BIRDB: A Metadata-secure and High-performance Encrypted Database using Compact Encrypted Search

Anonymous Submission #xxx

***Abstract*—The growing concerns about the confidentiality of data outsourced to untrusted cloud storage have prompted notable progress in *Encrypted Databases* (EDB). EDBs store encrypted data and allow for searches as traditional insecure databases while hiding the metadata of access patterns during searches. However, achieving both efficient encrypted search and metadata protection remains a challenge. Some EDBs construct cryptographic indexes on property-preserving encrypted data to enable logarithmic searches, but their performance downgrades as the index size substantially increases and also disregard metadata protection. Other approaches resort to scanning the entire dataset to hide metadata, leading to significant overhead during encrypted searches.**

We present BIRDB, the first efficient and compact EDB with metadata protection. At the core of BIRDB lies a new *boolean information retrieval* technique that structures the database in a logarithmic-complexity search data structure (index) and performs the search obliviously over the index. The boolean nature of the index provides two crucial advantages: (1) lossless index compression through linear approximation and (2) seamless integration with efficient private information retrieval protocols. This integration allows computationally expensive query-independent computations to be shifted to an offline phase, leading to sublinear computation during the query-specific online phase, all while ensuring metadata protection. Extensive evaluations show that BIRDB reduced up to 40.7% query latency and achieved up to 45x higher throughput and provided robust metadata protection compared to existing notable approaches.

I. INTRODUCTION

The popularity of cloud computing has driven widespread adoption of cloud storage services such as DropBox [28] and Onedrive [71]. However, these services have raised grave security concerns due to their processing of user data in plaintext. Recent incidents, including a data breach exposing the personal information of 237,000 US government employees [1], the leak of Samsung's software source code when the employees uploaded it to ChatGPT [7], and accusations against TikTok for compromising the data of over 150 million US citizens stored in the cloud [8], highlight the serious data security risks associated with cloud storage.

Recently, *Encrypted Databases* (EDBs) [24], [47], [58], [68] have emerged as a dependable solution for enhancing the security of cloud storage. EDBs store user data in an encrypted form without knowing the decryption keys, ensuring data confidentiality even against privileged database administrators and external attackers (i.e., *adversaries*). To enable efficient query processing on the encrypted data, EDBs require efficient and compact search indexes, comparable to the binary search tree (BST) used in traditional insecure databases [50]. Furthermore, EDBs must employ oblivious algorithms to protect search metadata, preventing adversaries from extracting sensitive information from index access footprints during searches [21].

Figure 1 illustrates how an adversary can exploit search access patterns to extract plaintext information and emphasizes

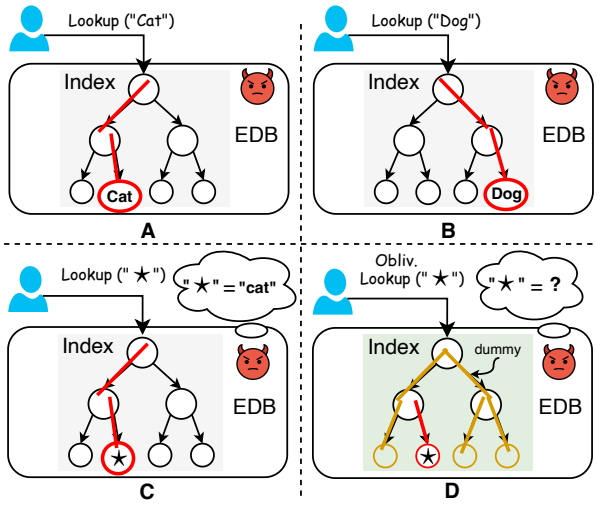

Fig. 1: Compared to an EDB that uses only end-to-end encryption shown in **A**∼**C**, a metadata-hiding EDB in **D** disguises intended index operations with dummy operations, preventing adversaries from learning the metadata (i.e., index access footprints during searches).

the advantages of metadata protection [21], [24]. Consider an EDB that employs a BST search index, responsible for mapping encrypted keywords to their corresponding satellite data (e.g., address). The adversary passively monitors a series of plaintext queries and their associated index traversal paths (**A**∼**B**). By repeating this process, the adversary can deduce the keyword associated with each index path. Consequently, when a user submits a confidential query with an encrypted search keyword ('⋆' in **C**), the adversary can infer the keyword based on the specific index node accessed. Nevertheless, an EDB implementing metadata protection utilizes oblivious algorithms that include dummy accesses (as depicted in **D**), effectively concealing the actual search access patterns and mitigating the vulnerability associated with search pattern exposure.

A commonly used approach is Oblivious RAM (ORAM), which is typically structured as a BST and employs dummy accesses to cryptographically conceal the actual path traversed during searches [27], [31], [64]. The utilization of a tree-based data structure allows ORAM to achieve a polylogarithmic asymptotic complexity relative to the size of the database. Such a theoretically low time complexity is comparable to that of BSTs used in insecure databases.

Unfortunately, ORAM is not suitable for our particular context due to two main reasons. Firstly, the one-phase design of ORAM requires repeatedly executing the entire set of oblivious algorithms for each online search [25], [56], [64]. Secondly, ORAM's cryptographic nature renders it uncompressible while maintaining searchability. This is due to the inherent randomness introduced during cryptographic operations. Consequently, when dealing with a large-scale ORAM-based index containing a significant amount of data, a considerable portion of it needs to be transferred from fast primary storage

to slower secondary storage [24], [29]. This reliance on costly I/O operations to secondary storage becomes a bottleneck and leads to severe performance degradation (proved in §VI-A).

As an alternative to ORAM, OO-PIR [47] proposes a two-phase private information retrieval (PIR) design. It preprocesses the database into multiple chunks, where each online query retrieves a random chunk to conceal the specific item being fetched. While OO-PIR moves costly oblivious operations (chunk division) to an offline phase, it still requires storing the entire set of chunks and linearly scanning each chunk during online queries. HELibDB [22] eliminates the need for index construction by utilizing Fermat's Little Theorem [70] to assess the equivalence of encrypted data, thus achieving compactness compared to OO-PIR. However, such direct evaluation requires linearly scanning all encrypted data thus hampers performance.

