ASPLOS'24 Submission #405– Confidential Draft – Do Not Distribute!!

NEXUS: Secure and Efficient Collaborative Analytics on Encrypted Databases

Abstract

Encrypted Databases (EDBs) have become a compelling solution to facilitate collaborative analytics among multiple organizations, by allowing to pool the organizations' private data in an encrypted form and run analytical queries over the encrypted data to derive mutually beneficial insights, e.g., for medical studies across hospitals, fraud detection across banks, etc. However, existing EDBs support neither general queries efficiently with sublinear (ideally logarithmic) search complexity, nor protecting query integrity in the face of prevailing integrity attacks.

We present NEXUS*, the first query system that allows multiple parties to efficiently run general SQL queries on EDBs with both data confidentiality and query integrity guarantees.* NEXUS *features a new Merkle rangeFilter Tree (MFT) secure index abstraction, which utilizes the membership information of encrypted data to construct a logarithmic-depth recursive index structure resembling a Merkle tree. Unlike prior works employing expensive query-specific cryptographic schemes, MFT supports general queries by relying on efficient data membership evaluation, without revealing plaintexts. Moreover, the recursive index structure of MFT empowers* NEXUS *to authenticate query results, thus ensuring query integrity simutaneously. Evaluation of* NEXUS *on popular SQL and NoSQL databases shows up to 94.7% lower latency and up to 15.2X higher throughput compared to the state-of-the-art.*

1. Introduction

Numerous mission-critical services today collect valuable yet sensitive user data. By pooling this data together and jointly performing query analytics on the aggregated data, these services can obtain valuable insights that benefit all parties involved. For instance, such collaboration can enhance credit risk models in financial services [12, 69], facilitate medical studies for prompt disaster responses [32, 37, 65, 20, 40], and optimize transportation routes to reduce congestion and enhance overall mobility in cities [71, 54, 23]. However, the sharing of sensitive data among different organizations is often hindered due to privacy concerns or business competition.

To securely store and query private data, organizations are increasingly adopting cloud *encrypted databases* (EDBs) [51, 64, 3, 5, 50]. EDBs promise to enable such collaborative scenarios by storing data in an encrypted form and refraining from decrypting it on the fly. This ensures data confidentiality even against privileged database administrators or external attackers with full access to the data stored in EDB servers. To attain comparable computing capabilities to insecure databases like MySQL [45], EDBs often rely on Homomorphic Encryption (HE) [22, 43, 46], which allows computation over encrypted data without the need for decryption.

Unfortunately, although the industry and academia have made great efforts to develop EDBs (e.g., CryptDB [51], HE-libDB [5], Senate [50]) with query capabilities similar to insecure databases, none of existing EDBs can efficiently run collaborative analytical queries. This is due to two major challenges that stem from the encrypted nature of EDB data.

Firstly, analytical queries can encompass various types, including point and range queries, in many scenarios [12, 32, 54]. While HE allows for arbitrary query computations, it lacks support for *encrypted search* capabilities. Thus, one primary goal for a collaborative query system is to handle diverse query types on HE data while aiming to achieve a search complexity that is sublinear to the size of the database. Ideally, this complexity should be comparable to that of an insecure database [3, 53, 45] that utilize a B$^+$-tree (or its variants) to achieve logarithmic-complexity searches.

However, existing systems lack efficient support for general query types with logarithmic search complexity, as they rely on cryptographic tools that are inherently designed to meet specific query requirements. For instance, Senate [50] uses multi-party computation [18, 73] and requires negotiating the specific query type and data sharing among parties for each query, leading to significant communication overhead. HE-libDB [5] effectively supports point queries by computing specific HE functions over the entire database to compare the equivalence between HE-encrypted data, but it fails to handle range queries. Although CryptDB [51] leverages property-preserving encryption [26, 4] to support both point and range queries with logarithmic search complexity, it still incurs significant cryptographic overhead during encrypted searches, as evaluated in §6.

The second challenge arises from the vulnerability of EDBs to integrity breaches when hosted on an untrusted cloud. In such scenarios, attackers can manipulate query results by modifying, replaying, or dropping data [76, 82]. These manipulations result in incorrect analytical outcomes, jeopardizing the collaborative query service for all parties involved.

Nevertheless, existing systems overlook the vulnerability to integrity breaches and provide a weaker security guarantee. Specifically, they assume that even if the EDB is compromised, it will faithfully follow the protocol. This assumption is unrealistic in many scenarios for two main reasons. First, since each party needs to entrust its sensitive data to EDBs, if the database colludes with competitors, the failure to produce accurate analytical results would lead to a loss of business interests. Second, in real-world attacks, attackers often have the ability to install malware on the server or gain control over it [44, 77], allowing them to alter the database's behavior.

In this work, we present NEXUS[1], the first secure query system for collaborative data analytics that addresses the aforementioned challenges, via a secure index abstraction named *Merkle rangeFilter Tree* (MFT). Based on the MFT abstraction, we establish an end-to-end secure query workflow that ensures both data confidentiality and query integrity: NEXUS utilizes MFTs to locate query-dependent EDBs, which are databases containing matching records for the query parameters, then distributes client queries to these identified EDBs for execution. Within these EDBs, NEXUS performs MFT searches to retrieve the requested data without decrypting them. Last, NEXUS verifies the integrity of query results and securely aggregates them before sending them back to the clients.

To efficiently support general queries in NEXUS's workflow, instead of relying on expensive cryptographic techniques (like HElibDB [5] and CryptDB [51]), MFT employs a lightweight approach by utilizing *data membership* information to construct a searchable data structure, drawing inspiration from the traditional Bloom filter [13]. As depicted in Figure 1, NEXUS transforms data into confidential *cipher identifiers* and places them at the bottom layer of the structure. The membership information linked to these identifiers is propagated to the root through the *Union* operation and subsequently utilized for encrypted searching purposes.

This design choice offers two notable benefits. Firstly, it enables top-down binary encrypted search by evaluating the membership of parameters (i.e., by determining $\in$ or $\notin$) at each index branch, without revealing plaintexts. Secondly, the membership propagation approach leads to a Merkle-tree-like structure that facilitates query authentication by verifying if the requested data is included in the query results, similar to the Merkle tree verification in cryptocurrency systems [55, 48].

Concretely, to defend against integrity breaches, NEXUS employs query authentication via two rules. Firstly (freshness rule), NEXUS verifies whether the query results are based on the latest version by checking the MAC on both the fetched records and identifiers used for indexing. This rule ensures that any modification or replay attempts are detected. Secondly (completeness rule), to detect drops of query results, NEXUS verifies whether the query results fall within the specified query boundaries and whether the MFT root rebuilt from the results aligns with the original MFT root. Therefore, by leveraging the MFT abstraction, NEXUS achieves data confidentiality and query integrity in a unified manner.

We implemented NEXUS on CryptDB [51], a widely used modular framework for evaluating EDBs. We built two EDBs by integrating NEXUS with MySQL [45], the popular SQL backend for EDBs, and WiredTiger [16], the default NoSQL key-value backend for MongoDB. In addition, we devised

[1]NEXUS stands for the *interconnected* membership-based search index.

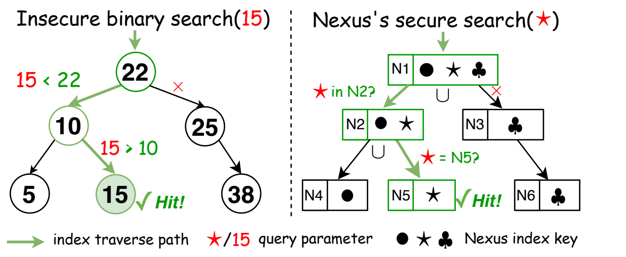


Figure 1: NEXUS's core index concept for secure search. The left part depicts the traditional insecure approach that relies on numerical comparison for index traversal. In contrast, NEXUS traverses by evaluating ***data membership*** of transformed cipher identifiers (e.g., ★), without revealing plaintexts (§4).

a data aggregation protocol to securely aggregate analytical results from collaborative parties (§4.3). We compared NEXUS with CryptDB [51] and HElibDB [5] using typical workloads (e.g., TPC-C [41]). The evaluation results show that:

- NEXUS is general. NEXUS was able to support general queries, including *point*, *range*, *join*, *insert*, *update*, and *delete* among EDBs, while HElibDB only supports point queries. Moreover, NEXUS achieved logarithmic search complexity for all these query types, as proven in §4.4 and evaluated in §6.
- NEXUS is efficient. NEXUS achieved up to 28% lower latency and 1.88X higher throughput than CryptDB. Compared to HElibDB, NEXUS achieved up to 94.7% lower latency and 15.2X higher throughput. All these improvements were achieved while maintaining query integrity, which was not enforced in the baselines.
- NEXUS is scalable. NEXUS scales to an increasing number of involving parties and larger databases. As the size of databases grows, NEXUS maintains logarithmically increasing latency, which is consistent with our complexity analysis.

Our main contribution is MFT, a new secure index abstraction that enables multiple parties to run general SQL queries on EDBs with logarithmic search complexity. We exploit the membership information for secure index construction, allowing for encrypted search and query authentication through membership evaluation, ensuring both data confidentiality and query integrity in a unified manner. These unique features enable NEXUS to facilitate secure data sharing and collaborative queries across multiple parties, making it highly suitable for various mission-critical distributed applications such as medical diagnosis [42, 39], fraud detection [79, 8], and anti-money laundering [59, 70]. NEXUS's source code is released on `github.com/2024asplos405/Nexus`.

