE-endorsed signature with the resultset (detailed in Section 6). It supports both data- and query-oriented verification, which none of the existing database systems support.

3.4 SQL Enhancement

We enhance SQL syntax for privacy-preserving and tamper-resistant functionalities of in-enclave databases, including DDL, DCL, and DML as presented in Table 1.

3.4.1 Privacy-preserving SQL Syntax. When creating or altering a table, a privacy-preserving column is defined with an additional keyword `SECRET`. It will grant the user who can see the plaintext as the owner of the secret column. All other users cannot observe the plaintext in any way, such as data retrieval, predicate handling, log probing, or statistic viewing. The owner can grant column visibility by `GRANT VIEWER` DCL to another user, explicitly block secret column visibility by `DENY VIWER`, and remove the visibility by `REVOKE VIWER` to a previously granted user. All the above DCL operations can be only executed by the secret column owner, to prevent unexpected operations from high-privileged roles such as DBA. An owner can transfer the ownership to another user. This statement will downgrade the original user as a viewer automatically. Note that there is only one user holding the owner role at one time of fine-grained column-level privacy. If the owner transfers ownership to another user by mistake. It can be reverted by highly privileged users using `REVOKE OWNER` DCL relied on a system-maintained historical ownership chain that records all historical ownership transfers. This statement provides a safe mechanism to undo unexpected ownership transfers. To prevent secret data leaking by spoofing attacks, we deactivate secret owner or viewer password reset by high-privileged roles such as DBA.

3.4.2 Tamper-resistant SQL Statement. A TEE-assisted tamper-resistant table in SecuDB can be defined using `IMMUTABLE` keyword with `TEE` specified. It creates an advanced temporal table containing implicit column *SN* discussed in Section 3.6.2. The table data and query resultset can be verified by specifying an additional suffix `WITH SIGNATURE` in the selection. To achieve practical regulatory compliant and overcome storage overhead (detailed in Section 6), we offer a constraint predicate on implicit column *SN* to locate the target data. The equal predicate is used to hide the violated information, while the no greater than predicate is used to erase consecutively historical data to save storage. Note that only high-privileged roles can execute this predicate on a delete statement.

3.5 System Architecture

SecuDB deploys the entire VeDB [76] engine, a relational database with trusted features on ByteDance public cloud platform called Volcano Engine [11, 14], into Intel TDX by provable attestation. All database internal computations are securely protected within the enclave, and data storage is protected by TEE encryption before any disk write as depicted in Figure 1. This provides coarse-grained privacy-preserving capability which is transparent to an end user.

To support fine-grained privacy-preserving, SecuDB creates a visibility control catalog table to process related DML predicate and select list. Logical logs and statistics on the secret columns will be masked when retrieved by an end user who does not have permission to view the column, while the log and statistics database engine processing and data all remain unchanged. Verifiability in SecuDB covers both data-oriented tamper-resistance and query resultset authentication, which can be verified with a TEE-endorsed signature. A tamper-resistant table is realized by an advanced temporal table with a related audit table.

3.6 Table Enhancement

3.6.1 Catalog Table. We introduce a new catalog table named *sec-cat*, to support fine-grained privacy-preserving functionality on a secret column. *sec-cat* does not change existing SQL privileges, but introduces an additional visibility control method for secret columns to bar privileged users from viewing the data. *sec-cat* has three columns: *ColID* (the unique ID of the column), *Owner* (the owner of the table), and *Viewer* (users who can see the column data). When a column is defined as a secret column (detailed in Section 3.4.1), a row with the creator as the owner will be inserted into *sec-cat*. An owner transferring or a viewer granting SQL statement will update the *Owner* and *Viewer* column correspondingly. *sec-cat* is used by SQL parser and runtime when executing related DML statements (detailed in Section 5.2).

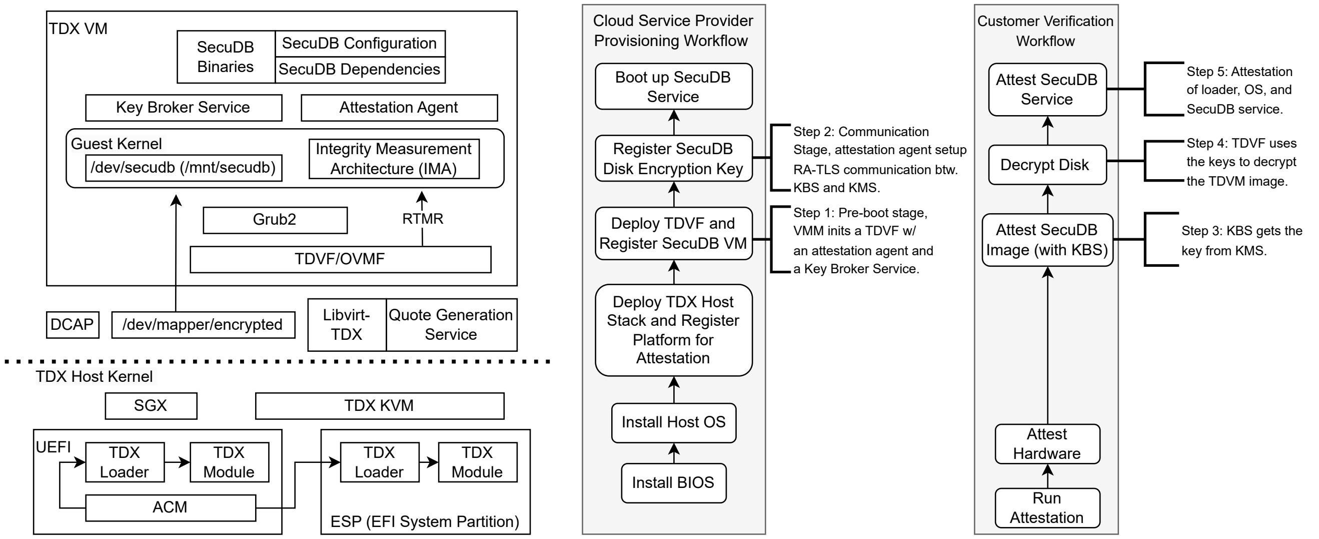

TDX VM
SecuDB Binaries
SecuDB Configuration
SecuDB Dependencies
Key Broker Service
Attestation Agent
Guest Kernel
/dev/secudb (/mnt/secudb)
Integrity Measurement Architecture (IMA)
Grub2
RTMR
TDVF/OVMF
DCAP
/dev/mapper/encrypted
Libvirt-TDX
Quote Generation Service
TDX Host Kernel
SGX
TDX KVM
UEFI
TDX Loader
TDX Module
ACM
TDX Loader
TDX Module
ESP (EFI System Partition)
Cloud Service Provider Provisioning Workflow
Boot up SecuDB Service
Register SecuDB Disk Encryption Key
Deploy TDVF and Register SecuDB VM
Deploy TDX Host Stack and Register Platform for Attestation
Install Host OS
Install BIOS
Step 2: Communication Stage, attestation agent setup RA-TLS communication btw. KBS and KMS.
Step 1: Pre-boot stage, VMM inits a TDVF w/ an attestation agent and a Key Broker Service.
Customer Verification Workflow
Attest SecuDB Service
Decrypt Disk
Attest SecuDB Image (with KBS)
Attest Hardware
Run Attestation
Step 5: Attestation of loader, OS, and SecuDB service.
Step 4: TDVF uses the keys to decrypt the TDVM image.
Step 3: KBS gets the key from KMS.