To enhance encrypted search efficiency and expedite the aforementioned metadata-hiding solutions, one approach is to utilize property-preserving encryption algorithms [9], [15] to construct cryptographic indexes on the encrypted data. These algorithms preserve the determinism and order of the encrypted data, thereby enabling the construction of BSTs with logarithmic search complexity. However, this cryptographic index is not suitable due to its incompatibility with the PIR process in existing metadata-hiding solutions. Moreover, the cryptographic index suffers from the same limitation of uncompressibility, further restricting its performance.

Overall, the challenge of striking a balance between efficiency (logarithmic search complexity) and compactness (compressible encrypted index) while ensuring metadata protection in an EDB remains an open problem. We believe the root cause is that existing studies use mutually isolated cryptographic primitives to address individual challenges, rather than seeking a unified solution that can rule all aspects.

Drawing inspiration from Boolean Information Retrieval protocols developed for document retrieval [11], [43], we made a key observation that the boolean form of encrypted data can provide significant *membership* information, which can assist in achieving all three desired properties simultaneously. Firstly, the boolean form enables a new encrypted search concept by replacing traditional numerical comparisons in BSTs with membership tests. Secondly, the boolean characteristic of the data allows for the use of linear approximation to compress the encrypted indexes within constant space occupancy [40], [44], overcoming the inherent uncompressibility of cryptographic indexes in prior work. Lastly, the boolean form of the data aligns well with the two-phase PIR process [34], [47], where the database is preprocessed, and users ultimately recover results using boolean operations.

This paper presents BIRDB[1], the first encrypted database with metadata protection that achieves efficiency and compactness. The key weapon of BIRDB is a new compressible encrypted index called BINDEX. Unlike prior cryptographic approaches that build uncompressible indexes due to random noise introduced during encryption, BINDEX utilizes efficient one-way functions (OWF) such as SHA-256 [33]. These OWFs transform the data into irreversible ciphertexts in boolean form and then BIRDB organizes them in a membership-preserving

[1]BIRDB stands for **B**oolean **I**nformation **R**etrieval based E**DB**.

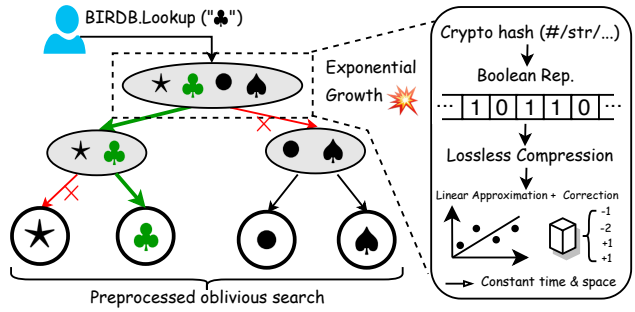

Fig. 2: BIRDB's fundamental concept revolves around leveraging boolean information to construct a searchable and compact index that incorporates an oblivious search protocol.

data structure. The membership information is propagated from the leaf to the root, enabling logarithmic search, while the boolean nature of the index allows for compressibility.

Figure 2 illustrates the new encrypted search method employed by BINDEX, showcasing its construction and search process. During the index construction, each item in the database undergoes a transformation using OWFs, resulting in a set of boolean-form cipher identifiers. These identifiers are positioned at the leaf layer of the data structure. BINDEX is then constructed by recursively combining these identifiers in a Merkle tree fashion, where each inner node represents the *union* of its child identifiers (§IV-A). This process ensures that the membership information of items is propagated up to the root of the tree. During encrypted searches, BINDEX employs a membership testing approach in each node to determine the appropriate tree path to traverse. This enables binary encrypted search by testing the membership of identifiers, mimicking the behavior of BSTs without the need for numerical comparisons on plaintext values.

A key performance challenges faced by BIRDB is its space efficiency, primarily attributed to its unique leaf-to-root membership propagation mechanism. This mechanism, which enables logarithmic-complexity encrypted searches as mentioned earlier, amplifies the space occupancy of BINDEX. This is due to the fact that inner nodes in BINDEX are required to store the membership information of all their child nodes. As a result, the size of each inner node grows exponentially, resulting in a significant increase in the overall index size and adversely impacting the performance of BINDEX.

To address the space efficiency challenge of BINDEX while maintaining its searchability, we draw inspiration from recent advancements in the field of linear approximation models in the database community [39], [40], [44]. Our key idea is to use *offsets* to represent each giant BINDEX node in boolean form. These offsets are then compressed by approximating the entries in the offset arrays using succinct linear models. This allows us to achieve compressed index nodes with constant space occupancy, as only a few slopes and intercepts in the models need to be stored (§IV-B). By employing this compression technique in combination with the logarithmic search complexity of BINDEX, we can significantly reduce the performance penalty caused by the expanded index and enable BIRDB's queries to run at high performance without compromising confidentiality.

Moreover, the boolean nature of BINDEX is compatible with and can be seamlessly integrated into two-phase PIR

protocols. In our implementation, BINDEX is constructed and deployed on two servers: a master server and a replica server. The replica performs database preprocessing to generate auxiliary hints, which are then used by the master server during the online phase to provide efficient responses to user queries with sublinear computation and communication. This integration is seamless because the preprocessing step also relies on boolean operations (§IV-C). The integration between BINDEX and the PIR protocol is mutually beneficial: BINDEX accelerates the PIR process for encrypted searches, while the PIR protocol preserves the metadata confidentiality during searches.

We implemented BIRDB on OO-PIR's codebase [23]. BIRDB's modular architecture allows for independent deployment of index and storage optimizations, providing flexibility and transparency. To evaluate its performance, we conducted a comparative analysis of BIRDB against an uncompressed version (UC-BIRDB), HElibDB [22], and OO-PIR [47], which are two efficient metadata-hidng EDBs, using the same NoSQL workload [77]. Evaluation shows that:

• BIRDB is efficient: BIRDB achieved 5.9x lower latency and 45x higher throughput compared to HElibDB. BIRDB achieved 40.7% lower latency and 3.2x higher throughput compared to OO-PIR. The compressed BINDEX occupied approximately 1037x less space than UC-BIRDB, resulting in further reduced latency by 1.15x. These performance gains can be attributed to the low complexity and compactness (§VI-A).