2. Related Work

2.1. Encrypted Databases on the Cloud

Organizations (e.g., enterprises or individuals) today are increasingly shifting their sensitive data to cloud databases due to their scalability and rich query capabilities. *Encrypted databases* (EDBs) promise to protect the confidentiality of outsourced sensitive data from both privileged insiders, such

System	Data Confidentiality	Query Integrity	Joint Query	Point Search	Range Search
MySQL [45]	✗	✗	✓	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
∗ EnclaveDB [53]	✓	✓	✗	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
∗ HybrTC [75]	✓	✓	✓	$\mathcal{O}(n)$	$n/a.$
• CryptDB [51]	✓	✗	✗	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
• HElibDB [5]	✓	✗	✗	$\mathcal{O}(n)$	$n/a.$
• Senate [50]	✓	✓	✓	$n/a.$	$n/a.$
• Dory [17]	✓	✓	✗	$\mathcal{O}(n)$	$n/a.$
• **NEXUS**	✓	✓	✓	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Table 1: Comparison of representative systems. "∗ / •" denotes whether the system utilizes trusted hardware (∗), or a software-only approach (•). $n/a.$ means not supporting such query type.

as database administrators, and external attackers that alter the database behaviors [1, 79, 3, 5, 50]. Existing EDBs can be classified into two main categories, as outlined in Table 1.

The first category utilizes *Trusted Execution Environment* (TEE) to create a secure area that ensures both data confidentiality and query integrity [53, 63]. TEE databases like EnclaveDB [53] and EdgelessDB [3] offer straightforward query support by searching on plaintexts through TEE-shielded standard indexes like B$^+$-trees, thus retaining the logarithmic time complexity typical of these indexes. However, relying on TEEs involves trusting specific hardware vendors like Intel [30] and AMD [62], which has prompted many organizations to explore software-only approaches [1, 79, 27]. NEXUS belongs to the software-only category.

The second (software-only) category uses *homomorphic encryption* (HE), especially fully HE schemes like BGV [22], to enable arbitrary computations while concealing the plaintexts. Existing HE databases employ various cryptographic tools to enhance the searchability of encrypted data. CryptDB [51] utilizes a recursive onion-like encryption process with different property-preserving encryption algorithms [11, 66, 49], building indexes on each layer of onion-encrypted data to facilitate specific types of queries. HElibDB relies on Fermat's Little Theorem [21] to directly evaluate the equivalence of HE data but requires scanning the entire database. Senate [50] employs secure multi-party computation [18, 73] but necessitates online negotiation and does not support encrypted search.

Indeed, these cryptographic approaches provide a strong confidentiality guarantee (named *IND-CPA* [2]), but they suffer from limited query capabilities, high cryptographic overhead, and overlook the vulnerability to integrity breaches. Dory [17] takes the first step in building indexes using efficient cryptographic hash functions like SHA-256 [60], ensuring relaxed yet sufficient *irreversible* security [57], which means the encrypted data cannot be reversed to plaintexts even with the knowledge of the hash functions. Although Dory effectively protects both data confidentiality and query integrity, it supports only point queries on file stores and with linear search complexity, making it unsuitable for our requirements. Like Dory, NEXUS also ensures irreversible security through the use of cryptographic hash functions, but NEXUS goes beyond by supporting general queries with sublinear complexity.

2.2. Example Queries

Our motivation is to enable efficient searches on EDBs and connect them, such that a client can run general analytical queries in distributed EDBs managed by different parties. In principle, NEXUS can support arbitrary queries powered by point and range searches; it does not currently support fuzzy search using operators such as 'LIKE' [45]. We now give two use cases and example queries facilitated by NEXUS.

Query 1. Collaborative medical study [27]. COVID-19 is a contagious disease caused by the coronavirus. As part of a clinical research study, there are t hospitals $P_1 \dots P_t$ that wish to collaboratively analyze the sequela and the total number of individuals affected by COVID-19. However, they cannot directly share their databases with each other to run this query due to privacy concerns regarding patient information.

```
SELECT COUNT(*), sequela
FROM medical_encrypted_databases
JOIN on patient_ID
WHERE disease.is_infected = true
AND disease.category = 'COVID-19';
```

Query 2. Government statistics [79]. The government needs to query the total fiscal spending last year of t different departments $P_1 \dots P_t$. However, these departments are bound by privacy regulations and thus cannot directly publish private citizen information with the government. Despite this constraint, the departments still need to collaboratively run the analytical query, to sum up the fiscal spending.

```
SELECT SUM(expenses), department_ID
FROM departmental_encrypted_databases
JOIN on expenses.department_ID
WHERE expenses.timestamp BETWEEN 2022 AND 2023
AND department.category = 'financial';
```

2.3. Preliminaries

Bloom filter [13]. It is a compact data structure designed for efficient membership testing. It operates by using a single set to store the hash values of all elements. Specifically, a Bloom filter B is an m-bit array that uses k independent and uniformly distributed cryptographic hash functions $\{H_k\}$ (e.g., SHA-256), each of which maps an element to a position in B. To insert an element x into the filter, we compute its hash using $\{H_k\}$ and set all bits at the corresponding indices in B to 1. To test if an element y is a member of the set, we again hash y using $\{H_k\}$. If any of the bits at the corresponding indices in B are 0, then y is definitely not in the set (i.e., zero false negatives); otherwise, y has a high probability of being in the set (with small configurable false positives [13]).

Notably, while prior systems have utilized Bloom filters to build indexes, their use was restricted to specific database types (e.g., file stores) or targeted specific queries for a single party [17, 47, 35]. NEXUS differentiates itself by striving to facilitate general queries in a multi-party joint query scenario.

3. Overview

3.1. System Setup

Entities. Same as existing query systems [45, 5, 52], NEXUS consists of three participants: clients, EDB servers (for short, servers), and a proxy. Clients are grouped into *parties* (e.g., banks and hospitals), with each party outsourcing its sensitive data to servers hosted on an untrusted cloud. The servers provide query services to clients collaboratively, by intercepting all SQL queries through a proxy, which is responsible for dispatching queries to execute on query-dependent servers (EDBs containing the requested data) and aggregating the query results for clients.

Data model. We employ the relational data model to exemplify our design. In NEXUS, an encrypted table T contains n confidential columns (or attributes) and has a primary key. Clients encrypt every value before sending it to the server. Concretely, for a row (i.e., record) r, the value v_i in column c_i is represented as ciphertext $E(v_i)$ using an encryption algorithm E, which is typically the fully HE scheme BGV [22]. Each party (along with its clients) *cannot* disclose its HE secret key to any servers or other parties for data confidentiality. NEXUS provides the flexibility for clients to encrypt different columns with distinct secret keys and supports searching on non-primary keys using secondary indexes [61].

In NEXUS, all records are *versioned*. This involves clients maintaining the latest version of their own record locally, and keeping it synchronized with the corresponding servers. In such a case, clients are assumed to have already retrieved the latest version of a record before submitting a new update query to that particular record.

NEXUS exhibits extensibility owing to its *unclustered* architecture [64] resembling insecure databases [45, 52]. This separates the design of database indexes from the underlying storage, thus allowing NEXUS, with its new secure index, to seamlessly support both the structured relational data model and unstructured key-value pairs (both evaluated in §6).

3.2. Threat Model and Guarantees

NEXUS adopts a strong threat model in which a malicious adversary $\mathfrak{A}$ can corrupt databases and arbitrarily deviate from NEXUS's protocol to learn private data. $\mathfrak{A}$ can either passively observe or actively manipulate query results and tamper with stored data in the databases via modification, replay, or deletion. However, in line with prior research [17, 51], we assume that $\mathfrak{A}$ cannot reverse cryptographic hash functions (e.g., SHA-256) or compromise secret keys for encryption. This threat includes compromises of database software and even access to the RAM of physical machines. Given the increasing trend of outsourcing databases to public cloud environments and relying on third-party database administrators [45, 52], we believe this threat is increasingly important.

Apart from the databases, we assume all other entities are honest. Specifically, we assume that clients are always honest and do not collude with any database, as the identities of all joint parties (including their clients) are typically known in a collaborative query system [75, 50]. Same as existing systems (e.g., CryptDB [51]), we also assume that the proxy is honest. In real-world deployments of collaborative query systems [34, 33], a proxy is usually deployed within a secure and controlled environment in an organization's infrastructure; the trustworthiness of the proxy is based on the reputation of the organization. While blacklisting [74, 67] and BFT techniques [55, 48] are alternative methods to tolerate a corrupted proxy, they are outside the scope of this work.

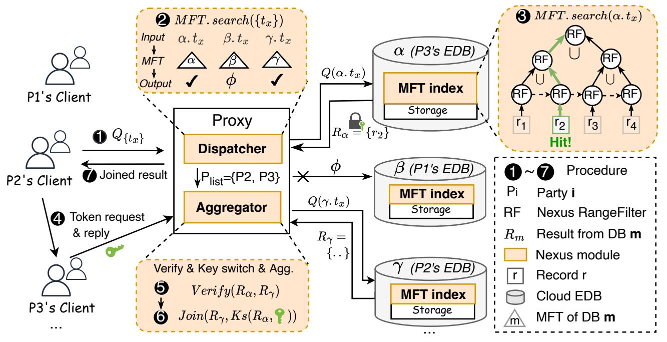


Figure 2: NEXUS's architecture and detailed workflow powered by its secure index abstraction *Merkle rangeFilter Tree* (MFT).

Same as [17, 51], we assume the underlying storage engine is secure. This ensures that a server can reliably retrieve the correct data version from the underlying storage by detecting and preventing rollback and fork attacks, which have been extensively studied in prior works [15, 68, 76].

Guarantees. NEXUS offers two crucial security guarantees. Firstly, it provides confidentiality for both data content and the names of columns and tables. Secondly, NEXUS maintains query integrity, which means any integrity breaches in query results will be detected. However, it is important to note that NEXUS *does not* hide the size of databases, the index volume, the table structure, the number of rows, or the types of columns. The security of NEXUS is *not* perfect: NEXUS could reveal side-channel information that corresponds to the type of computation that queries perform on the database, such as equality comparisons and sorting operations.

How close is NEXUS to "optimal" security? Fundamentally, optimal security is achieved in recent theoretical cryptography works by concealing all side-channel information or database access patterns [50, 75, 78]; however, these proposals are impractical due to prohibitively high computational overhead. In contrast, NEXUS prioritizes practicality. We leave the integration of advanced side-channel defenses for future work.