Figure 2: Provisioning and Verification Workflow of SecuDB. Three stages of provisioning and verification, attestation of hardware, attestation of SecuDB VM image and secondary drives, and attestation of SecuDB.

3.6.2 Temporal Table. To realize the tamper-resistant feature in SecuDB, we present an advanced temporal table (aTT) that meets the regulation-compliant requirement and resolves the storage overhead of immutable tables (detailed in Section 6.2). An original temporal table [23] is a kind of immutable table composed of a current table that stores current data, and a historical table containing all the obsolete data. In an in-enclave architecture, a TEE-endorsed temporal table is natively tamper-resistant in the discussed threat model. In addition, we create a new implicit column called *SN*, whose contents are filled with serial numbers. The serial numbers are used to locate data to be erased or hidden for audit and verification (detailed in Section 3.4.2).

3.6.3 Audit Table. An audit table is pairwise with aTT, and records the metadata for all the erased or hidden data trails in aTT, to keep aTT auditable. It stores the executor of the operation, the operation type, the specified value of *SN*, the number of rows affected, and the timestamp. The audit table is only created after an erasing or hiding operation is executed to an aTT.

4 ATTESTATION OF SECUDB SERVICE

In this section, we introduce the trust model for SecuDB attestation, and then present attestation of SecuDB.

4.1 Trust Model for SecuDB Attestation

Attestation evaluates the trustworthiness of the TDX guest when it comes to external entities seeking access to sensitive information. In the case of SecuDB, the key server initiates an attestation process before granting authorization to release encryption keys, which is required for mounting the encrypted rootfs and secondary drives in full disk encryption mode. Once the encrypted rootfs or secondary drive is successfully mounted, a further step is taken to guarantee that SecuDB functions within a memory-encrypted environment by incorporating its runtime and dependencies into the attestation chain. Within this trust chain, several key objectives are accomplished:

- Verification of the secure container's environment to ensure it runs on genuine Hardware-based TEE.
- Validation that the disk partition is trusted and no data leakages happen.
- Validation that the virtual machine operates with third-party programs as expected, ensuring expected behavior.
- Confirmation that the runtime of SecuDB remains unaltered, preserving its integrity.

This trust chain establishes a robust and secure foundation for deploying and running SecuDB, assuring customers of the integrity and authenticity of their computing environment.

4.2 Attestation of SecuDB

SecuDB aims to safeguard data confidentiality at rest and at runtime. Confidentiality of data at rest is accomplished by encrypting the SecuDB runtime and its dependencies, including virtual machine (VM) disk image, SecuDB binary, SecuDB configuration files, and dependencies. Meanwhile, the confidentiality of data at runtime is achieved through memory encryption within the Trusted Execution Environment (i.e., a Trusted Domain Instance). We also preserve the confidentiality of data at rest capability through storage encryption using a trusted Key Broker Service (KBS), shown in Figure 2. Two agents accomplish attestation of SecuDB: the Attestation Agent and the Key Broker Service (KBS).

The attestation process comprises two steps: the generation of TDREPORT and the subsequent generation of a Quote.

Attestation Agent. In the early OS boot attestation phase, the attestation agent is located in the initrd (initial RAM disk), as part of the TDVF. During the OS runtime attestation phase, the attestation agent functions as an application within the operating system in its userspace. Its main tasks include obtaining the storage volume key from a remote key server and passing it to the KBS, triggering the generation of TDREPORT, and getting the Quote for attestation.

Key Broker Service. The KBS resides within trusted TDVF. The primary role of KBS is to receive the storage volume keys from the attestation agent and utilize them to decrypt the storage volume, ensuring that both rootfs and disk partitions are accessible and secure. During the rootfs loading phase, KBS undertakes the decryption of the disk image. The TDVF can load and initiate the operating system's launch from the decrypted partition. In instances involving encrypted secondary volumes encountered in scenarios like data partition or container image encryption, the KBS decrypts the storage volumes after the system has been successfully booted.

SecuDB adheres to Intel's official guidelines on attestation to conduct hardware attestation [20], followed by performing full disk encryption [21] for attestation of secondary storage volumes. VM attestation and the attestation of SecuDB are detailed as follows.

Attestation of Virtual Machine. Attestation of Virtual Machine ensures guest OS's confidentiality by measuring and attesting the rootfs. The TDX KMS manages the key for disk encryption to ensure key security. We encrypt the disk image containing the SecuDB binaries and its dependencies. This ensures that even if an unauthorized entity gains access to the physical storage, it cannot disclose the data in a disk image without the decryption key.

The process begins with the VMM receiving the customer's encrypted disk image and consists of 5 major steps. In step 1, in the pre-boot stage of TDVM instances, the VMM inits a TDVF containing an attestation agent and a KBS. In step 2, within TDVF, an attestation agent is launched to facilitate communication with a remote key server. The attestation agent, operating under the TDVF's secure environment, establishes a trusted and authenticated connection with the remote key server using the protocols of Remote Attestation with Transport Layer Security (RA-TLS). In step 3, once the secure session is established, the attestation agent communities with KMS and gets the keys. The attestation agent acquires the essential disk encryption key from the remote key server and then efficiently passes this key to the KBS. In step 4, the TDVF uses the keys to decrypt the TDVM image. In step 5, after successful decryption, TDVF locates the Operating System (OS) loader and initiates the boot process, ensuring the system boots into the intended OS. The OS loader loads the OS kernel and kernel-required data, such as boot parameters, into private memory. The OS loader measures the kernel and required data, including the boot parameter, before passing the control to the OS kernel. During this step, the Integrity Measurement Architecture (IMA) module is measured for its integrity as part of the guest operating system.

Attestation of SecuDB. As depicted in TDVM attestation, the integrity of the IMA module is an integral part of the attestation chain for the rootfs disk. Thus, IMA is trusted. We employ the IMA to gauge the integrity of files within the SecuDB engine, accessed through system calls like execve(), mmap(), and open, all guided by customized policies. These files undergo a thorough assessment, covering both binary and configuration dependencies. We use IMA appraisal to control access to SecuDB and its dependencies by comparing the reference value of file integrity measurements with their standard reference values stored as security extension attributes, where security.ima contains the hash value of file content and security.evm holds the hash value signature of extended file attributes.

To minimize the TCB in the attestation chain, we utilize Linux's List Dynamic Dependencies (e.g., 'ldd') to identify the shared library requisites of executable files at runtime. We establish a comprehensive set of IMA policies for attestation to safeguard the integrity and security of both SecuDB and its associated components. These policies encompass the use of 'ima_hash' utilizing the SHA384 algorithm [6], 'ima_appraise' configured to log events, 'ima_policy' tailored to address critical data, and the implementation of the 'BPRM_CHECK' measurement function, which takes into account the effective user ID ('euid') specific to the SecuDB engine. This multifaceted approach to security and integrity ensures robust protection for the SecuDB binary, its configurations, and the requisite dependency libraries.