• BIRDB is scalable: BIRDB's query latency scales logarithmically with an increasing size of the database (§VI-B).

• BIRDB is lossless: BINDEX successfully performed encrypted searches without any errors based on both empirical evaluations and theoretical analysis presented in §IV-B.

Our main contribution is BINDEX, the first efficient and compact index designed for encrypted search. BIRDB makes the first effective attempt to utilize boolean information generated by one-way functions to enable the construction of a searchable and compressible index. Furthermore, it seamlessly integrates with a highly efficient two-phase PIR protocol that also preprocesses the database using boolean information, all within a unified system. BIRDB demonstrates considerable potential for deployment in various safety-critical applications, such as privacy-preserving video streaming [57], [75], password manager [30], recommendation [35], [61], and anonymous messaging [37], [60].

II. BACKGROUND

In this section, we aim to highlight the importance of meeting three key properties - *searchability*, *compactness*, and *metadata protection* - in encrypted databases (compared in Table I). Understanding the significance of these properties will provide motivation for our design of BIRDB.

A. Toward efficient server-side search

In response to growing security concerns about data leakage, enterprises like Keybase [5], PreVeil [6], SpiderOak [2], Sync [3], and Tresorit [4] have emerged, offering encrypted storage services to ensure user data confidentiality. These companies have expressed a need for server-side search functionality on encrypted data, in a way similar to conventional plaintext databases like MySQL [50]. However, none of these companies currently support server-side search, as their development efforts have been hindered by performance or the disclosure of search access patterns.

System	Data Confidentiality	Metadata Security	Sublinear Complexity	Compact Index	General Purpose
MySQL [50]	✗	✗	✓	✓	✓
CryptDB [58]	✓	✗	✓	✗	✓
HElibDB [22]	✓	✓	✗	✓	✓
Dory [24]	✓	✓	✗	✗	✗
OO-PIR [47]	✓	✓	✗	✗	✓
Titanium [47]	✓	✓	✗	✗	✓
BIRDB	✓	✓	✓	✓	✓

TABLE I: Characterization of representative EDBs.

One straightforward yet inefficient way to enable search on encrypted data is to download the entire database. However, this approach is impractical due to the significant bandwidth consumption and user-side storage resources required. Thus although it provides the highest level of security, no existing industrial solution utilizes this approach.

To enable efficient server-side search on encrypted data, extensive academic research has explored the use of specific cryptographic tools, such as property-preserving encryption, as seen in the case of CryptDB [58]. The core concept behind this approach is to preserve the determinism and maintain the order of encrypted data, thereby allowing for the direct construction of an index (typically a BST [20]) on top of the encrypted data with logarithmic search complexity.

The need for compacting index. Despite extensive efforts to achieve efficient search performance, evaluations have shown that the actual performance often falls short of theoretical (i.e., logarithmic) expectations. The primary reason for this discrepancy is that the performance bottleneck does not lie in the binary search. Instead, it is the I/O operations to secondary storage that significantly degrade the overall performance. This is particularly true when dealing with substantial data volumes that cannot be indexed and fit within the limited main memory, compounded by the fact that encryption algorithms typically expand ciphertext size linearly [10], [66].

Therefore, in the context of EDB, achieving high performance necessitates not only sublinear (logarithmic) search complexity but also effective compression techniques. By ensuring sublinear search complexity, the search operations can be executed efficiently, while compression helps mitigate the impact of I/O operations and reduces the storage requirements.

B. Toward efficient metadata protection

Private Information Retrieval (PIR). The initial concept of PIR was introduced by Chor et al. [51], allowing users to retrieve specific items from an untrusted server-managed database without revealing the accessed object. PIR protocols aim to minimize communication costs, achieving sublinear complexity relative to the database size. Modern PIR deployment models include multi-server PIR [26], [42], [45], [46], where the database is replicated across non-colluding servers for enhanced fault tolerance. Alternatively, single-server PIR protocols [48], [49], [54] are computationally more demanding but avoid non-collusion assumptions through cryptographic hardness assumptions. These advancements provide

a range of options for secure and private information retrieval, considering efficiency, security assumptions, and deployment requirements.

Oblivious RAM (ORAM) is a traditional PIR technique known for concealing access patterns during interactions with untrusted servers [55], [65], however, it is not suitable for our setting due to its impractical performance penalty. ORAM protocols achieve access pattern obfuscation through techniques like oblivious data shuffling and path encryption. Researchers have also proposed various ORAM variants like Titanium [21] and Dory [25] to improve performance and scalability while preserving data privacy [17], [32], [59], [63], [64]. However, ORAM-based techniques still suffer from the drawback of processing the entire database with linear computation for every query, resulting in significant performance penalties.

Two-phase PIR on the boolean information. Many research efforts have been put on improving the performance of ORAM by introducing preprocessing techniques [14], [23], [62], [76]. Preprocessing involves shifting the inevitable PIR computation to a query-independent offline phase and generating auxiliary data or hints along the way. This approach aims to make the computational cost of online query processing, with the assistance of these hints, sublinear to the database size.

Two recent PIR protocols, namely the Corrigan-Gibbs and Kogan scheme (CK) [23], and the Shi, Aqeel, Chandrasekaran, and Maggs scheme (SACM) [62], operate on the boolean information to enable a two-phase design. Notably, the CK protocol [23] presents a two-server PIR construction that stores $\mathcal{O}(\sqrt{n})$ bits at the user and achieves $\mathcal{O}(\sqrt{n})$ server computation for online queries. The fundamental idea behind CK is the reflexivity of XOR operations. The CK protocol divides the database into random chunks, and the user preserves a preprocessed hint $h = p \oplus s$, where p represents the desired value to be queried and s is obtained by querying a random chunk of the database. Through this approach, the user can ultimately obtain $p = h \oplus s$ using its hint and queried random result, with correctness guaranteed by the reflexivity property (i.e., $p \oplus s \oplus s = p$).