3.3. NEXUS's Workflow Overview

Figure 2 shows a typical point query workflow in three phases.
• **Phase 1: query dispatch.** To submit a point query Q (❶), the client transforms her parameter x into a secure primitive called *rangefilter* (describe in §4.2). This primitive, denoted as t_x, acts as a cipher identifier for x to search and is always

encrypted. The client generates rangefilters $\{t_x\}$ for all joint parties using the shared rangefilter keys (i.e., SHA-256 hash keys) as the query parameter, encrypts Q via AES, signs Q, and sends to the proxy via a TLS-enabled link. The proxy will drop the connection if malformed queries are received, e.g., those with invalid signatures from unknown parties.

The proxy runs a dispatching thread to determine the appropriate servers to forward queries (❷). Upon receiving Q, the proxy decrypts it and searches for $\{t_x\}$ in pre-stored secure indexes called *Merkle rangeFilter Trees* (MFT) of collaborative servers, which function as a group of *filters*. If there are relevant records in d (i.e., records that match the query parameter), the proxy forwards Q to that server. This relevance is confirmed by searching parameters in d's MFT and it does not return a ϕ (empty set). Meanwhile, the proxy generates a list of parties P_{list} to which d_i belongs, and submits P_{list} to the client and the aggregator thread (see **Phase 3**).

• **Phase 2: local search.** NEXUS adopts a new approach to search for requested data (e.g., $\alpha.t_x$ in server α) within encrypted data on each server by using MFTs (❸). The MFT is constructed by organizing rangefilters into a Merkle-tree-like data structure, where each inner node's value is the *Union* operation result of its child nodes. The insight of such index construction is to enable efficient binary search using *data membership* information, eliminating the need for plaintext numerical comparisons as in B$^+$-trees.

• **Phase 3: data aggregate.** Upon receiving query results R from P_i's EDB, the proxy initiates a two-step integrity check (❺). Firstly, it verifies whether R is based on the latest version and whether the data content remains complete. This is achieved by validating the MAC on both the versioned records and rangefilters, ensuring that any modifications or replay attempts are detected. Secondly, it detects any drops in the query results by confirming whether R matches the query parameters or falls within the specified boundary (depending on the query type) and whether the MFT root rebuilt from R aligns with P_i's original MFT root.

Once the integrity check is passed, the proxy aggregates all query results by performing key switching on the results using the tokens requested from the parties belonging to P_{list} (❹). The token requesting process runs in parallel with **Phase 2**. The key switching process ensures that the client can decrypt and utilize the analytical results without revealing other parties' secret keys (HE keys for encrypting data). Finally, the proxy joins the key-switched results (❻) and returns them to the client (❼).

Overall, the highlight of NEXUS lies in its ability to provide both data confidentiality and query integrity, while maintaining high efficiency with logarithmic search complexity for general SQL queries. This is achieved by NEXUS's MFT secure index, featuring a new insight of membership-based encrypted search and overcoming the shortcomings of prior cryptographic methods. Leveraging the MFT, NEXUS builds the first unified query system that facilitates multi-party collaborative analytics.

4. Protocol Description

4.1. Preliminary Index

Opportunities. In the context of enabling encrypted search for collaborative analytics, the Bloom filter possesses a key *confidential* characteristic, which stems from its design of storing only the *hash* of elements rather than the original plaintexts (§2.3). By relying on cryptographic hash functions, the Bloom filter becomes computationally infeasible to be reversed back to plaintexts even with the knowledge of these hash functions, which has been established in prior research [25].

Unfortunately, the conventional Bloom filter is *monolithic* in nature, in the sense that all elements are assigned to a single bit array. As a result, it can only support membership tests, indicating the presence of an element (hash) in a set. However, it is not capable of functioning as an index for locating encrypted data within the storage.

Design rationale. We observe that the confidential characteristic of the Bloom filter can be useful for encrypted search, thus in our preliminary design, we decompose the monolithic Bloom filter into multiple ***subfilters***, with each subfilter solely storing the hash (in bits) for a single element.

Definition 1 (NEXUS's subfilter)**.** *A subfilter is the Bloom filter for individual data, i.e., only one element (hash) is assigned to a set for membership tests.*

Figure 3 depicts the core concept of utilizing subfilters for encrypted search. Specifically, subfilters have a constant array size and are placed at the leaf nodes, wherein each subfilter represents a primary (index) key (e.g., `"111000"` for x_2). These subfilters are sorted in an order of plaintext keys. For each inner node, its value is determined by performing element-wise *OR* gate computations on its child subfilter nodes: given subfilters with length m, for an inner node *inner* with l child nodes $\{cn_1, \ldots, cn_l\}$, each bit in the inner node's subfilter is

$$inner[i] \;=\; cn_1[i] \; OR \; \ldots \; OR \; cn_l[i] \;, \; \forall \, i \in [0, m] \qquad (1)$$

The design rationale behind the subfilter tree is that, if any entry in a child subfilter node is `"1"`, the *OR* gate computation ensures that the corresponding entry in its parent subfilter node must also contain the `"1"` bit. This feature guarantees that, if a record *exists* in a node, it must also exist in its parent node. We term this approach *membership propagation*, which enables the utilization of data membership information for top-down tree traversal while preserving confidentiality inherited from cryptographic hash functions used for constructing subfilters, eliminating the need for plaintext numerical comparisons.

Search process. Figure 3 demonstrates the search process for a point query. To begin, NEXUS transforms the query parameter into a subfilter using Equation 1, NEXUS then traverses the subfilter tree to locate the corresponding record. During the traversal, NEXUS determines the path (in green) with subfilter comparisons, which verifies if every `"1"` bit in the parameter's subfilter exists in the current child node's subfilter. If such verification succeeds, the parameter is passed to that child node,

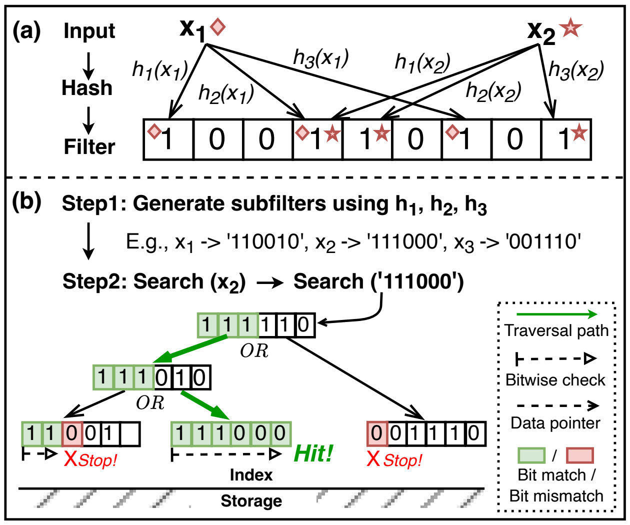


Figure 3: NEXUS's core concept of membership-based search.

and the process continues until the bottom layer is reached. At the bottom layer, a leaf node confirms the match of *every* bit since each subfilter exclusively contains one index key.

Similar to a point search, a range search process within NEXUS involves transforming the range parameters into subfilters and locating two matched leaf nodes; subsequently, all records situated between two leaf nodes are returned. This retrieval process is similar to the conventional B^+-tree.

4.2. NEXUS's Complete Encrypted Search Index

Challenges. The subfilter tree efficiently tackles point and range queries; however, the new membership-based indexing approach brings two distinct challenges. Firstly, the subfilter tree cannot search for *non-existent* keys. Clients can submit range queries with imprecise parameters that do not exist in any of the collaborative servers. The absence of ordering among subfilters makes it infeasible to locate the leaf node corresponding to a non-existent key by finding two adjacent subfilters. Secondly, for the same reason, the subfilter tree can solely support *read* queries because it cannot locate the intended *write* positions among the leaf nodes.

Rangefilter primitive. Based on the preliminary subfilter, we propose *rangefilter*, a primitive designed to address the aforementioned challenges. The key idea of this primitive lies in generating *confidential biases* for subfilters. These biases must adhere to two essential rules. Firstly, the bias should maintain *monotonicity*, ensuring that larger index keys consistently map to larger biases. Secondly, the bias must incorporate *randomness*, essential in preventing adversaries from immediately deducing plaintexts from the observed biases.

To achieve this goal, we utilize the order-reserving encryption (ORE) [14] to generate the biases. ORE relies on a probabilistic encryption scheme, mapping plaintexts into random positions within a specified range, while ensuring that larger plaintexts consistently map to larger ranges. This makes ORE fulfill our requirements of both monotonicity and randomness.

By employing the ORE algorithm to a subfilter and the version number of the subfilter's corresponding record, we generate a bias b. By appending b to the subfilter with length m, we produce a rangefilter with a range spanning from b to $m+b$.

Definition 2 (NEXUS's rangefilter)**.** *A rangefilter is a secure data structure that comprises a subfilter with a confidential bias, providing data membership and ordering information.*

NEXUS's complete index. Building upon the rangefilter primitive, we present the complete index named *Merkle rangeFilter Tree* (MFT). Below we detail the construction and encrypted search process using MFT.

- **MFT construction.** With a set of rangefilters for all index keys, the client constructs MFT by placing these rangefilters at the bottom layer of the structure as leaf nodes, then generating other inner nodes by merging the ranges of child rangefilters using the union operation (algorithm in Appendix A.2). Notably, although MFT uses the merged ranges for index traversal instead of subfilter comparison (as in Figure 3), the union operations on subfilters (performed as element-wise *OR* gate computation on bits) remain essential, as they play a crucial role in protecting query integrity (discuss in §4.3).
- **MFT search.** Next, we detail how MFT handles encrypted point and range searches, which are two fundamental capabilities for analytical queries [45, 52]. The principle is to utilize the ranges within rangefilters, which preserve the numerical ordering of records, for index traversal at the inner node level. At the leaf node level, MFT conducts bitwise subfilter comparison to precisely locate records by evaluating data membership. Our methodology is outlined in **Algorithm** 1.