5 PRIVACY-PRESERVING IN SECUDB

In this section, we first describe the visibility definition in an in-enclave architecture, and then present our design of visibility control on predicates, select lists, logs, and statistics.

5.1 Visibility Definition

In a F-HE database architecture, the mechanism protects all memory, CPU, and I/O security from data leaks in our threat model as discussed in Section 3.2. Thus, any DBMS internally used data structures that do not have explicit retrieval interfaces, cannot be viewed by adversaries, such as system and physical logs. To better understand the description, we take the redo log as an example. As a physical log, it stores all changes made to a database in log files. Redo log can be loaded into memory and participated in CPU computation, accessed by disk I/O, and transmitted between replicas through network I/O. None of these operations will leak data in a F-HE paradigm because enclave memory and CPU are protected; data will be encrypted by TEE before written to disk, and network transmission is secured by RA-TLS protocol [35]. Therefore, we define the visibility in the context of in-enclave databases as follows.

Definition 5.1. In the context of in-enclave databases, *visibility* is to protect the secret information from being observed by any exposed DBMS data retrieval interface.

By comprehensively analyzing all exposed interfaces in relational databases, we identify several components that have potential data leaks, namely DML predicates, select lists, regular views, logical logs, frequency and histogram statistics, plan explaining, and therefore must be redesigned.

5.2 DML Visibility Control

We propose a non-encryption involved framework to realize efficient fine-grained privacy-preserving F-HE database as depicted in Figure 3. Both host variables in the privacy-preserving column involved predicates and privacy-preserving columns on the select list avoid encryption and decryption computation in our method, due to the Trusted Computing Base (TCB) in F-HE system. A user defines a secret column via DDL as the owner, which is recorded into a catalog table *sec-cat* discussed in Section 3.6.

When a privacy-preserving column involved SELECT statement is executed, the procedure in SQL bindtime executes as follows:

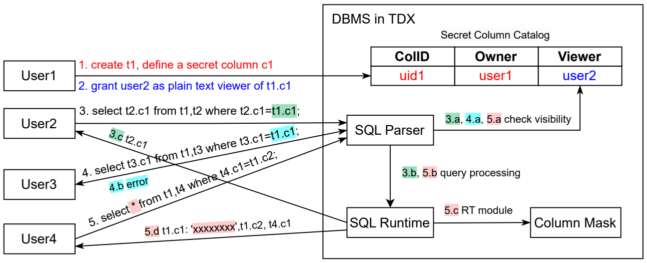

DBMS in TDX
Secret Column Catalog
ColID
Owner
Viewer
uid1
user1
user2
User1
User2
User3
User4
1. create t1, define a secret column c1
2. grant user2 as plain text viewer of t1.c1
3. select t2.c1 from t1,t2 where t2.c1=t1.c1;
3.c t2.c1
4. select t3.c1 from t1,t3 where t3.c1=t1.c1;
4.b error
5. select * from t1,t4 where t4.c1=t1.c2;
5.d t1.c1: 'xxxxxxxx',t1.c2, t4.c1
SQL Parser
3.a, 4.a, 5.a check visibility
3.b, 5.b query processing
SQL Runtime
5.c RT module
Column Mask

Figure 3: Fine-grained privacy-preserving workflow in SecuDB. A secret column catalog table stores information about who can see column plain text. SQL parser and runtime handle predicates and select list accordingly.

- Parser checks the user's visibility of secret columns in predicates. Any invisible predicate will trigger the early out phase, and return SQL error.
- Parser checks the user's visibility for secret columns in the select list. The invisible column will be marked to use column mask data for query runtime processing.
- SQL runtime structure generator identifies invisible columns based on parser data structures, and marks related variables for probing *sec-cat* during execution time.

The relevant query execution goes as follows:

- SQL runtime identifies secret column visibility on select list based on runtime structure during query runtime.
  - If visible, then retrieve data from data manager.
  - If invisible, then fill the column value buffer with the system default masking value for the related data type.
- SQL runtime identifies visibility of secret column in predicates based on runtime structure during query runtime.
  - If visible, then retrieve data from data manager.
  - If invisible, then return SQL error.

Note we do not allow users to alter a general column to a secret column and vice versa. Owner role can be granted and transferred between users by DCL. An owner can grant column visibility to other users as viewers. The DML visibility control does not only control SELECT statements, but also all the querying interfaces and objectives such as regular view querying and plan explaining.

5.3 Logical Log Visibility Control

SecuDB uses a logical log called Auditlog to record all changes made to the database. Auditlogs are created by the SecuDB server and contain a record of all SQL statements that modify data. It serves as an audit trail of changes and can be used for various purposes, such as data recovery, replication, and database monitoring. The content stored in Auditlog can be viewed in a human-friendly format using DML retrieval.

The logical log file in SecuDB is unstructured, the content retrieval can not be parsed and decoded into column-level by DBMS utility. Hence, there is a challenge to build a schemaless encoding to locate the value of a secret column in log statements. To tackle this problem, we assemble the secret column ID, length, and value with a predefined constant as a prefix, which is depicted in Figure 4.

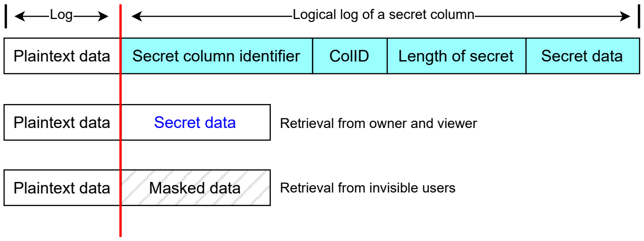


Figure 4: Logical log store for a secret column in SecuDB. Kernel log replay and data retrieval from granted users obtain plaintext after database kernel decryption, while invisible users get data mask values.

The constant is a 256-bit value that cannot be collided and is used as an identifier for data of a secret column.

When an end user tries to retrieve Auditlog, DBMS engine will search for the predefined identifier during log file parsing. The procedure goes as follows:

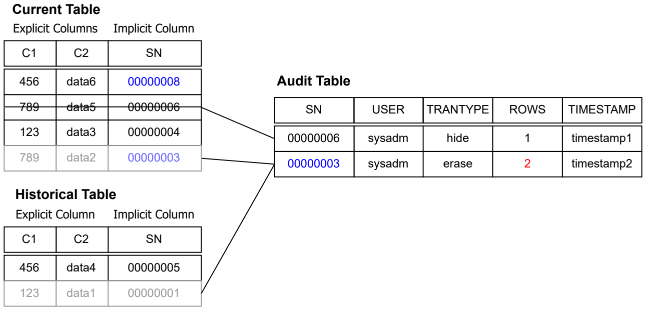
- If the identifier is located, then DBMS marks the beginning offset of the identifier as L_1 and reads the next 16 bytes to probe the secret column ID in *sec-cat* catalog table.
  - If the column is visible to the user, log manager reads the next two bytes to get the length of the secret data, gets the secret data, and marks the end of the secret data as L_2. The log manager will locate the secret data buffer, i.e., from $L_1 + 32 + 16 + 2$ to L_2, and bind it out.
  - If the column is invisible to the user, the log manager uses the same logic above to locate L_2. The related data between L_1 and L_2 will be masked before binding out as depicted in Figure 4.
- If none of the secret columns are identified, the log file processing remains the same as before.