C. Opportunities

Based on the survey results discussed above, we have identified several opportunities to address the challenge of practical yet secure encrypted search:

• Traditional cryptographic approaches are not capable of simultaneously achieving both searchability and compactness. Hence, there is a strong need for an alternative approach that offers a more flexible and practical index design, even if it entails sacrificing some degree of confidentiality. This trade-off is crucial to ensure the compactness of the encrypted search system, making it efficient for real-world applications.

• Boolean information has shown significant advancements in efficient metadata protection and holds great potential for implementation in an encrypted search index.

Collectively, these opportunities form the basis for our design of a new encrypted database.

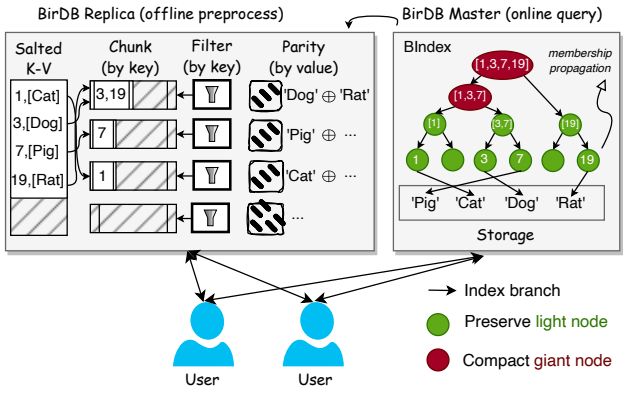

Fig. 3: BIRDB's architecture for searchable, compact, and metadata-hiding encrypted database built on the two-phase PIR.

III. Overview

A. System model

Same as [25], in BIRDB, the database items are divided into partitions, each managed by a different group of EDB servers. Multiple partitions can exist in a query system, and execution across partitions occurs in parallel.

Entities. For a single partition (depicted in Figure 3), we adopt the two-phase PIR model, specifically the CK protocol introduced by Corrigan-Gibbs and Kogan [23]. In our setup, a single partition consists of two servers deployed on the public cloud: an offline server (usually a replica) responsible for database preprocessing to generate runtime hints for PIR, and an online server (master) handling user queries. These servers are *semi-honest*, in the sense that they do not collude but have an interest in learning which objects the user is accessing from the database. Concretely, BIRDB's architecture for a single partition includes the following entities:

• **User:** Multiple users interact with the BIRDB replica to obtain hints and send search queries (e.g., point search) to the BIRDB master for information retrieval. Each user needs to store a *salt*, a randomly generated 128-bit key, used for salting and recovering the original data during offline preprocessing to ensure that plaintexts are hidden.

• **Master:** The BIRDB master processes user queries by searching BINDEX sublinearly and performing sublinear PIR computations. The master also ensures that the l replicas have a consistent view of the database state, and users know which servers to contact. In line with [25], we employ $l = 2$ in our implementation.

• **Replica:** The BIRDB replica conducts database preprocessing to generate runtime *hints*, ensuring metadata-hiding during information retrieval while keeping the computation required on the master sublinear.

Similar to [25], BIRDB does not provide availability if servers refuse to provide services. However, it is important to note that cloud providers are typically incentivized to ensure availability by either bringing the failed server back online or replacing it promptly [25], [38], [72].

Data model. To exemplify BIRDB's design, we employ the NoSQL key-value data model for its simplicity. However, it's important to note that BIRDB is highly extensible and can be applied to other data models transparently, including

structured relational data models and unstructured key-value pairs. This is made possible by the unclustered architecture on which BIRDB is built, as discussed in [50], which decouples BIRDB's encrypted search index from the underlying storage.

Similar to previous works [47], [58], BIRDB replicates the database D across both the offline and online servers. The database consists of two confidential columns, each containing n keys or values (items). Each item has a size of b bits. Before sending values to the replicas, users encrypt them using their salts to ensure confidentiality during preprocessing. The preprocessed results, also known as hints, can later be recovered by the user using the respective salts. The correctness of this process is guaranteed by the reflexivity property of XOR on boolean operations, as we will explain in §IV.

In line with typical PIR systems [25], [47], the value column in database D is treated as an array, where $D[i]$ represents the i-th value in D. The notation $[x, y]$ is used to denote the set of consecutive integers from x to y, inclusive. For simplicity, when x = 1, the notation $[y]$ is used to represent the set $\{1, ..., y\}$.

PIR model (two-phase PIR [23])**.** BIRDB is built on the two-phase PIR which typically comprises four algorithms (`Prep`, `Query`, `Resp`, `Recov`). These algorithms are defined as follows in the setup of BIRDB over a database D of n items:

• `Prep`$(D) \rightarrow h$: this algorithm is executed by BIRDB replicas for each user. It takes the database D as input and outputs a hint h. The hint h consists of three parts: randomly partitioned chunks, filters for chunks, and parities for chunks.

• `Query`$(h, i) \rightarrow q_i$: this algorithm is executed by each user. It takes the hint h and the desired query index i as input and outputs a query q_i.

• `Resp`$(D, q_i) \rightarrow r_i$: this algorithm is executed by the BIRDB master. It takes a query q_i from users as input and outputs a response r_i.

• `Recov`$(h, r_i) \rightarrow d_i$: this algorithm is executed by each user. It takes as input a hint h from a replica and a response r_i from the master, and outputs the desired data object d_i.

These four algorithms work together to enable the metadata-hiding retrieval of specific data objects in BIRDB, ensuring that the access patterns and queried data remain concealed from the untrusted servers.