For point searches, MFT performs a series of range comparisons and subfilter comparisons. Specifically, among the inner nodes, MFT performs inner comparisons to determine

Algorithm 1: MFT Encrypted Search

```
1  Function PointSearch(root, key, opt=point) do
2      if $isLeaf(root)$ then
           ▽ Subfilter comparison at the bottom layer
3          if $key.rft = root.rft$ then
4              return root
       ▽ Use rangefilter's ranges to traverse the MFT
5      if $key.rng \in root.left.rng$ then
6          return PointSearch(root.left, key, point)
7      if $key.rng \in root.right.rng$ then
8          return PointSearch(root.right, key, point)
       ▽ Left boundary is non-existent (Fig 4's case)
9      if $opt = left$ then
           ▽ Find smallest node larger than boundary
10         $root \leftarrow root.right$
11         while $!isLeaf(root)$ do
12             $root \leftarrow root.left$
13         return root
       ▽ Right boundary is non-existent
14     if $opt = right$ then
           ▽ Find largest node smaller than boundary
15         ...                              ▷ Similar to line 11∼14
16 Function RangeSearch(root, [key1, key2]) do
17     $left \leftarrow$ PointSearch(root, key1, left)
18     $right \leftarrow$ PointSearch(root, key2, right)
19     return $[left, right]$               ▷ Retrieve sorted leaf nodes
```

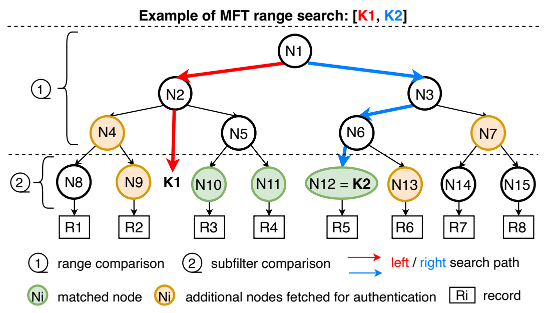


Figure 4: An example of a non-existent range query in MFT.

if the range of the query parameter falls within the range of the current inner node. Owing to the monotonicity of biases, such comparison reveals the order between encrypted data thus determining whether an inner node is on the search path (line 5∼8). Once the bottom layer is reached, MFT compares each bit of the parameter's subfilter with the corresponding bit in a leaf node (line 2∼4). The requested record is only found when *all* bits match. This is because each subfilter exclusively corresponds to one record (**Definition 1**).

For range queries, MFT conducts two point searches on the left and right boundary respectively, and outputs all leaf nodes that lie between the two searched leaf nodes (line 16∼19).

Notably, MFT even tackles non-existent key search (line 9∼15) owing to its range design. Consider a range query example shown in Figure 4, after top-down comparisons, MFT reaches N_2 (because K_1's range is within N_2's range), but MFT finds that none of N_2's child nodes' ranges fully contain K_1's range. Thus, MFT learns that K_1 is a non-existent key (owing to the monotonicity property), retrieves the smallest data larger than the left boundary, and confirms N_{10} as the searched leaf node. Similarly, if the right boundary is non-existent, MFT searches the largest data smaller than the right boundary. In Figure 4's case, the right boundary K_2 is existent because MFT conducts subfilter comparisons at leaf nodes and finds that N_{12}'s subfilter exactly matches K_2's. Finally, all records situated between N_{10} and N_{12} (i.e., R_3∼R_5) are returned.

• **MFT update.** The update process in MFT involves locating the leaf node containing the target record using the search procedure described above. Once the leaf node is found, the corresponding record is updated accordingly. MFT handles node splitting and merging same as a traditional B$^+$-tree.

4.3. MFT-driven Query Workflow

So far, we have assumed that all EDBs follow our protocol in an honest-but-curious manner. Next, we show how to defend against a malicious attacker that can deviate from NEXUS's protocol through query authentication. Then we demonstrate how the authenticated data can be securely aggregated.

Query authentication. NEXUS enforces two authentication rules to detect manipulation attempts, including modifications, replays, and dropping (deletions) of query results.

• **Rule 1: freshness check.** This rule serves to detect any modification or replay attempts of query results. NEXUS employs MACs to check the freshness of query results. The MAC must satisfy two conditions: (1) a MAC tag is only valid for the latest update to prevent replay attacks, and (2) the MAC tag does not apply to other record's rangefilters. Thus, we compute the MAC over both the rangefilter and the version of the record. For each record update, the client additionally sends a MAC for that record, which is implemented by appending a 256-bit tag to the tail of the rangefilter.

For verification purpose, multiple parties store their MAC keys in the proxy. Whenever a record is updated, a client synchronizes the latest version of that record with the proxy; the version is never revealed to any EDBs. All clients within a party share an identical MAC key as they mutually trust each other. After receiving query results, the proxy recomputes the MAC and verifies the last 256 bits (MACs) in each rangefilter.

• **Rule 2: completeness check.** This rule serves to detect drops and deletions of query results by ensuring that the results align with the specified range and that the recomputed MFT root matches the original root. Upon receiving a range query, the EDB first searches for records within the specified range (r_l and r_r). It then generates a proof that includes a node's rangefilter to the immediate left of the lower-bound (R_l) and another rangefilter to the immediate right of the upper-bound (R_r) of the searched records. Additionally, the EDB retrieves the rangefilters of all left sibling nodes in the left search path and the rangefilters of all right sibling nodes in the right search path of the query.

Upon receiving the proof, the proxy verifies it through two aspects. Firstly, it checks that the range of R_l is smaller than the left boundary, i.e., $R_l.range < r_l.range$, and it also verifies that $R_r.range > r_r.range$. Secondly, the proxy recomputes the MFT root in a Merkle-tree fashion using the retrieved rangefilters of all query results and their siblings on the search paths. It then compares the recomputed rangefilter with the MFT root stored in the proxy, which was pre-stored by the client and synchronized for each update. The completeness of results is guaranteed only if two rangefilters are equal.

Figure 4 shows a completeness check example. Given a range query that searches for records in [K_1, K_2], it returns three leaf nodes $\{N_{10}, N_{11}, N_{12}\}$. For authentication, MFT returns N_9∼N_{13}: N_9 and N_{13} are returned to prove that no records in range [K_1, N_{10}) and (N_{12}, K_2] are omitted. Two additional sibling nodes (N_4 and N_7) in the search paths are also returned to recompute the root's rangefilter for comparison.

In sum, **Rule 1** provides protection against modifications or replays, and **Rule 2** detects any potential drops in query results. Together, they complete the query authentication protocol. We omit the discussion of point query authentication as it follows the same authentication protocol as range queries.

MFT-driven data aggregation. After authentication, NEXUS aggregates the homomorphic encrypted analytical results from multiple collaborative EDBs and generates decryptable results for the client. However, a security challenge arises as clients cannot directly share HE keys for decryption, even if they are mutually trusted. This is because, if a party withdraws from the joint query system (e.g., an enterprise exits the cooperation), the remaining parties cannot continue to decrypt the departing party's data. If the HE key is shared, the departing party must re-encrypt all its data using a new key and update its EDB accordingly, which is prohibitively expensive.

To remedy this issue, we propose an MFT-driven Secure Data Aggregation (SDA) protocol that securely transforms joint analytical results encrypted by multi-keys into decryptable results for clients, without compromising the confidentiality of HE data. SDA leverages a key weapon known as *key switch* [22], a homomorphic algorithm that enables the conversion of the encryption key of HE data to a new key without decrypting it. The key switch (KS), denoted by KS = (Enc, Switch), has the following syntax:

- KS.Enc(S,S') $\rightarrow tkn$. Enc encrypts key S using a new key S' and outputs a token tkn. tkn takes the form of (a*, A*) = $(a',\ a' \cdot S' + S + te \ mod \ q)$, a' is a random parameter, other variants are modular parameters in HE.
- KS.Switch(ctx,tkn) $\rightarrow ctx'$. By using tkn, Switch converts the encryption key of ctx from S to S' and outputs a new ciphertext $nctx = (-a \cdot a^*,\ A - a \cdot A^*)$.

However, relying solely on the KS is insufficient because there is no locality information on where to obtain the tokens required for switching. Fortunately, we can combine the power of MFT with KS to effectively locate query-relevant parties for obtaining the tokens and then aggregate HE data.

SDA operates through three steps. First, NEXUS searches the parameters on MFTs stored in the proxy to locate relevant EDBs that contain matching data. This step generates a list of parties P_i that owns the relevant EDBs. Second, the proxy signs P_i and returns it to the client, who then verifies the signature and sends a set of one-time keys S_i to each party. Upon receiving S_i, each party runs KS.Enc(S, S_i) to generate a secret token tkn_i based on S_i and its encryption key S. Each party then sends tkn_i to the proxy. Last, the proxy transforms each retrieved record from party P_i (i.e., ctx_i) with the party's token (i.e., tkn_i) using KS.Switch(ctx_i, tkn_i). The proxy then aggregates the key-switched data based on common attributes and returns the results to the client. The client can then use S_i to securely decrypt the HE-encrypted results.

4.4. Security and Performance Analysis

Formally, the following theorem captures NEXUS's security:

Theorem 1. *Given that the cryptographic hash functions are mathematically irreversible, and assuming that the ORE algorithm, homomorphic key switch algorithm, and MAC algorithm are all semantically secure, and provided that the underlying storage is maliciously secure, the* NEXUS *protocol for collaborative analytical queries is mathematically irreversible with integrity, in the presence of a malicious adversary* $\mathfrak{A}$.

The above theorem is established through the following two lemmas: **Lemma 1** demonstrates the infeasibility of an adversary $\mathfrak{A}$ to reverse-engineer NEXUS's index keys (i.e., rangefilters) used for searching; **Lemma 2** proves the effective detection of any integrity breaches conducted by $\mathfrak{A}$.

Lemma 1 (Confidentiality of rangefilter)**.** Given a rangefilter rf and a set of cryptographic hash functions H, it is computationally infeasible to find any input x such that $H(x) = rf$.

Proof. The proof is given in Appendix §A.3.