Take an example from Figure 4, assuming the value of the identifier is 0xfffc, the ColID is 0x8001, the secret data is 0x1234. Its recorded data in the log file will be 0xfffc800100021234. Our algorithm locates the secret column when searching identifier. It reads ColID 0x8001 and probe *sec-cat*. It then reads the next two bytes and finds length 0x0002, followed by another two bytes to locate 0x1234. If the user has the privilege to view the data of the secret column, we then get 0x1234 and return its value. Otherwise, we will return masked data.

5.4 Statistics Visibility Control

Statistic information in SecuDB catalog tables is used for SQL optimizer to choose the most efficient access path when executing a query. The statistics include frequency, histogram, cardinality, etc, which can be retrieved by an end user.

Frequency catalog data records several top frequent values in a specific column with the percentage of their occurrence. If a user, who is unauthorized to see a secret column data, tries to retrieve its frequency statistics, SecuDB will mask the column value before replying. Histogram statistics containing data distribution information will also be masked by DBMS kernel before returning to a user who is not permitted to see a secret column. The statistics



Current Table

Explicit Columns		Implicit Column
C1	C2	SN
456	data6	00000008
~~789~~	~~data5~~	~~00000006~~
123	data3	00000004
789	data2	00000003

Historical Table

Explicit Column		Implicit Column
C1	C2	SN
456	data4	00000005
123	data1	00000001

Audit Table

SN	USER	TRANTYPE	ROWS	TIMESTAMP
00000006	sysadm	hide	1	timestamp1
00000003	sysadm	erase	2	timestamp2

Figure 5: Advanced temporal and audit tables in SecuDB. All erased and hided trails are recorded in the audit table. Note that the serial numbers are not monotonically constrained.

stores range value information of a specific column with the range frequency, and is used for access path optimization. Note that the column value masking only affects statistical data retrieval from invisible users. DBMS optimizer always uses the plaintext of all statistics for query planning.

6 TAMPER-RESISTANCE IN SECUDB

In this section, we present tamper-resistance in SecuDB using an advanced temporal table, and discuss its efficient verification endorsed by TEE, which does not affect insert performance.

6.1 Tamper-resistance in General

Tamper-resistance refers to a mechanism to prevent objective tampering and provide verifiable methods to detect any tampering attack. We categorize database tamper-resistance into data-oriented and query-oriented, as discussed in Section 1 when introducing S-VDB systems. Data-oriented tamper-resistance aims to keep an entire data trail withstanding any future update and deletion of historical data. Ledger databases [3, 7, 76, 77] are typical data-oriented tamper-resistant systems, which realize blockchain-like tamper-evidence in a centralized paradigm. Query-oriented tamper-resistance focuses on providing an endorsed and verifiable result set of a query from being tampered with, such as a verifiable result set retrieved from an outsourced database. We realize both data-oriented and query-oriented tamper-resistance in SecuDB.

6.2 Advanced Temporal Table

Advanced temporal table (*aTT*) is our practical design in the in-enclave database systems, to support the data-oriented (blockchain-like) tamper-resistant feature as discussed in Section 3.6.2.

A conventional temporal table [23] is designed as an immutable table containing timestamp information. The table is generally divided into a current table and a historical table, and data can be retrieved by timestamp filters. All historical data, i.e., old data of an update or delete statement, are stored in the historical table. Thus, the table is immutable. However, there are two strong requests to enhance the conventional temporal table design. First, the customer requests to delete data before a certain timestamp, to reduce the storage overhead of the immutable table. Second, real-world applications request violated data to be hidden to meet regulatory compliance such as GDPR [45, 72] and CCPA [51].

We present a new implicit column *SN* whose data type is serial number to resolve the two requirements above and keep the entire table auditable. The *SN* column is designed to be a unique searching key, so it does not have to be monotonic, which would involve significant overhead of transaction processing. To resolve the storage overhead problem, the data can be deleted before a certain timestamp using `DELETE` statement by specifying the target *SN* with a no greater than predicate described in Table 1. To be regulatory compliant, the violated data can be erased by the `DELETE` statement with an equal predicate. All these operations will be recorded in an audit table as depicted in Figure 5.

6.3 Native TEE-endorsed Verification

In an in-enclave architecture, the design of tamper-resistance is entirely different from conventional P-HE systems. Unlike a P-HE tamper-resistant database system, the insertion operation is entirely transparent, meaning any digest computation, TEE trusted parameter combination, and TEE signing is no longer needed, which is more efficient and practical.

Verification is conducted by adding `WITH SIGNATURE` keywords in a select statement as discussed in Section 3.4.2. The result set will be signed by TEE (detailed in Section 6.4), and can be verified on the client side. The endorsement is not only used to verify tamper-resistant data in *aTT*, but data retrieved from common queries in general tables, i.e., the query-oriented result set. It guarantees the data integrity of the retrieved result set endorsed by TEE. The verification interface is controlled by the new suffix keywords on the select statement, so it doesn't affect general selection performance. Note the query-oriented result set verification is a one-time verification, meaning the raw data returned can be only verified within the query. The same query in different or the same transaction can retrieve a different number of records. Even if the contents of the result sets are the same, the content ranking can be different, which makes a previous tamper-proof not reusable.

6.4 Query Result Set Verification

The main challenge of a query result set verification in the architecture of SecuDB lies in how to efficiently build the tamper-proof, and how to form efficient client-side decoding to interpret the raw data encoded at the database kernel (e.g., a server-side schema may encode small int value 1 0x80000001 in an internal format, while the client-side schema encodes it 0x00000001), where the tamper-proof is signed by TEE.

We present a data type assembled serialization method on database data manager (DM) raw data format to resolve these problems. When a user issues a verification DML using `WITH SIGNATURE`, the database kernel will generate and return a data type array composed of all elements of the result set serially, together with the combined DM format raw data that is the input of digest computation. The combined raw data is composed of the flattened hex data by concatenating each row of the result set sequentially. Thus, the entire result set and all the elements, i.e., each column in each row can be verified. The element verification is conducted by retrieving its data based on matching data type and length from the first column onwards to find the designated column. For example, consider a result set with c1 nullable INT (data type 497) and c2 VARCHAR(20) (data type 448), and we want to verify c2. Suppose the current row data to be verified is '*1, Ledger*', which is encoded as 0x00800000010006d38584878599 by EBCDIC CCSID 37 [31]. For the first column, the cursor moves forward for 5 bytes due to the nullable INT data type of c1; for the second, since it is a VARCHAR data type, the length data is read from the raw data (0x0006) to determine the actual buffer size 8 (length itself added). Thus, the data to be verified is located as 'd38584878599'.

7 EVALUATION

This section presents experimental results of coarse-grained privacy-preserving, fine-grained privacy-preserving, and tamper-resistant performance in SecuDB.