B. Threat model

In line with recent cloud-based EDBs [21], [25], [47], [58], we assume semi-honest database servers, in the sense that these servers do not collude but are interested in learning the plaintext data and the accessed items. Adversaries, denoted as $\mathcal{A}$, can include database administrators and privileged insiders who can perform passive attacks such as cipher analysis [16] and dictionary attack [18] to extract plaintext information. We rely on the security of one-way functions (OWFs) like SHA-256 [33], assuming that $\mathcal{A}$ is computationally incapable of reversing OWFs or compromising cryptographic encryption keys. These assumptions form the foundation for ensuring the confidentiality of both the data and metadata in BIRDB's encrypted data during encrypted search.

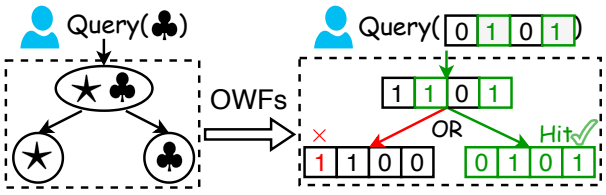

Fig. 4: BIRDB employs boolean information for logarithmic encrypted search by testing membership in index branches and finally comparing every bit at leaf nodes.

We assume that users are always honest and do not engage in collusion with any servers. While access control mechanisms [52] and blacklists [69] can be used to prevent corrupted users, we do not consider them within the scope of this work. Additionally, similar to prior research [25], we assume that the underlying storage engine is maliciously secure. This ensures that servers can retrieve the correct data from the underlying storage by detecting rollback and fork attacks, which have been extensively studied in previous storage works [19], [41], [67].

Out-of-scope attacks. We make the assumption that servers do not engage in active attacks such as modifying, dropping, or replaying queries, and that they adhere to BIRDB's protocol. Mitigations exist to authenticate queries and ensure query integrity on plaintext data under active attacks, but addressing integrity under active attacks on encrypted data remains an open problem that we leave for future work [74], [78]. Furthermore, we assume a fixed size for items, which eliminates size leakage. It is challenging to prevent size leakage efficiently without extensive padding to the largest file size (which is costly) or relying on trusted hardware (which requires trust in a third-party vendor). However, mitigations such as partial padding and delayed accessing, where different file chunks are downloaded at different times, can be employed [53], [68].

BIRDB's guarantees. BIRDB's confidentiality guarantee ensures that all index keys are irreversible, in the sense that if $\mathcal{A}$ is a computationally bounded adversary, $\mathcal{A}$ is infeasible to reverse the encrypted index keys and retrieve the original items in plaintext. These guarantees are facilitated by BIRDB's index, which effectively achieves high-performance encrypted search with logarithmic complexity and compactness. Formally, we have the following theorem that captures BIRDB's security and and is proved in §V:

Theorem 1. *Given that the one-way functions (OWFs) used in index construction is mathematically irreversible, and assuming that the ϵ-approximate linear models and the two-phase PIR scheme of Corrigan-Gibbs and Kogan [23] are secure against passive attacks, the* BIRDB *encrypted database is secure in the presence of a computationally bounded adversary.*

C. System goals and approaches

At a high level, our goal is to achieve searchability, compactness, and metadata protection for encrypted search, while maintaining the confidentiality of both data and metadata. To accomplish this, our key weapon is a new boolean information retrieval (BIR) technique that allows us to achieve all of these objectives simultaneously. Below, we provide a brief overview of the approaches to achieve these desirable features in EDBs.

Goal 1: Logarithmic encrypted search. Enabling search functionality on encrypted data is crucial for efficient retrieval

of specific items without the requirement of decrypting the entire dataset. To address this challenge, our key idea is to replace conventional numerical comparison operations at index branches with secure boolean membership tests. By incorporating secure boolean information into an orchestrated searchable data structure, we can enable encrypted searches with a logarithmic time complexity.

Figure 4 exemplifies our encrypted search method named BINDEX. This approach involves converting database items into boolean information using cryptographic one-way functions (OWFs), such as SHA-256. Drawing inspiration from the Bloom Filter [13], BINDEX employs a membership-preserving data structure to maintain the membership of the OWF-transformed items, which are represented as binary strings (e.g., '⋆' is represented as '1100'). These binary strings serve as leaf nodes positioned at the lowest index layer, while the inner nodes at higher levels are the *unions* of their child nodes. Each inner node performs an *OR* gate computation on the boolean entries of its child nodes to generate its own boolean information. To search for a specific item, BINDEX compares the '1' bits of the item at each index branch, starting from the root, to ensure a match for every '1' bit. At the leaf node, BINDEX compares each entry, including '0' bits, to ensure an exact match with the queried item.

Goal 2: Compact index with constant overhead.

TODO

Goal 3: End-to-end metadata protection. To effectively conceal metadata and eliminate the leakage of search access patterns, we implement BIRDB using the CK protocol introduced by Corrigan-Gibbs and Kogan [23]. The CK protocol is an efficient two-phase PIR protocol that offloads query-independent computations to a preprocessing phase, resulting in sublinear computation for online queries.

Figure 6 demonstrates the integration with two-phase PIR discussed in §III-B. Specifically, the process consists of a preprocessing phase and a query phase. During the preprocessing phase, the BIRDB replica divides the database into random chunks. For each chunk of keys, it computes the parity of the corresponding values and builds filters for each chunk (❶). In the query phase, the user searches for their desired query item by testing its membership in each chunk's filter. Once the target chunk is found, the user deletes the intended query item (❷). The BIRDB master then searches the database using

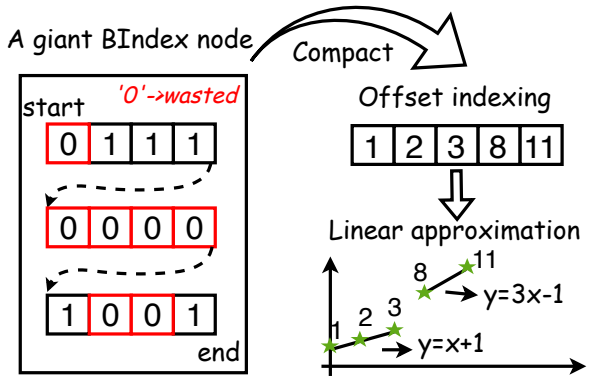

Fig. 5: Based on the boolean information, BIRDB compacts giant index nodes to optimize I/O efficiency and achieve high performance.