Lemma 2 (Integrity of query results)**.** Any manipulation attempts to EDBs' query results, including modifications, replays, dropping, and deletions can be detected in NEXUS.

Proof. The proof is given in Appendix §A.3.

High performance. NEXUS achieves the ideal logarithmic search complexity as illustrated below. The time complexity of traversing inner nodes using range comparison is $\mathcal{O}(\log n)$, where n is the total number of records; the time complexity of subfilter comparison at leaf nodes is $\mathcal{O}(m)$, where m is the length of subfilters. Consequently, the overall time complexity is $\mathcal{O}(\log n)$+$\mathcal{O}(m)$. As $m \ll n$, the resulting complexity of NEXUS's encrypted search is logarithmic to the database size.

5. Implementation

We implemented NEXUS with 5021 lines of C++ code on CryptDB [51], a modular framework for evaluating EDBs. For NEXUS's protocol messages, we utilized asynchronous RPC calls [72], and these messages were safeguarded via AES [36]. We modified CryptDB's proxy to spawn two new threads: one thread calls `MFT_dispatch()` to route queries to execute in query-dependent databases; the other thread verifies query results by calling `MFT_verify()`, and executes the SDA protocol by calling `MFT_aggregate()`.

Case study. We built two EDBs on MySQL [45] and MongoDB's NoSQL KV backend WiredTiger [16]. In both cases, we replaced MySQL's default B$^+$-tree and WiredTiger's default B-tree with NEXUS's MFT. This replacement is facilitated by NEXUS's unclustered design that separates the design of indexes from the underlying storage (§3.1). Notably, since WiredTiger maintains an LRU cache for the entire B-tree traversal path including both inner and leaf nodes' pages, to comply with such caching semantics, NEXUS also persisted the traversed MFT pages so that WiredTiger can cache them.

Optimization. Firstly, we adopted a lazy update strategy to improve query performance for deletions. Concretely, MFT refrains from executing an immediate update for each deletion, which involves costly root filter recomputation. Instead, MFT periodically updates itself at regular intervals for batches of deletions. This does not harm the correctness as we stored a supplementary bloom filter for deleted records (DBF). Each query will verify the presence of queried records in the DBF to ascertain whether they have been deleted. Secondly, we employed homomorphic batching [19] for key switching to maximize query throughput. Thirdly, we fine-tuned WiredTiger's LRU cache size for a higher cache hit ratio. Lastly, we losslessly compressed MFTs by storing only the subscripts of `"1"` bits in rangefilters, to preserve larger working sets of MFTs in memory and reduce time-consuming disk I/O operations.

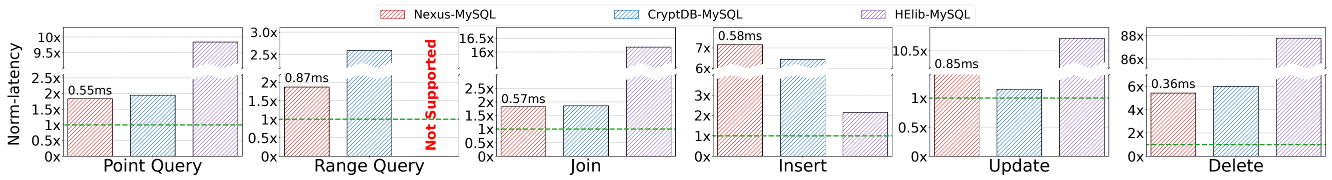


Figure 5: Normalized end-to-end latency of running TPC-C on all baselines in SQL instances. All systems' latency results are normalized to vanilla insecure MySQL (in green dashed line). Values on each red bar indicate NEXUS's average latency.

6. Evaluation

Testbed. All experiments were done on our cluster machines, each equipped with a 2.60GHz Intel E5-2690 V3 CPU, 64GB memory, 40Gbps NIC, and 24 cores. Each node, including clients, databases, and the proxy, was executed in a docker container. The average ping latency between the nodes was set at 0.17ms with the aid of Linux traffic control [6].

Baseline. We implemented a SQL instance using MySQL [45] and a NoSQL instance using WiredTiger [16]. We compared NEXUS with three baselines: CryptDB [51], HElibDB [5], and vanilla insecure databases (i.e., MySQL and WiredTiger). CryptDB and HElibDB are imperative HE databases. CryptDB cryptographically enables point queries with deterministic encryption (DET) [11] and range queries with order-preserving encryption (OPE) [4], with separate indexes. HElibDB utilizes Fermat's Little Theorem [21] to directly evaluate the equivalence of HE data but requires scanning the entire database.

For an apple-to-apple comparison, we expanded upon baselines in three ways. Firstly, we upgraded CryptDB's HE algorithm from partial HE to fully HE BGV [22] to enable arbitrary computation on ciphertexts as in NEXUS. Secondly, all baselines were extensively implemented on both SQL and NoSQL instances. Thirdly, we integrated NEXUS's core proxy components into CryptDB and HElibDB to support multi-party collaborative analytics as in NEXUS.

Workload and default setting. Since there is no standardized benchmark for evaluating collaborative SQL analytics on EDBs, for a fair comparison, we evaluated workloads from recent EDBs [51, 3, 76, 82], which includes the TPC-C benchmark [41] for relational databases and a customized workload for NoSQL key-value stores.

Specifically, we followed the multi-party deployment approach [75] by first partitioning TPC-C randomly and uniformly across databases based on warehouse ID, and assigned clients to parties based on their associated warehouse IDs. For customized NoSQL workloads, we simulated governmental statistics and medical workloads (as described in §2.2) by generating 2^{10} fixed-size keys (8 bytes) and values (64 bytes) in a manner similar to [80], and horizontally partitioned all key-value pairs across databases as in TPC-C.

To match the setup of existing multi-party query systems [75, 17], we equipped each party with two databases. Unless for scalability experiments, we simulated 10 parties. For the homomorphic setting, we employed HElib's BGV implementation [5], using a plaintext prime modulus of 29 and a ciphertext modulus of 1009 to compress ciphertexts with an expansion coefficient of 4. We ran each experiment for 60 seconds and collected the result in the middle 30s (i.e., 15s∼45s) to avoid the disturbance caused by system start-up and cool-down. We focus on the following questions:

§6.1 How efficient is NEXUS compared to baselines?
§6.2 Can NEXUS scale to more parties and larger datasets?
§6.3 Does NEXUS exhibit extensibility?
§6.4 What are the lessons we learned?

Distributed query latency (in milliseconds)

Systems (in SQL instance)	NEXUS		CryptDB	
Query type	PQ	RQ	PQ	RQ
P1 (***client***): Parameter transform	0.11	0.13	0.19	0.32
P2 (***proxy***): Query dispatch	0.12	0.18	0.06	0.1
P3 (***database***): Data lookup	4e-4	1e-3	2e-4	1e-3
P4 (***proxy***): Query authenticate	1e-3	1e-3	N/S	N/S
P5 (***proxy***): Key switch & aggregate	0.17	0.21	0.12	0.24
P6 (***client***): Result decrypt	0.15	0.35	0.21	0.52
End-to-end ($\sum P_i$)	0.55	0.87	0.58	1.18

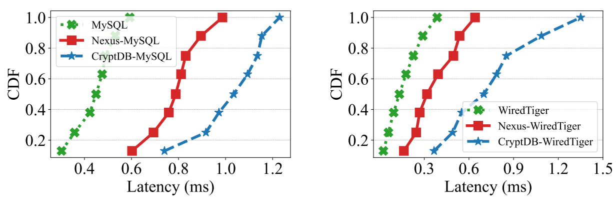
Table 2: Breakdown and comparison of query latency. **PQ** and **RQ** mean point and range query respectively. **P1** (***client***) means the first phase of parameter transformation occurs at the client, etc.

6.1. End-to-end Performance

We first evaluated the performance of NEXUS and baselines in a fault-free scenario where no integrity breaches occurred in the query results. The results presented in Figure 5 indicate that NEXUS demonstrated a lower latency (between 5.4x to 16.6x, excluding *insert* queries) compared to HElibDB. This is attributed to the use of MFT index by NEXUS, which provides logarithmic search complexity as opposed to HElibDB that requires linear scans to search for equivalent encrypted data, leading to reduced latency for all *read* queries.

Priority: read query latency. HElibDB achieved 3x to 3.3x lower *insert* latency compared to NEXUS and CryptDB. This is because HElibDB simply appends new records to its storage by trading off the linear search latency on *read* queries. In contrast, NEXUS and CryptDB both maintain sorted records in storage, leading to higher *insert* latency by first looking up an insert position via indexing before writing a new record to that position in the storage. Luckily, it is worth noting that in the joint query scenario of NEXUS, *read* could occur more frequently (refer to §2.2), and hence, we deem it acceptable to make such a performance trade-off, which has been a common practice in existing insecure databases [45, 16].

General capabilities with breakdown. On the TPC-C workload depicted in Figure 5, NEXUS achieved 6% to 28% lower



(a) Run in a SQL instance (b) Run in a NoSQL instance

Figure 6: Distributions of end-to-end range query latency running TPC-C and customized key-value workload on all baselines.

latency. For the representative queries (§2.2) running on the NoSQL workload with varying numbers of key-value pairs, NEXUS had 32% to 55% lower latency shown in Figure 8(a). Q2 exhibited higher latency than Q1 because Q2 involved range searches and a *sum* with aggregates while Q1 simply counted the total number. Both NEXUS and CryptDB supported point and range queries with logarithmic search complexity; interestingly, NEXUS still outperformed CryptDB in terms of both query latency (verified in Table 2) and throughput (shown in Figure 7), despite sharing the same complexity.

Table 2 shows the performance breakdown. CryptDB's primary overhead was observed to be in **P1** and **P6** for cryptographic operations. In contrast, NEXUS incurred additional latency in **P2** for subfilter comparison at leaf nodes (the time is covered by index traversal, analyzed in §4.4), and **P4** for query authentication. Nonetheless, the cryptographic overhead of CryptDB surpassed that of NEXUS, ultimately leading to the overall performance advantages exhibited by NEXUS.