7.1 Hardware Configuration

Our experiments are conducted on a ByteDance Volcengine [11, 14] ecs.ebmg3id.48xlarge node. It is equipped with the fourth-generation Intel® Xeon® Scalable Processor (Sapphire Rapids), with a base frequency of 2.6GHz and an all-core turbo frequency of 3.1GHz. We install bare-metal ecs.ebmg3id.48xlarge with Ubuntu 22.04 and Linux version 5.19.0 with a backported TDX patch from Linux version 6.3.0. All tests are conducted on the CPUs of NUMA node0, utilizing vCPU0 through vCPU15, with each VM instance allocated 128GB of RAM and 500GB SSD storage. All VM instances are linked together through 25Gb Ethernet connectivity. Specifically for the *TDX* test, we applied the IMA patch to the guest Linux OS. In the *TDopt* test, we applied patches for both IMA and swiotlb.

7.2 Benchmark and Workload

We evaluate SecuDB using two prevalent datasets Sysbench [67] and TPC-C [68], and one real-world dataset T-bench which consists of data from the TikTok copyright database.

TikTok [13], being a social media platform, allows users to create, edit, discover, and share videos. When an artwork is uploaded, a hash digest is computed as a unique digital DNA, which will be recorded onto a verifiable SecuDB aTT. Artwork producers can claim the ownership, and track the full trail of their video/music's royalty changes to different creators and holders, by tracing all the records related to the related artwork ID on aTT, which is provable by TEE-endorsed signatures. Its database schema contains attributes such as media type (e.g., music, video) and the user details. The average size of each row is around 2KB. The T-bench workload consists of 200 million anonymised video/music records that have been recently uploaded onto TikTok.

We shall now discuss our testing scenarios as follows.

nonTD. We launch the DBMS server on a guest OS with TDX disabled. Subsequently, we conduct an assessment of the performance under this configuration. This setup is established as the baseline against which we measure the overheads incurred by coarse-grained privacy-preserving system performance in TDX.

TDX. In this scenario, we deploy the SecuDB within the TDX VM to evaluate the end-to-end system performance of the coarse-grained privacy-preserving feature under all benchmarks. Since TDX pre-allocates encrypted memory, the NUMA allocation is more sensitive to TD than in non-TD environments. Additionally, as TDX's bottleneck is I/O, we allocated more threads for parallel I/O processing. We further tune the TDX I/O threads, localities parameters to optimize the system performance. We also lock CPU frequency modes for fair comparison. The optimized system is marked as *TDopt*.

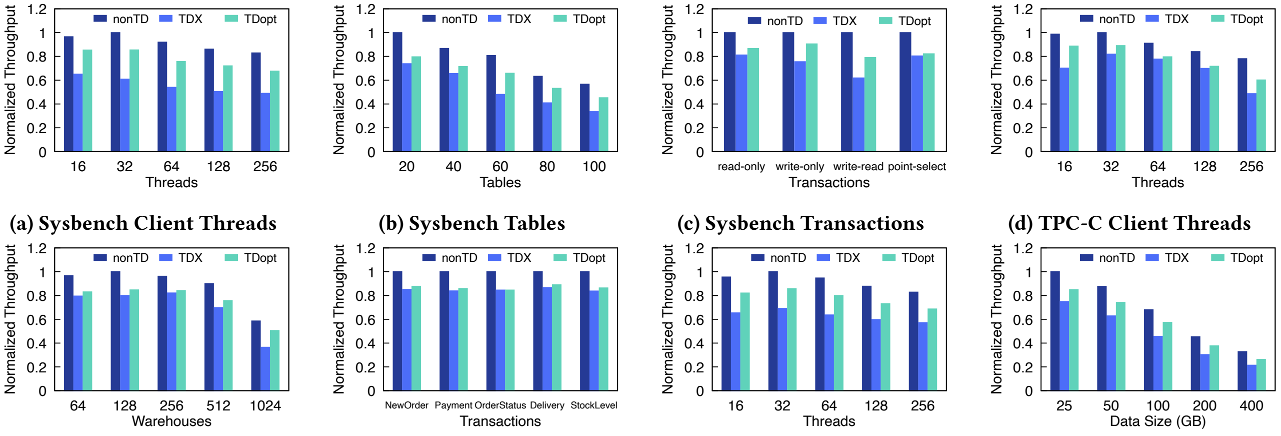
FP. To test end-to-end performance of fine-grained privacy-preserving features, we made minor changes to DDL to define secret columns in Sysbench and T-bench, and grant different column visibility to other users for execution. More specifically, we prefix secret columns on select list as *FP-SEL*. For example, if a select list contains 4 secret columns, it will be specified as *FP-SEL4*.

TR. To evaluate the tamper-resistant feature's end-to-end benchmark performance, we create all the tables in all the benchmarks as aTT, and test system performance in *TDopt* with and without aTT. Specifically, we evaluate the insertion overhead of aTT, then test its selection baseline and verification using selection `WITH SIGNATURE` syntax in all queries.

7.3 Privacy-preserving Experiments

7.3.1 Coarse-grained Privacy-preserving. We place SecuDB within secure enclaves, which enables it with the capabilities of Always-Encrypted confidential queries. The enclave environment, coupled with an encrypted 500GB disk, which is encrypted using LUKS (e.g., dm-crypt), creates a protected realm within the database system process, appearing as an impenetrable entity to both the database system and any other processes running on the hosting machine. Additionally, it safeguards the stored data on disk and isolates it from other processes, ensuring comprehensive protection of the data. According to Intel, TDX environments typically incur overheads in both I/O and memory access due to the added security features. For example, I/O operations can experience an additional 9.3% latency overhead due to encryption/decryption, and memory access patterns might suffer from up to 5% latency overhead due to the isolation enforced by TDX [59]. In our actual measurements, we observed similar trends, with I/O operations experiencing up to 30% overhead and memory access patterns seeing up to 20% overhead in some cases. We identified vsock processing as the bottleneck. This prompted us to raise the I/O thread count from 2 to 8 for efficient disk I/O workload management.

Figure 6a and 6c show coarse-grained privacy-preserving Sysbench throughput testing results by varying client threads and transaction modes using 60 tables with one million rows in each, normalized to the baseline performed in the same database launched without TEE. We observe the increasing throughput with more threads until they peak at twice the vCPU number, after which the throughput declines. Figure6b illustrates a decline in the throughput performance of *nonTD* with an increase in the number of tables using 64 client threads. In Sysbench tests, *TDX* achieves a range of 58.5% to 67.4%. After optimizing by adding more I/O threads, its throughput increases to 81.6%-88.1%.



(a) Sysbench Client Threads

(b) Sysbench Tables

(c) Sysbench Transactions

(d) TPC-C Client Threads

(e) TPC-C Warehouses

(f) TPC-C Transactions

(g) T-bench Client Threads

(h) T-bench Data Size

Figure 6: Normalized Sysbench and TPC-C benchmark transaction evaluation and system comparison for coarse-grained privacy-preserving without enclave, with in-enclave architecture, and with in-enclave after adding 4 more I/O threads. The optimized coarse-grained SQL maintains an average 84.7% of the throughput performance consistently across all benchmarks.