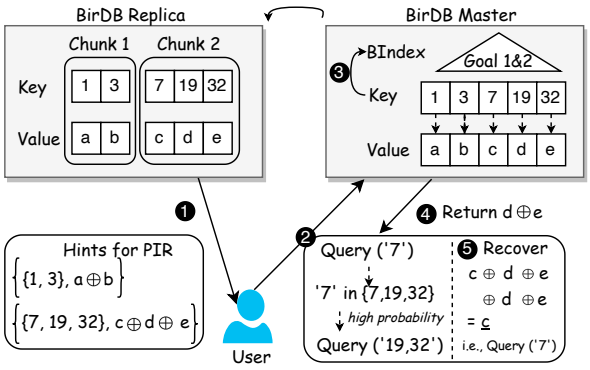

Fig. 6: The third goal of metadata-hiding in BIRDB is achieved through the utilization of two-phase PIR. BINDEX, which accomplishes **Goals 1 and 2**, accelerates the PIR process, while PIR ensures the metadata confidentiality of BINDEX.

BINDEX based on the query items and computes the parity of the searched values (❸ and ❹). Finally, the user recovers the result by XORing the returned parity with the parity of the original chunk (❺). The correctness of this process is ensured by the reflexivity of XOR on boolean operations; BIRDB accelerates the PIR process using BINDEX (§IV-C).

Overall, by utilizing one core concept of boolean information retrieval, we construct a new encrypted search index with logarithmic complexity using OWFs. Furthermore, we compact the index by employing linear approximation techniques. Finally, we ensure the concealment of search metadata by integrating with two-phase PIR based on boolean operations. By doing so, we address all three desired properties: searchability, compactness, and metadata-hiding, in a unified manner.

IV. PROTOCOL DESCRIPTION

A. Desiging an efficient encrypted search index

We first establish the foundation for constructing BIRDB's encrypted search index and subsequently delve into the details of index construction and search methods.

Preliminaries. The underlying searchable data structure in BIRDB is based on the Bloom filter [13], a compact data structure designed for efficient membership testing. The Bloom filter operates by utilizing a single set to store the hash values of all items. Specifically, a Bloom filter B is represented by an m-bit array and employs k independent and uniformly distributed OWFs, i.e., cryptographic hash functions $\{H_k\}$. Each hash function maps an item to an index in the Bloom filter array. To insert an item x into the filter, we compute its hash using $\{H_k\}$ and set all the bits at the corresponding indices in B to 1. For testing whether an item 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). Conversely, if all the bits at the corresponding indices in B are 1, y is considered to have a high probability of being in the set, albeit with a small configurable false positive rate [13].

Challenges and opportunities. As mentioned above, traditional Bloom filters offer valuable information regarding the *membership* of data for tests. In addition to this, we have observed two key characteristics of a Bloom filter that make it promising for encrypted search. Firstly, a Bloom filter is

monolithic in nature, meaning that all items are assigned to a single boolean set. This characteristic allows for efficient item membership tests related to data existence. Secondly, a Bloom filter ensures data *confidentiality* by utilizing one-way functions (OWFs) to transform original plaintexts into hash values of items. The use of OWFs makes the Bloom filter computationally irreversible, effectively hiding the original plaintexts. This security property relies on the assumption of the one-wayness of cryptographic hash functions, which has been extensively studied in prior research [13].

However, despite the valuable membership property and data confidentiality offered by a Bloom filter, the monolithic nature limits its functionality to supporting only membership tests by indicating whether an item (hash) is present in a set. Consequently, Bloom filters alone are not capable of functioning as an index for efficiently searching encrypted data. We observe two significant benefits of using Bloom filters for encrypted search:

• **Weaker but sufficient security guarantee.** In encrypted search, users do not require the encrypted index keys to be decryptable, as they are used solely for searching corresponding values. Therefore, the cryptographic approaches that prior works use are overkilled [22], [47]; the utilization of one-way functions (OWFs) to efficiently transform plaintexts into computationally irreversible ciphertexts is sufficient.

• **Compressability**. The boolean nature of Bloom filters allows for the utilization of boolean information compression techniques. Leveraging these techniques can lead to significant compression gains in the encrypted search index. This is particularly advantageous as traditional encrypted search indexes, such as Order-Preserving Encryption (OPE)-based indexes, are not easily compressible [58], [25]. By reducing the index size through compression, the encrypted search system can benefit from improved performance by avoiding the drawbacks associated with large index sizes and potential performance penalties. The details regarding BIRDB compression are introduced in §IV-B.

Approach. To construct the encrypted search index, we modify the traditional monolithic Bloom filter by decomposing it into a collection of *minifilters*. Each minifilter is responsible for storing the hash (in bits) of a single item. In other words, a mini-filter represents a Bloom filter dedicated to an individual data item, where only one item (hash) is assigned to the set.

The minifilters are recursively organized to form a searchable index known as BINDEX. These minifilters have a fixed array size m, and are arranged in a Merkle-tree structure. Each minifilter is stored at a leaf node of the tree, representing an index key. The index keys, such as `"1100"` for record '⋆' in Figure 4, are sorted based on their plaintext key order. The values of the inner nodes, which include all nodes except the leaf nodes, are determined through an element-wise OR gate computation on the minifilters of their child nodes. Let's consider an inner node with l child nodes, denoted as Cn. For each bit (array entry) in the minifilter of the inner node, its value is computed as follows: $Sub_{in}[i] = Cn_1[i]$ *or ... or* $Cn_l[i]$, for all i in the range $[0, m]$. Note that, in the construction of BINDEX, only the leaf nodes in the minifilter tree store actual values, such as record addresses. The inner nodes serve the primary purpose of indexing, facilitating tree traversal for efficient search operations.