We then evaluated the latency distribution by running range queries that accessed 20%∼80% records on all baselines, using various workloads. Figure 6 shows that NEXUS reduced the latency for all workloads in comparison to CryptDB. The latency distribution was affected by the length of queries, where longer query ranges led to higher latency of lookup, key switch, aggregation, and decryption. Notably, insecure databases do not require key switch and decryption. HElibDB was not evaluated as it does not support range queries.

Performance on mixed queries. Figure 7 shows that NEXUS achieved 1.31x to 1.88x higher throughput over CryptDB on the customized workload with mixed point and range queries. This improvement is mainly because CryptDB's design necessitates frequent shuffling between the point search index and range search index, whereas NEXUS uniformly tackled both query types through a single MFT index abstraction. The throughput gain is also bolstered by our homomorphic implementation that employs the batching strategy (§5).

Robustness under attacks. Next, to assess the robustness of NEXUS, we simulated attacks by randomly manipulating half of the queries running on the KV workload, involving either modifying, replaying, or dropping (deleting) query results. The stable effective throughput in Figure 8(b) demonstrates NEXUS's 100% detection rate, whereas other baselines with no defenses had significant performance drops. This is due to NEXUS's two-step integrity check (§4.3): the MAC check

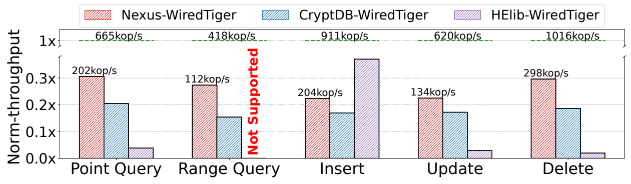
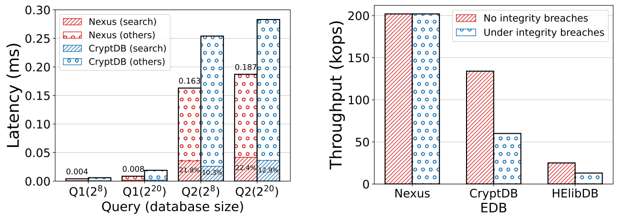


Figure 7: Normalized throughput in NoSQL instances. The green dashed line is vanilla (insecure) WiredTiger's averaged throughput.



(a) Example query in §2.2 (b) Effective throughput

Figure 8: Comparison of example query latency and effective query throughput under integrity breaches.

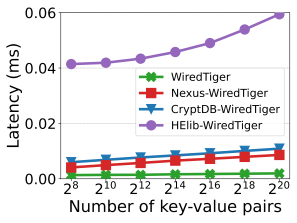
detects any modifications or replays on versioned data; the MFT check detects any drops or deletions of query results.

In sum, NEXUS presents a robust solution for enabling general queries among multiple collaborative parties, with low latency and high throughput. NEXUS favors *read* query performance and tolerates moderate write performance downgrades. NEXUS is most suitable for applications that prioritize *read* performance and require end-to-end security, which is common in collaborative scenarios such as financial statistics across banks [79] and medical studies across hospitals [42].

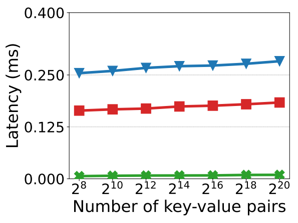
6.2. Scalability

Scale to larger databases. We evaluated baselines with varying database sizes on the KV workload. As depicted in Figure 9, NEXUS's latency scales logarithmically, matching our complexity analysis in §4.4. For point queries, both NEXUS and CryptDB incurred moderate overhead when compared to the insecure WiredTiger, while HElibDB had significantly higher latency due to its linear scanning approach. For range queries, NEXUS achieved a greater performance advantage than baselines compared to point queries because NEXUS's membership-based indexing is more efficient than prior cryptographic approaches (as also evident in Table 2). HElibDB was not evaluated because it is designed solely for point searches.

Scale to more parties. We varied the number of collaborative parties in NEXUS, ranging from 2 to 10, which is considered sufficient for collaborative analytics in real-world scenarios [75]. Figure 10 depicts NEXUS's throughput-latency (99th% tail latency) performance on the KV workload. As the number of parties increased, both NEXUS's throughput and latency increased. Note that the latency gap between different numbers of parties was more pronounced in range queries compared to point queries, because the range queries typically searched, verified, and aggregated more data, leading to increased query processing time.

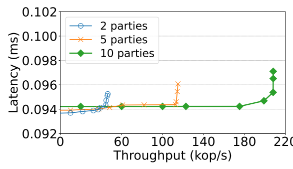


(a) Point Query

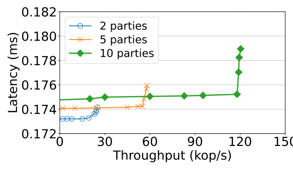


(b) Range Query

Figure 9: Performance comparison with varying database sizes.



(a) Point Query



(b) Range Query

Figure 10: Performance comparison with varying numbers of parties.

6.3. Extensibility Study

One of the fundamental capabilities of Nexus lies in its use of the MFT secure index to dispatch general queries to execute in query-dependent HE databases. This query routing capability is essential for a secure query system and enables Nexus to extend its functionality to work seamlessly with TEE databases [3, 64, 53]: Nexus can be extended to allow collaborative parties to outsource their data to TEE databases and perform joint queries alongside other parties' HE databases. Concretely, in this study, the clients initialized MFTs for both HE and TEE databases and uploaded MFTs to the proxy for query dispatching. When dealing with queries in TEE databases, the workflow remained unchanged, involving parameter decryption within TEE databases, searching plaintexts in a TEE-shielding B$^+$-tree, and ultimately verifying plaintext results within the TEE (§2.1).

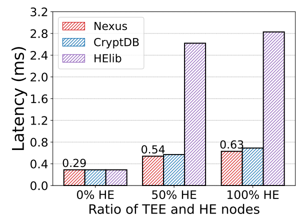
To validate this extensibility, we conducted experiments by varying the proportion of HE and TEE databases in our default ten-party setting and tested the query latency. We ran TEE databases using the open-sourced Azure EdgelessDB [3], which equips an TEE-shielded B$^+$-tree with optimized index performance. Figure 11 confirms the feasibility of running a combination of HE and TEE databases for collaborative query analytics. Furthermore, the comparison between Figure 11(a) and Figure 11(b) highlights Nexus's primary advantage in efficiently supporting range queries when compared to baselines, corroborating the findings from previous experiments.

6.4. Lessons Learned

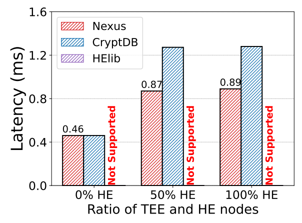
Cryptographic hash-based security. Cryptographic hash has been widely adopted in security-critical applications such as password managers [9, 10] and phishing detectors [38], showcasing its effectiveness in protecting sensitive information. Nexus builds the MFT secure index using cryptographic hash functions like SHA-256 and inherits the irreversible security [57], in the sense that the encrypted data cannot be reversed to plaintexts even with the knowledge of the hash functions. This design choice enables practical performance and rich query capabilities with query integrity, all achieved through MFT. Unlike previous research (e.g., CryptDB and HElibDB) that prioritizes a stronger confidentiality guarantee of IND-CPA [2], which often sacrifices query capability or efficiency and overlooks query integrity, the Nexus design strikes a balance between practicality and security. As such, it is well-suited to connect EDBs for collaborative query analytics [27, 42, 39, 79] in a malicious threat environment.

Dismissing an alternative design. One may think of using solely monotonic bias with ORE for building an encrypted index, which poses two problems. Firstly, this approach limits query capability to range queries, akin to CryptDB's OPE solution. Secondly, it fails to support query authentication for detecting integrity breaches. In contrast, Nexus enables both data confidentiality and query integrity in a unified manner.

Limitations. Nexus has two limitations. Firstly, its security is *not* perfect as it does not hide side-channel information related to the specific type of computation performed by queries, such as equality comparisons and sorting operations. This is an inherent issue in most encrypted databases that aim to achieve practicality without resorting to expensive oblivious algorithms [53, 3, 51]. Secondly, Nexus's MFT secure index is built on a single searchable attribute, making it unsuitable for databases that search over multiple attributes such as graph databases [7, 31]. For these databases, while Nexus is capable of handling disjunctive queries by searching each attribute independently, Nexus cannot handle conjunctive queries without integrating MFT with multidimensional indexes such as *k*-d tree [81], which is an interesting direction for future work.



(a) Point Query



(b) Range Query

Figure 11: Extensibility study of running different ratios of HE and TEE databases in SQL instances distributedly.

7. Conclusion

We present Nexus, the first secure query system that supports general collaborative SQL analytics on EDBs, providing both data confidentiality and query integrity guarantees. Nexus leverages the new Merkle rangeFilter Tree (MFT) secure index to efficiently dispatch encrypted queries to query-dependent EDBs, authenticate, and aggregate query results. Extensive results on both SQL and NoSQL instances shown that Nexus is secure, highly efficient, scalable, and extensible compared to baselines. Nexus is open-sourced and its code is released on `github.com/2024asplos405/Nexus`.

References

[1] Jeddak project. https://github.com/bytedance-jeddak/jeddak.

[2] *Web resource: https://en.wikipedia.org/wiki/Ciphertext indistinguishability*, 1997.

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

[4] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574, 2004.

[5] Ishtiyaque Ahmad, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Pantheon: Private retrieval from public key-value store. *Proceedings of the VLDB Endowment*, 16(4):643–656, 2022.

[6] Werner Almesberger et al. Linux network traffic control—implementation overview, 1999.

[7] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1–39, 2008.

[8] David W Archer, Dan Bogdanov, Yehuda Lindell, Liina Kamm, Kurt Nielsen, Jakob Illeborg Pagter, Nigel P Smart, and Rebecca N Wright. From keys to databases—real-world applications of secure multi-party computation. *The Computer Journal*, 61(12):1749–1771, 2018.