Figure 6d-6f compare the TPC-C performance of the different environments. The stability of throughput performance in *nonTD* is depicted in Figure 6e, showcasing consistency as the number of warehouses (W) rises from 64 to 512. However, I/O saturation occurs when W escalates from 512 to 1024. The decrease in performance primarily resulted from slower speed due to the limitation in I/O processing for encryption/decryption threads. To address this, we augmented the I/O processing threads from 2 to 8, resulting in an improvement. The *TDopt* throughput remains relatively stable, fluctuating between 85.4% and 89.8% in Figure 6d with thread numbers ranging from 16 to 128 when W=256. Figure 6f evaluates the performance of TPC-C transactions, the 5 transactions in *TDopt* perform 87.8%, 85.9%, 84.6%, 89.0%, 86.4% throughput of that in *TDX* after adding 4 more threads for I/O processing.

Figure 6g-6h evaluate the T-bench performance in enclave. The results are normalized to the baseline run in the same database launched without enclave. We vary client concurrency by using a fixed 50 million rows as shown in Figure 6g. We observe that the performance of *TDX* achieved 68.4% of the throughput compared to *nonTD*, revealing a significant discrepancy when across NUMA nodes, with the bottleneck being I/O. Consequently, we set up *TDX* instances to be located within a single NUMA node and progressively added more I/O threads until there was no further improvement in performance, indicating that the bottleneck was no longer related to I/O. After TDX tuning, it hits 84.4% throughput of the baseline on average. We then vary the data size in T-bench by using 64 client threads, as shown in Figure 6h. After TDX tuning, *TDopt* performs around 83.5% throughput of *nonTD*.

7.3.2 Fine-grained Privacy-preserving. We evaluate comprehensive scenarios in our fine-grained privacy-preserving DB design, with both granted and ungranted secret columns on select list and predicate. In the Sysbench experiments, we defined the select list and predicate columns of Sysbench read-only SQL statements as secret columns, and removed the two queries containing only aggregation function on the select list from the read-only benchmark. In the T-bench experiments, we conducted artwork point read query and video range read by period.

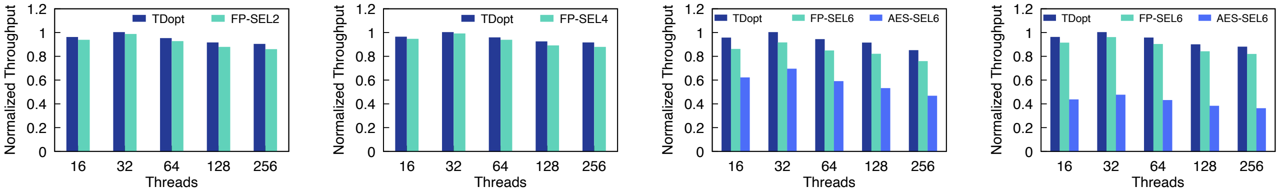
Figure 7a and 7b compare the normalized (relative) performance without and with defining fine-grained privacy-preserving secret columns in different scenarios using Sysbench transactions. Since all *FP* experiments contain granted secret columns, the cost of data retrieval is inevitable. *FP-SEL4* retrieves all the granted columns by the table owner on the select list with one granted predicate. *FP-SEL2* fetches the granted column and padded default mask value for the ungranted columns with one granted predicate. We can see the overheads of fine-grained privacy-preserving features for *FP-SEL4* and *FP-SEL2* are lower than 3.1% and 2.6% respectively.

Figure 7c and 7d show the normalized throughput of the systems with T-bench by using 6 granted secret columns in point and range retrieval, compared to the same column privacy protection realized by the software-encrypted method, which is conducted by utilizing AES-128 algorithm to encrypt and decrypt the 6 secret column data. We observe the overheads of fine-grained privacy-preserving features in SecuDB for point and range read are lower than 9.8% and 5.7% compared to the baseline, respectively. SecuDB performs 1.3× faster than the software-encrypted method. In range read, SecuDB outperforms the software-encrypted method nearly 2.1×, due to the significant cost of encryption and decryption tasks in the software-encrypted method.

7.4 Tamper-resistant Experiments

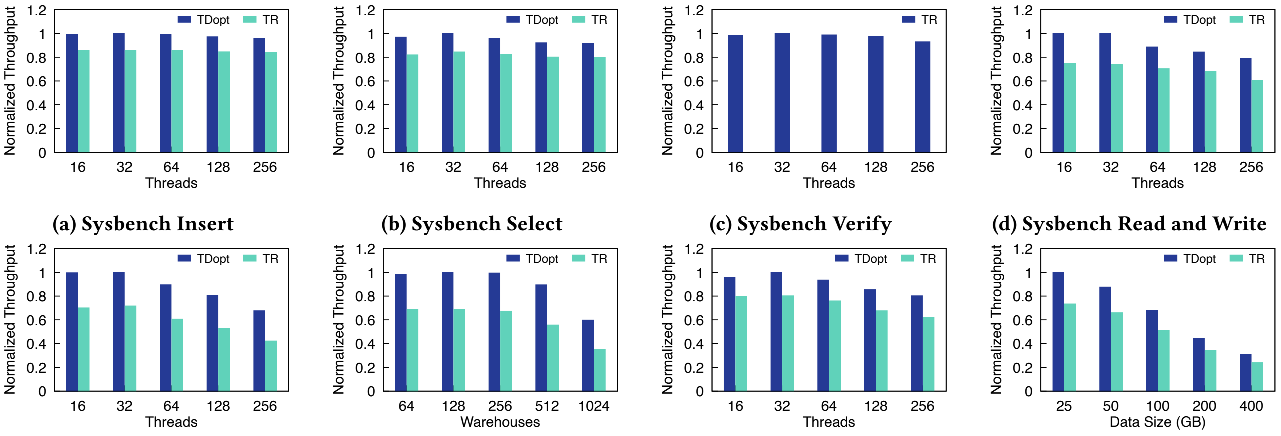
We use the write-only and read-only transaction modes in Sysbench to test the insertion overhead of *aTT*, and then evaluate the verification performance of *aTT* with read-only mode by adding `WITH SIGNATURE` in each select statement. We then evaluate Sysbench, TPC-C, and T-bench performance of tamper-resistant functionality by defining each table as *aTT*.

We populate 60 tables with 1 million records in each to evaluate Sysbench performance of *aTT*. Figure 8a compares the normalized



(a) Sysbench Read **(b) Sysbench Read** **(c) T-bench Point Read** **(d) T-bench Range Read**

Figure 7: Normalized Sysbench and T-bench performance evaluation and comparison for fine-grained privacy-preserving techniques with different scenarios of secret columns on select list and predicates. Fine-grained privacy-preserving features experiments perform nearly 97.2% and 92.2% throughput of that in the Sysbench and T-bench baseline, respectively.



(a) Sysbench Insert **(b) Sysbench Select** **(c) Sysbench Verify** **(d) Sysbench Read and Write**

(e) TPC-C Client Threads **(f) TPC-C Warehouses** **(g) T-bench Client Threads** **(h) T-bench Data Size**

Figure 8: Normalized system performance evaluation and comparison for the tamper-resistant feature across all benchmarks. Tamper-resistant experiments with aTT defined reach an average 77%, 68%, and 78% throughput of that in the Sysbench, TPC-C, and T-bench baseline, respectively.