Algorithm 1: Minifilter-based encrypted search

```
1  Function PointSearch(root, key) do
2      if isLeaf(root) then
           ▽ Every corresponding bit should match
3          if key.rft = root.rft        ▷ Minifilter comparison
4          then
5              return root
       ▽ In inner nodes, every '1' bit should match
6      if isInner(root) then
7          if left_minifilter.contain(key) then
8              return PointSearch(root.left, key)
9          if right_minifilter.contain(key) then
10             return PointSearch(root.right, key)
11         return NULL                  ▷ Key is non-existent
12 Function RangeSearch(root, [key1, key2]) do
13     left ← PointSearch(root, key1)
14     right ← PointSearch(root, key2)
15     while left.next != right do
16         value[] ← left.next      ▷ Retrieve sorted leaf nodes
17     return value
```

Design rationale. The searchability of BINDEX is achieved through the membership-preserving property of the minifilters. Specifically, when any entry in a child node's minifilter is set to `"1"`, the OR gate computation guarantees that the corresponding entry in its parent node's minifilter will also contain a `"1"` bit. For example, in Figure 4, the first entry of the minifilter for '⋆' is `"1"` while the first entry of the minifilter for '♣' is `"0"`. As a result, the first entry of their parent node's minifilter is computed as $1\ OR\ 0 = 1$.

The BINDEX design ensures that if an item exists in a minifilter node, it must also exist in its parent node. In other words, the membership information is propagated to higher-level index nodes. This novel observation, utilized by BINDEX, allows for efficient top-down logarithmic tree traversal based on the membership information. Furthermore, the irreversible confidentiality provided by OWFs directly carries over to the construction of minifilters. By testing the membership of query items in the minifilter nodes, BINDEX eliminates the need for plaintext comparisons used in BSTs of traditional insecure databases.

Using BINDEX for encrypted searches. Algorithm 1 outlines the process of an encrypted point query through BINDEX, which involves generating a minifilter for the query item and traversing BINDEX to locate the corresponding value. BIRDB performs minifilter comparisons at each inner node to verify the membership of each `"1"` bit in the query item's minifilter. The traversal continues until reaching the leaf nodes, where the match of every bit confirms the presence of a unique index key.

The point search method can be extended to handle range queries. By transforming the range boundaries into minifilters, BINDEX locates two matched leaf nodes using the same method as point queries. All values between these leaf nodes are then returned, akin to the retrieval process in a B^+-tree.

B. BINDEX *compression*

A core index challenge: size and cost. Despite BINDEX's efficient tree-based index structure, which facilitates logarithmic search complexity, achieving a low search complexity alone is insufficient for achieving high-performance encrypted search. The reasons are two fold. Firstly, as the total number

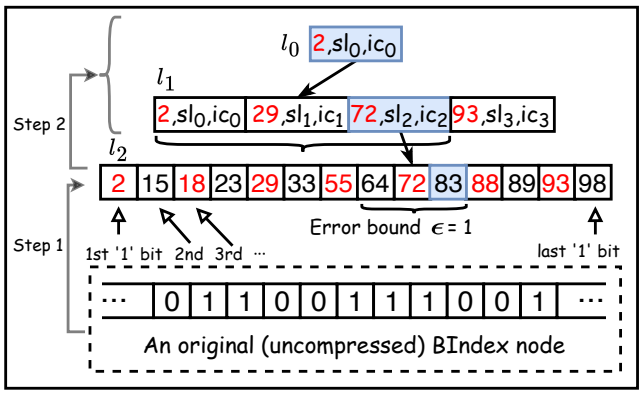

Fig. 7: The construction and a running example that show our compact design for BINDEX. **Step 1** extracts an original BINDEX node into offset representation wherein the ith entry is the position of the ith '1' bit. In **Step 2**, we uses the Piece-wise Linear Approximation model [29] to compact and search the extracted offset array.

of items increases, each BINDEX node experiences exponential growth in membership information storage due to the membership propagation design (§IV-A). Consequently, the index size significantly expands, adversely affecting search efficiency. Secondly, even in a static setting, the false positive issue, attributed to the fundamental nature of Bloom filters, further amplifies the size of each minifilter node.

To illustrate, let's consider a database with 2^{20} static items and the user aims for a low false positive rate of 1×10^{-3}, same as the configuration in [73], each minifilter in BINDEX would need to be approximately 1.79MB in size according to the Bloom filter theory [13]. Consequently, the entire BINDEX would require a substantial amount of storage space, around 3.57TB, which is impractical. The significant increase in space occupancy compared to a plaintext database index results in a notable performance decline due to expensive I/O operations involving secondary storage.

Facing this challenge, our contention is that relying solely on achieving theoretical logarithmic search complexity is insufficient for achieving high-performance encrypted search, even though it represents a significant improvement over linear search methods [22], [24]. Instead, an encrypted search index should adopt a compact design to ensure that the overall query performance remains manageable and cost-effective.

The compact design.

TODO

Running example. Figure 7 provides a running example of searching a compressed BINDEX node using Algorithm **??**. Each accessed node in the search path is highlighted in blue. We assume the compressed BINDEX has an error bound $\epsilon = 1$ and the key being searched is $k = 83$.

The search begins from the root segment $s' = l_0[0][0]$. The next position, $\lfloor fs'(k) \rfloor = \lfloor k * sl_0 + ic_0 \rfloor$, is computed to be 1 for the next level. The search then proceeds to locate k within the range $[1-\epsilon, 1+\epsilon]$ in l_1, considering the keys [2, 29, 72]. It is determined that the next segment for the search is $s'' = l_1[2]$ because $k > 72$. Finally, the position for the next (leaf) level, l_2, which represents the extracted offset array, is computed by evaluating $\lfloor fs''(k) \rfloor = 8$. Consequently, a binary search is performed to search for k within the range $[8-\epsilon, 8+\epsilon]$ in $l2$. Ultimately, it is found that k is located at position 9 since $A[9] = 83$.

In the compacted BINDEX, Algorithm **??** functions as the internal search API of `minifilter.contain(`k`)` in Algorithm 1. Combined, they constitute the complete encrypted search algorithm (without metadata protection) of BIRDB.