[9] Sareh Assiri and Bertrand Cambou. Homomorphic password manager using multiple-hash with puf. In *Advances in Information and Communication: Proceedings of the 2021 Future of Information and Communication Conference (FICC), Volume 1*, pages 772–792. Springer, 2021.

[10] Andrey Belenko and Dmitry Sklyarov. “secure password managers” and “military-grade encryption” on smartphones: Oh, really? *Blackhat Europe*, page 56, 2012.

[11] Mihir Bellare, Marc Fischlin, Adam O’Neill, and Thomas Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. In *Annual International Cryptology Conference*, pages 360–378. Springer, 2008.

[12] Carole Bernard, Ludger Rüschendorf, Steven Vanduffel, and Jing Yao. How robust is the value-at-risk of credit risk portfolios? *The European Journal of Finance*, 23(6):507–534, 2017.

[13] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[14] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 563–594. Springer, 2015.

[15] Marcus Brandenburger, Christian Cachin, Matthias Lorenz, and Rüdiger Kapitza. Rollback and forking detection for trusted execution environments using lightweight collective memory. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 157–168. IEEE, 2017.

[16] Rupali Chopade and Vinod Pachghare. Mongodb indexing for performance improvement. In *ICT Systems and Sustainability*, pages 529–539. Springer, 2020.

[17] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. Dory: An encrypted search system with distributed trust. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1101–1119, 2020.

[18] Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica. Waldo: A private time-series database from function secret sharing. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2450–2468. IEEE, 2022.

[19] Yarkın Doröz, Gizem S Çetin, and Berk Sunar. On-the-fly homomorphic batching/unbatching. In *International Conference on Financial Cryptography and Data Security*, pages 288–301. Springer, 2016.

[20] Mohammad J Emami, Ali R Tavakoli, Hossein Alemzadeh, Farzad Abdinejad, Gholamhossain Shahcheraghi, Mohammad A Erfani, Kamran Mozafarian, Saeed Solooki, Sorena Rezazadeh, Ahmad Ensafdaran, et al. Strategies in evaluation and management of bam earthquake victims. *Prehospital and disaster medicine*, 20(5):327–330, 2005.

[21] Fermat. Fermat’s last theorem. *Web resource: https://en.wikipedia.org/wiki/Fermat27sLastTheorem*, 1997.

[22] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P Smart. Ring switching in bgv-style homomorphic encryption. In *International Conference on Security and Cryptography for Networks*, pages 19–37. Springer, 2012.

[23] Krzysztof Goczyła and Janusz Cielatkowski. Optimal routing in a transportation network. *European Journal of Operational Research*, 87(2):214–222, 1995.

[24] Oded Goldreich. *Foundations of cryptography: Volume 1, Basic tools*. 2001.

[25] Vipul Goyal, Adam O’Neill, and Vanishree Rao. Correlated-input secure hash functions. In *Theory of Cryptography Conference*, pages 182–200. Springer, 2011.

[26] Paul Grubbs. On deploying property-preserving encryption. *Real World Cryptography*, 2016.

[27] CodeBlue: Sensor networks for medical care, 2008. http://www.eecs.harvard.edu/mdw/ proj/codeblue/.

[28] James L Hieronymus. Ascii phonetic symbols for the world’s languages: Worldbet. *Journal of the International Phonetic Association*, 23:72, 1993.

[29] Ilia Iliashenko and Vincent Zucca. Faster homomorphic comparison operations for bgv and bfv. *Proceedings on Privacy Enhancing Technologies*, 2021(3):246–264, 2021.

[30] Scott R. Intel. Sgx 2.0 (scalable sgx).

[31] Borislav Iordanov. Hypergraphdb: a generalized graph database. In *International conference on web-age information management*, pages 25–36. Springer, 2010.

[32] KS Jaiswal, David J Wald, Paul S Earle, Keith A Porter, and Mike Hearne. Earthquake casualty models within the usgs prompt assessment of global earthquakes for response (pager) system. In *Human casualties in earthquakes: progress in modelling and mitigation*, pages 83–94. Springer, 2010.

[33] Shivam Kalra, Junfeng Wen, Jesse C Cresswell, Maksims Volkovs, and Hamid R Tizhoosh. Proxyfl: decentralized federated learning through proxy model sharing. *arXiv preprint arXiv:2111.11343*, 2021.

[34] Shivam Kalra, Junfeng Wen, Jesse C Cresswell, Maksims Volkovs, and HR Tizhoosh. Decentralized federated learning through proxy model sharing. *Nature communications*, 14(1):2899, 2023.

[35] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*, pages 258–274. Springer, 2013.

[36] Emilia Käsper and Peter Schwabe. Faster and timing-attack resistant aes-gcm. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–17. Springer, 2009.

[37] Sara Kathleen Geale. The ethics of disaster management. *Disaster Prevention and Management: an international journal*, 21(4):445–462, 2012.

[38] Mahmoud Khonji, Youssef Iraqi, and Andrew Jones. Phishing detection: a literature survey. *IEEE Communications Surveys & Tutorials*, 15(4):2091–2121, 2013.

[39] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973, 2021.

[40] Yitshak Kreiss, Ofer Merin, Kobi Peleg, Gad Levy, Shlomo Vinker, Ram Sagi, Avi Abargel, Carmi Bartal, Guy Lin, Ariel Bar, et al. Early disaster response in haiti: the israeli field hospital experience, 2010.

[41] Scott T Leutenegger and Daniel Dias. A modeling study of the tpc-c benchmark. *ACM Sigmod Record*, 22(2):22–31, 1993.

[42] Dong Li, Xiaofeng Liao, Tao Xiang, Jiahui Wu, and Junqing Le. Privacy-preserving self-serviced medical diagnosis scheme based on secure multi-party computation. *Computers & Security*, 90:101701, 2020.

[43] Murali Mani, Kinnari Shah, and Manikanta Gunda. Enabling secure database as a service using fully homomorphic encryption: Challenges and opportunities. *arXiv preprint arXiv:1302.2654*, 2013.

[44] Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, Markus Huber, and Edgar Weippl. Dark clouds on the horizon: Using cloud storage as attack vector and online slack space. In *20th USENIX Security Symposium (USENIX Security 11)*, 2011.

[45] AB MySQL. Mysql, 2001.

[46] IBM Homomorphic Encryption News. Top brazilian bank pilots privacy encryption quantum computers can’t break.

[47] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos Keromytis, and Steve Bellovin. Blind seer: A scalable private dbms. In *2014 IEEE Symposium on Security and Privacy*, pages 359–374. IEEE, 2014.

[48] Zeshun Peng, Yanfeng Zhang, Qian Xu, Haixu Liu, Yuxiao Gao, Xiaohua Li, and Ge Yu. Neuchain: a fast permissioned blockchain system with deterministic ordering. *Proceedings of the VLDB Endowment*, 15(11):2585–2598, 2022.

[49] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. Arx: A strongly encrypted database system. *IACR Cryptol. ePrint Arch.*, 2016:591, 2016.

[50] Rishabh Poddar, Sukrit Kalra, Avishay Yanai, Ryan Deng, Raluca Ada Popa, and Joseph M Hellerstein. Senate: a {Maliciously-Secure}{MPC} platform for collaborative analytics. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2129–2146, 2021.

[51] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the twenty-third ACM symposium on operating systems principles*, pages 85–100, 2011.

[52] Behandelt PostgreSQL. Postgresql. *Web resource: http://www. PostgreSQL. org/about*, 1996.

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

[54] Lilian SC Pun-Cheng. An interactive web-based public transport enquiry system with real-time optimal route computation. *IEEE Transactions on Intelligent Transportation Systems*, 13(2):983–988, 2012.

[55] Ji Qi, Xusheng Chen, Yunpeng Jiang, Jianyu Jiang, Tianxiang Shen, Shixiong Zhao, Sen Wang, Gong Zhang, Li Chen, Man Ho Au, et al. Bidl: A high-throughput, low-latency permissioned blockchain framework for datacenter networks. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 18–34, 2021.

[56] Michael O Rabin. Probabilistic algorithms in finite fields. *SIAM Journal on Computing*, 9(2):273–280, 1980.

[57] Dian Rachmawati, JT Tarigan, and ABC Ginting. A comparative study of message digest 5 (md5) and sha256 algorithm. In *Journal of Physics: Conference Series*, volume 978, page 012116. IOP Publishing, 2018.

[58] Addanki Purna Ramesh, AVN Tilak, and AM Prasad. An fpga based high speed ieee-754 double precision floating point multiplier using verilog. In *2013 International Conference on Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT)*, pages 1–5. IEEE, 2013.

[59] Kalle Johannes Rose. The problem of regulating the easy way out–eu money laundering regulation. *Journal of Money Laundering Control*, 22(4):666–677, 2019.

[60] Mehdi Saeedi, Morteza Saheb Zamani, and Mehdi Sedighi. A library-based synthesis methodology for reversible logic. *Microelectron. J.*, 41(4):185–194, April 2010.

[61] Mario Schkolnick. Secondary index optimization. In *Proceedings of the 1975 ACM SIGMOD international conference on Management of data*, pages 186–192, 1975.

[62] AMD Sev-Snp. Strengthening vm isolation with integrity protection and more. *White Paper, January*, 53:1450–1465, 2020.

[63] Tianxiang Shen, Jianyu Jiang, Yunpeng Jiang, Xusheng Chen, Ji Qi, Shixiong Zhao, Fengwei Zhang, Xiapu Luo, and Heming Cui. Daenet: Making strong anonymity scale in a fully decentralized network. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[64] Yuanyuan Sun, Sheng Wang, Huorong Li, and Feifei Li. Building enclave-native storage engines for practical encrypted databases. *Proceedings of the VLDB Endowment*, 14(6):1019–1032, 2021.

[65] Ken'ichi Tanaka. The kobe earthquake: the system response. a disaster report from japan. *European Journal of Emergency Medicine*, 3(4):263–269, 1996.