(relative) insertion performance between aTT and the baseline by varying the concurrent client threads. We can see that aTT insertion achieves 85% to 91% throughput of the baseline. Figure 8b shows that aTT selection reaches 87% to 91% throughput of the baseline. Figure 8c shows aTT verification performance by adding `WITH SIGNATURE` syntax in all Sysbench queries. We observe the data verification throughput is steady when the concurrent threads exceed 64, due to the performance bottleneck of the signature algorithm. Figure 8d shows that aTT reaches around 76.8% of the baseline throughput on average in Sysbench read-write workload.

Figure 8e and 8f show the normalized TPC-C benchmark performance of the two configurations. We observe the trend of throughput difference between aTT version and the baseline in Figure 8e is slight, from 68% to 71 %, when we vary the concurrent threads from 16 to 256 using 256 warehouses. We then fix the client threads to 64 and vary the warehouse number (`W`) to compare the TPC-C benchmark performance between the original and aTT versions. We see the aTT version performs from 74% to 58% TpmC of that in the baseline when `W` grows from 64 to 1024, respectively. This suggests the performance dropping trend is caused by heavier I/O cost in aTT when data volume gets larger, due to the bigger size compared to general table.

Figure 8g and 8h compares the T-bench performance with and without aTT defined. The results are normalized to the baseline performed in the same database launched without TEE. We observe T-bench with aTT version achieves 80.2% performance of that in the baseline when using 50 million records in Figure 8g, while the aTT version reaches 75.7% throughput of that in the original version defined without aTT shown in Figure 8h, meaning the overhead of the tamper-resistant feature is insignificant in practice.

8 CONCLUSION

In this paper, we present SecuDB, a secured relational database with privacy-preserving and tamper-resistant functionalities by residing the entire database engine into TEE enclave, i.e., Intel TDX. We devise a novel fine-grained privacy-preserving framework by column mask-enabled visibility control in an in-enclave architecture, and propose TEE-endorsed advanced temporal tables to support efficient tamper-resistance. The experimental results show that the multi-granularity privacy-preserving and tamper-resistance features are efficient and practical in both standard OLTP benchmarks and real-world use cases.

REFERENCES

[1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*. 1–15.

[2] Panagiotis Antonopoulos, Arvind Arasu, Kunal D Singh, Ken Eguro, Nitish Gupta, Rajat Jain, Raghav Kaushik, Hanuma Kodavalla, Donald Kossmann, Nikolas Ogg, et al. 2020. Azure SQL database always encrypted. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1511–1525.

[3] Panagiotis Antonopoulos, Raghav Kaushik, Hanuma Kodavalla, Sergio Rosales Aceves, Reilly Wong, Jason Anderson, and Jakub Szymaszek. 2021. SQL Ledger: Cryptographically Verifiable Data in Azure SQL Database. In *Proceedings of the 2021 International Conference on Management of Data*. 2437–2449.

[4] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, and Ramarathnam Venkatesan. 2013. Orthogonal Security with Cipherbase. In *CIDR*.

[5] 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 *SIGMOD*. 251–266.

[6] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C-W Phan. 2008. Sha-3 proposal blake. *Submission to NIST* 92 (2008), 1–79.

[7] AWS. 2018. *Amazon quantum ledger database (qldb)*. https://aws.amazon.com/qldb

[8] Sumeet Bajaj and Radu Sion. 2011. TrustedDB: a trusted hardware based database with privacy and data confidentiality. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 205–216.

[9] Sumeet Bajaj and Radu Sion. 2013. TrustedDB: A trusted hardware-based database with privacy and data confidentiality. *IEEE Transactions on Knowledge and Data Engineering* 26, 3 (2013), 752–765.

[10] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. 2011. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Annual Cryptology Conference*. Springer, 578–595.

[11] ByteDance. 2022. *Volcano Engine*. https://www.volcengine.com

[12] ByteDance. 2023. *Make everything a breeze*. https://www.larksuite.com/

[13] ByteDance. 2023. *Make Your Day - TikTok*. https://www.tiktok.com/

[14] ByteDance. 2023. *Volcengine*. https://github.com/volcengine

[15] 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. *arXiv preprint arXiv:2303.15540* (2023).

[16] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*. Springer, 409–437.

[17] Alibaba Cloud. 2023. *Enable PolarDB Always Encrypted*. https://www.alibabacloud.com/help/polardb/polardb-for-mysql/user-guide/enable-confidential-engine

[18] Alibaba Cloud. 2023. *Operator supported in always encrypted database*. https://help.aliyun.com/rds/apsaradb-rds-for-postgresql/supported-features

[19] Intel Corp. 2020. *Software guard extensions sdk developer reference for linux* os*. https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/linux-overview.html

[20] Intel Corp. 2024. *Intel® Trust Domain Extensions (Intel® TDX)*. https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html

[21] Intel Corp and Bytedance Inc. 2024. *Full disk encryption solution in the confidential computing environment*. https://github.com/cc-api/full-disk-encryption

[22] Joan Daemen and Vincent Rijmen. 1999. AES proposal: Rijndael. (1999).

[23] Christopher John Date, Hugh Darwen, and Nikos Lorentzos. 2014. *Time and relational theory: temporal databases in the relational model and SQL*. Morgan Kaufmann.

[24] Jan-Erik Ekberg, Kari Kostiainen, and Nadarajah Asokan. 2013. Trusted execution environments on mobile devices. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 1497–1498.

[25] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. 2019. BlockchainDB: A shared database on blockchains. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1597–1609.

[26] Muhammad El-Hindi, Tobias Ziegler, Matthias Heinrich, Adrian Lutsch, Zheguang Zhao, and Carsten Binnig. 2022. Benchmarking the second generation of intel sgx hardware. In *Proceedings of the 18th International Workshop on Data Management on New Hardware*. 1–8.

[27] Intel Trust Domain Extension. 2023. *Intel TDX Connect Architecture Specification*. https://www.intel.com/content/www/us/en/content-details/773614/intel-tdx-connect-architecture-specification.html

[28] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 169–178.

[29] 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 ACM SIGMOD international conference on Management of data*. 216–227.

[30] Mike Hearn and Richard Gendal Brown. 2016. Corda: A distributed ledger. *Corda Technical White Paper* 2016 (2016). https://www.corda.net/content/corda-technical-whitepaper.pdf

[31] IBM. 2021. *CODEPAGE option syntax*. https://www.ibm.com/docs/en/cobol-zos/6.3?topic=options-codepage

[32] IBM. 2023. *Column-level privileges*. https://www.ibm.com/docs/en/informix-servers/14.10?topic=privileges-column-level

[33] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2020. Trusted execution environments: properties, applications, and challenges. *IEEE Security & Privacy* 18, 2 (2020), 56–60.

[34] David Kaplan, Jeremy Powell, and Tom Woller. 2016. *AMD memory encryption*. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf

[35] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. 2018. Integrating remote attestation with transport layer security. *arXiv preprint arXiv:1801.05863* (2018).

[36] Kevin Kollenda. 2023. General overview of AMD SEV-SNP and Intel TDX. (2023). https://sys.cs.fau.de/extern/lehre/ws22/akss/material/amd-sev-intel-tdx.pdf

[37] 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*. 1–16.

[38] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. 2022. Design and Verification of the Arm Confidential Compute Architecture. In *OSDI*. 465–484.

[39] Trent McConaghy, Rodolphe Marques, Andreas Müller, Dimitri De Jonghe, Troy McConaghy, Greg McMullen, Ryan Henderson, Sylvain Bellemare, and Alberto Granzotto. 2016. Bigchaindb: a scalable blockchain database. *white paper, BigChainDB* (2016).

[40] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. 2016. Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. 1–9.

[41] Microsoft. 2023. *Temporal tables*. https://learn.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables

[42] Microsoft. 2024. *GRANT (Transact-SQL)*. https://learn.microsoft.com/en-us/sql/t-sql/statements/grant-transact-sql

[43] MongoDB. 2022. *Queryable Encryption: Protect your confidential workloads*. https://www.mongodb.com/products/queryable-encryption

[44] MySQL. 2023. *Prepared Statements*. https://dev.mysql.com/doc/refman/8.4/en/sql-prepared-statements.html#prepared-statements-in-applications

[45] The Horizon 2020 Framework Programme of the European Union. 2020. *Everything you need to know about the Right to be forgotten*. https://gdpr.eu/right-to-be-forgotten

[46] Monique Ogburn, Claude Turner, and Pushkar Dahal. 2013. Homomorphic encryption. *Procedia Computer Science* 20 (2013), 502–509.

[47] Oracle. 2022. *Blockchain Tables in Oracle Database 21c*. https://oracle-base.com/articles/21c/blockchain-tables-21c

[48] Oracle. 2023. *Encryption and Compression Functions*. https://dev.mysql.com/doc/refman/8.0/en/encryption-functions.html

[49] Oracle. 2023. *Implementing Temporal Validity*. https://www.oracle.com/webfolder/technetwork/tutorials/obe/db/12c/r1/ilm/temporal/temporal.html

[50] Oracle. 2024. *Column-level privileges*. https://dev.mysql.com/doc/refman/en/grant.html

[51] Stuart L Pardau. 2018. The California consumer privacy act: Towards a European-style privacy regime in the United States. *J. Tech. L. & Pol'y* 23 (2018), 68.

[52] Sandro Pinto and Nuno Santos. 2019. Demystifying arm trustzone: A comprehensive survey. *ACM computing surveys (CSUR)* 51, 6 (2019), 1–36.

[53] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. 2016. Arx: an encrypted database using semantically secure encryption. *Cryptology ePrint Archive* (2016).

[54] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. 2019. Arx: An Encrypted Database using Semantically Secure Encryption. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1664–1678.

[55] Microsoft SQL Sever 2022 Preview. 2022. *Always Encrypted with secure enclaves*. https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-enclaves

[56] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A secure database using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 264–278.

[57] Ravi Sahita, Dror Caspi, Barry Huntley, Vincent Scarlata, Baruch Chaikin, Siddhartha Chhabra, Arie Aharon, and Ido Ouziel. 2021. Security analysis of confidential-compute instruction set architecture for virtualized workloads. In *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 121–131.

[58] Muhammad Usama Sardar, Saidgani Musaev, and Christof Fetzer. 2021. Demystifying attestation in intel trust domain extensions via formal verification. *IEEE access* 9 (2021), 83067–83079.

[59] Shiny Sebastian, Simon P. Johnson, Md Iqbal Hossain, and Chao Gao. 2023. *Performance Considerations of Intel® Trust Domain Extensions on 4th Generation Intel® Xeon® Scalable Processors.* https://www.intel.com/content/www/us/en/developer/articles/technical/trust-domain-extensions-on-4th-gen-xeon-processors.html

[60] Oracle Cloud Security. 2023. *Column-level security with Oracle Database's data redaction.* https://blogs.oracle.com/cloudsecurity/post/columnlevel-security-oracle-databases-data-redaction

[61] Nathan Senthil, Govindarajan Chander, Saraf Adarsh, et al. 2019. Blockchain meets database: design and implementation of a blockchain relational database [J]. In *Proceedings of the VLDB Endowment*, Vol. 12. 1539–1552.

[62] Amazon Web Services. 2022. *Encrypting Amazon Aurora resources.* https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Overview.Encryption.html

[63] AMD Sev-Snp. 2020. Strengthening VM isolation with integrity protection and more. *White Paper, January* 53 (2020), 1450–1465.

[64] Youren Shen, Hongliang Tian, Yu Chen, Kang Chen, Runji Wang, Yi Xu, Yubin Xia, and Shoumeng Yan. 2020. Occlum: Secure and efficient multitasking inside a single enclave of intel sgx. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 955–970.

[65] Rohit Sinha and Mihai Christodorescu. 2018. VeritasDB: High Throughput Key-Value Store with Integrity. *IACR* 2018 (2018), 251.

[66] Intel Developer Site. 2023. Intel Trust Domain Extensions (Intel TDX).

[67] Sysbench. 2023. *Scriptable database and system performance benchmark.* https://github.com/akopytov/sysbench

[68] TPC. 2022. *TPC-C Benchmark.* https://www.tpc.org/tpcc

[69] Chia-Che Tsai, Donald E Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 645–658.

[70] Stephen Lyle Tu, M Frans Kaashoek, Samuel R Madden, and Nickolai Zeldovich. 2013. Processing analytical queries over encrypted data. (2013).

[71] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. 2019. StealthDB: a Scalable Encrypted Database with Full SQL Query Support. *Proc. Priv. Enhancing Technol.* 2019, 3 (2019), 370–388.

[72] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10, 3152676 (2017), 10–5555.

[73] ByteDance Volcengine. 2022. *Cloud database for veDB.* https://www.volcengine.com/product/vedb-mysql

[74] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted execution environments on {GPUs}. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 681–696.

[75] Xinying Yang, Sheng Wang, Feifei Li, Yuan Zhang, Wenyuan Yan, Fangyu Gai, Benquan Yu, Likai Feng, Qun Gao, and Yize Li. 2022. Ubiquitous Verification in Centralized Ledger Database. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 1808–1821.

[76] Xinying Yang, Ruide Zhang, Cong Yue, Yang Liu, Beng Chin Ooi, Qun Gao, Yuan Zhang, and Hao Yang. 2023. VeDB: A Software and Hardware Enabled Trusted Relational Database. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.

[77] Xinying Yang, Yuan Zhang, Sheng Wang, Benquan Yu, Feifei Li, Yize Li, and Wenyuan Yan. 2020. LedgerDB: A centralized ledger database for universal audit and verification. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3138–3151.

[78] Cong Yue, Tien Tuan Anh Dinh, Zhongle Xie, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Xiaokui Xiao. 2023. GlassDB: An Efficient Verifiable Ledger Database System Through Transparency. *PVLDB* 16, 6 (2023), 1359–1371.

[79] Cong Yue, Meihui Zhang, Changhao Zhu, Gang Chen, Dumitrel Loghin, and Beng Chin Ooi. 2023. VeriBench: Analyzing the Performance of Database Systems with Verifiability. *PVLDB* 16, 9 (2023), 2145–2157.

[80] Yiming Zhang, Yuxin Hu, Zhenyu Ning, Fengwei Zhang, Xiapu Luo, Haoyang Huang, Shoumeng Yan, and Zhengyu He. 2023. SHELTER: Extending Arm CCA with Isolation in User Space. In *32nd USENIX Security Symposium (USENIX Security'23)*.