[66] Christine Tex, Martin Schäler, and Klemens Böhm. Towards meaningful distance-preserving encryption. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2018.

[67] Dwen-Ren Tsai, Allen Y Chang, Sheng-Chieh Chung, and You Sheng Li. A proxy-based real-time protection mechanism for social networking sites. In *44th Annual 2010 IEEE International Carnahan Conference on Security Technology*, pages 30–34. IEEE, 2010.

[68] Eugene Tsyrklevich and Bennet Yee. Dynamic detection and prevention of race conditions in file accesses. In *12th USENIX Security Symposium (USENIX Security 03)*, 2003.

[69] Francisco Vazquez, Benjamin M Tabak, and Marcos Souto. A macro stress test model of credit risk for the brazilian banking sector. *Journal of Financial Stability*, 8(2):69–83, 2012.

[70] Benjamin Vogel and Jean-Baptiste Maillart. *National and international anti-money laundering law: developing the architecture of criminal justice, regulation and data protection*. Intersentia, 2020.

[71] Miao Wang, Hangguan Shan, Rongxing Lu, Ran Zhang, Xuemin Shen, and Fan Bai. Real-time path planning based on hybrid-vanet-enhanced transportation system. *IEEE Transactions on vehicular technology*, 64(5):1664–1678, 2014.

[72] Xingwei Wang, Hong Zhao, and Jiakeng Zhu. Grpc: A communication cooperation mechanism in distributed systems. *ACM SIGOPS Operating Systems Review*, 27(3):75–86, 1993.

[73] Yilei Wang and Ke Yi. Secure yannakakis: Join-aggregate queries over private data. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1969–1981, 2021.

[74] Zhiyong Wang, Ruijie Hu, Tao Yu, and Chunyong Chen. Design and implementation of a database client blacklist mechanism. In *2019 International Conference on Computing, Communications and Intelligence Systems (ICCCIS)*, pages 71–74. IEEE, 2019.

[75] Pengfei Wu, Jianting Ning, Jiamin Shen, Hongbing Wang, and Ee-Chien Chang. Hybrid trust multi-party computation with trusted execution environment.

[76] Yu Xia, Xiangyao Yu, Matthew Butrovich, Andrew Pavlo, and Srinivas Devadas. Litmus: Towards a practical database management system with verifiable acid properties and transaction correctness. In *Proceedings of the 2022 International Conference on Management of Data, Philadelphia, PA, USA*, pages 12–17, 2022.

[77] Liang Xiao, Dongjin Xu, Caixia Xie, Narayan B Mandayam, and H Vincent Poor. Cloud storage defense against advanced persistent threats: A prospect theoretic study. *IEEE Journal on Selected Areas in Communications*, 35(3):534–544, 2017.

[78] Tao Yang, Jinming Wang, Weijie Hao, Qiang Yang, and Wenhai Wang. Hybrid cloud-edge collaborative data anomaly detection in industrial sensor networks. *arXiv preprint arXiv:2204.09942*, 2022.

[79] Statistics Canada. Zachary Zanussi. Privacy preserving technologies part two: Introduction to homomorphic encryption.

[80] Yuhong Zhong, Haoyu Li, Yu Jian Wu, Ioannis Zarkadas, Jeffrey Tao, Evan Mesterhazy, Michael Makris, Junfeng Yang, Amy Tai, Ryan Stutsman, et al. {XRP}:{In-Kernel} storage functions with {eBPF}. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 375–393, 2022.

[81] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time kd-tree construction on graphics hardware. *ACM Transactions on Graphics (TOG)*, 27(5):1–11, 2008.

[82] Wenchao Zhou, Yifan Cai, Yanqing Peng, Sheng Wang, Ke Ma, and Feifei Li. Veridb: An sgx-based verifiable database. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2182–2194, 2021.

A. Appendices

A.1. Background: Homomorphic Encryption

To avoid trusting one specific hardware vendor of trusted hardwares, some organizations are seeking a cryptographic root-of-trust from homomorphic encryption (HE), and are willing to pay for the deficiencies of using HE-based solutions, especially fully HE schemes like BGV [22]. BGV allows arbitrary aggregation (e.g., `sum`, `avg`) on encrypted data (with noises) without requiring a secret key for decryption, and the computation result can be fully revealed by the owner of the secret key [22]. BGV uses *key switch* algorithm to reduce ciphertext's noises, which is leveraged by NEXUS to aggregate multi-party data encrypted by distinct secret keys (§4.3).

A.2. MFT Construction Algorithm

Algorithm 2: MFT Construction

```
/*Initialize MFT on the client side, update MFT on the database side.*/
Struct {
    Rangefilter rft                    ▷ Subfilter with confidential bias (Def 2)
    Node parent                        ▷ Parent node pointer
} MFTNode;
Function CreateLeaf() do
    ▽ Initialize bottom-layer rangefilters
    ...                                ▷ Procedure in §4.2
    return leaf[]
Function CreateInner($Node_i$, $Node_j$) do
    ($inner.left$, $inner.right$) ← ($Node_i$, $Node_j$)
    ($Node_i.parent$, $Node_j.parent$) ← $inner$
    ▽ 'Union' the child ranges for traversal
    $inner.rng \leftarrow Node_i.rng \cup Node_j.rng$
    ▽ 'OR' the child subfilters for integrity §4.3
    $inner.rft \leftarrow Node_i.rft$ or $Node_j.rft$
    return inner
Function CreatMFTIndex() do
    $le$[]← CreateLeaf(), $in$[]← ∅, $ptr \leftarrow 0$
    while $in.length > 2$ do
        $n$[]← ∅, $ptr \leftarrow 0$
        while $ptr < in.length$ do
            if $ptr = in.length - 1$ then
                $n$.insert(CreateInner(in[ptr],in[ptr-1].pn))
            else
                $n$.insert(CreateInner(in[ptr],in[ptr+1]))
            $ptr \leftarrow ptr + 2$
        $in \leftarrow n$
    return root ← CreateInner($in$[0], $in$[1])        ▷ MFT Entry
```

A.3. Proof Sketch of Security

Lemma 1 (Confidentiality of rangefilter)**.** Given a rangefilter rf and a set of cryptographic hash functions H, it is computationally infeasible to find any input x such that $H(x) = rf$.

Proof. We prove this lemma from two aspects. Firstly, the irreversibility of the subfilter in a rangefilter is a direct result of the one-way property of cryptographic hash functions, which ensures that a computationally-bounded adversary cannot reverse the plaintexts even with the knowledge of the set of used hash functions [56, 24]. Secondly, the confidentiality of the bias in a rangefilter is ensured by the randomness of the semantically secure ORE, as the input version number is never revealed to an adversary $\mathfrak{A}$ (§4.2). By combining these properties, $\mathfrak{A}$ can, at most, derive the index key from the subfilter using exhaustive search (because reverse-engineering a semantically secure bias is more difficult than reverse-engineering a irreversible subfilter). Therefore, it is computationally infeasible to find any input x such that $H(x) = rf$.

Lemma 2 (Integrity of query results)**.** Any manipulation attempts to EDBs' query results, including modifications, replays, dropping, and deletions can be detected in NEXUS.

Proof. We prove this lemma from two authentication rules (§4.3). Firstly, let S be the set of search results, l and r be the query range boundary, NEXUS's alignment rule states that $max\{x \in S : x < l\} \notin S$ and $min\{x \in S : x > r\} \notin S$, ensuring that no records within the range of query parameters are omitted; also, any modification or drop in results would be detected by failing to reconstruct the root rangefilter. Secondly, let $\tau = \text{MAC}(S, mid, v)$ be the MAC of S with the secret MFT ID mid, version number v held by the non-colluded proxy, the freshness rule ensures that any reply with a different τ can be detected. This directly inherits the security of the MAC scheme. By combining the two rules, any modifications, dropping, deletions, or replays conducted by $\mathfrak{A}$ can be detected with the help of MFT, completing the proof.

To sum, **Lemma 1** proves that $\mathfrak{A}$ is infeasible to reverse the NEXUS's index keys and **Lemma 2** proves that any integrity breaches conducted by $\mathfrak{A}$ can be detected in NEXUS. Together these complete the proof of **Theorem 1**.

A.4. NEXUS's Storage Model

Recall that NEXUS operates under the assumption that the underlying storage engine is secure. This security ensures that an EDB server can consistently and reliably retrieve the correct data version from the underlying storage by detecting and preventing rollback and fork attacks. NEXUS relies on the underlying storage to provide essential functionalities for data retrieval. Specifically, we assume a table T with n columns $CL = \{c_0, c_1, ..., c_{n-1}\}$ and is indexed on the primary key c_0 (i.e., $PK = c_0$). A record r can be depicted as $[E(k), E(v_1), ..., E(v_{n-1})]$ where k is the value for c_0. The storage engine should provide the following API:

- EqualGet(E(k)): Retrieve the row r from T given a key k.
- RangeGet(E(k_s),E(k_e)): Retrieve all rows $\{r\}$ from T whose keys are between E(k_s) and E(k_e).
- FullGet(): Retrieve all rows $\{r\}$ from T.
- Put(r): Insert (or update) a row r to T if its key k is non-existent (or existent).
- Delete(E(k)): Delete the row r from T given a key k.

A.5. Generality of NEXUS

Is NEXUS general? We illustrate NEXUS's generality by discussing two NEXUS's core components: MFT confidential index and SDA protocol. First, MFT is general to database types including both relational and NoSQL databases, because we assume a widely-adopted unclustered architecture where index and other components (e.g., storage) are decoupled (§3.1) and MFT only substitutes the original index; MFT is also general to attribute types (integer, real value, and string): as MFT so far works on the integer attribute, we can transform real values into integers using IEEE 754 floating point number [58] and transform string values using ASCII character [28] by encoding any length-l ASCII string with integers $[0,\ 2^{7 \cdot l} - 1]$. Second, SDA is general as well, because SDA uses the generic MFT for searching, and aggregates data with key switch, a generic tool in common HE algorithms to reduce HE data's cumulative noises [22, 29].