C. Seamless integrating BINDEX with two-phase PIR

Our metadata-hiding solution builds on the two-phase PIR scheme of Corrigan-Gibbs and Kogan [23]. We start by describing the CK protocol and then we decribe our approach to seamlessly integrate BINDEX with it.

Overview of the CK protocol. As described in §II-B, the CK protocol comprises two phases: an offline preprocessing phase conducted on BIRDB's replica, and an online query processing phase involving BIRDB's master. The offline preprocessing phase is independent of the specific item that the user wants to query and involves the `Prep` step. This phase is performed on BIRDB's replica and includes the necessary preparations for the PIR search process. In the online query processing phase, which depends on the specific item being queried, the interactions occur between BIRDB's master (who has no knowledge of the preprocessed hints generated by the replica) and the user. This phase consists of the `Query`, `Resp`, and `Recov` steps, where the user submits the query, the master provides the corresponding response, and the necessary recovery actions are taken.

- `Prep(`D`)` $\rightarrow h$. The replica divides all index keys of the database items into $\sqrt{n}\log n$ independent random sets, denoted as S_1, ..., $S_{\sqrt{n}\log n}$, where each set contains $\sqrt{n}$ keys sampled uniformly at random. For each set S_i, the replica computes its parity, P_i, where $p_i = \oplus_{t \in S_i} D[t]$ for $i \in [1, \sqrt{n}\log n]$. Each parity is uniformly padded to b bits. In short, a parity p_i for set s_i is the XOR of all items in D referenced by the keys in S_i. The replica sends the hint $h = \{(S_1, P_1), ..., (S_{\sqrt{n}\log n}, P_{\sqrt{n}\log n})\}$ to the user.

- `Query(`h, i`)` $\rightarrow q_i$. The user generates a query for retrieving an item with index key i (i.e., D[i]). The user selects a set S from the hint h such that $i \in S$ (the corresponding parity is P_S). The user then removes i from set S with a high probability (i.e., $1 - \frac{\sqrt{n}-1}{n}$); otherwise, it removes a randomly chosen key from $S \backslash \{i\}$. The query q_i is transformed to the resulting set S^*, which has a size of $\sqrt{n} - 1$. The user sends q_i to the master.

- `Resp(`D, q_i`)` $\rightarrow r_i$. The master computes the PIR, $r_i = \oplus_{t \in S^*} D[t]$, where S^* is the query set received from the user. The master sends r_i back to the user.

- `Recov(`h, r_i`)` $\rightarrow d_i$. Using the parity P_S obtained during the `Query` step, the user recovers the real desired query item $D[i]$ by computing $d_i = P_S \oplus r_i$.

To ensure security against a semi-honest online server, the CK protocol incorporates a cheap *refresh* operation after each query. This operation involves the user generating a new set with $\sqrt{n}-1$ new random keys and obtaining the corresponding parity P_{new} from the replica. The user then adds the previously queried key k to the new set and updates the parity by XORing the key's parity with P_{new}. This refresh operation prevents the

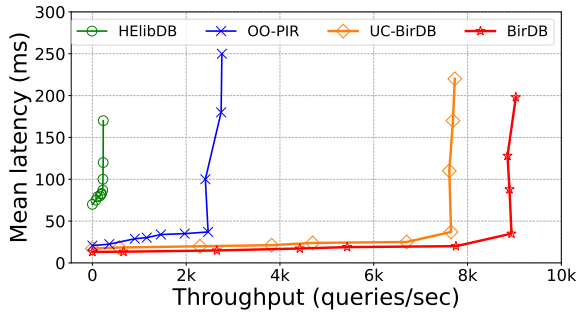

Fig. 8: Throughput and latency comparison among different metadata-hiding baselines, each equipped with 2^{20} items.

server from deducing search access patterns by comparing sets in multiple queries.

Accelerating PIR with BINDEX.

TODO

V. PERFORMANCE AND SECURITY ANALYSIS

A. Proof Sketch of Security

TODO

B. High performance

TODO

VI. EVALUATION

Testbed. The experiments were conducted on a cluster of lab machines, each featuring a 2.60GHz Intel E5-2690 V3 CPU, 64GB memory, 40Gbps NIC, and 24 cores. All nodes, including users, master, and the replica, were executed within Docker containers. The average ping latency between the nodes was set at 0.17ms with the aid of Linux traffic control [12].

Baseline. We utilized WiredTiger [36], a prominent NoSQL key-value storage engine, to implement all of our baseline systems. WiredTiger is widely employed in industrial MongoDB deployments, making it an appropriate default choice. Our comparative analysis involved three state-of-the-art baseline approaches: HElibDB [22], an imperative oblivious database developed by IBM, which employs Fermat's Little Theorem for comparison on homomorphic encrypted data. However, each query in HElibDB necessitates processing the entire database for metadata protection. We also incorporated the CK [23] two-server OO-PIR protocol, known for its exceptional performance. The CK protocol moves query-independent computation to an offline preprocessing phase. Lastly, we assessed a version of BIRDB without compression, denoted as UC-BIRDB in subsequent evaluations. Although UC-BIRDB maintains logarithmic search complexity through our implementation of a boolean information-based index structure (§IV-A), it exhibits a significantly larger size due to lack of compression.

Workload and default setting. We adopted the same NoSQL key-value workload as CK OO-PIR. This involved generating a range of keys and values with fixed sizes, specifically ranging from 2^{10} to 2^{20}. The key size was set at 8 bytes, while the value size was set at 64 bytes. Unless specified for the scalability study, our evaluations were conducted on a default dataset of 2^{20} key-value pairs. We used point searches for experiments by default. Each experiment was executed for a duration of 60 seconds, and the results were collected during the middle 30 seconds (i.e., 15s to 45s) to mitigate any potential disturbances caused by system start-up or cool-down. Our extensive evaluation focused on addressing the following research questions:

- How efficient is BIRDB compared to baselines?
- What is the communication cost of BIRDB?
- Can BIRDB scale efficiently?
- How sensitive is BIRDB to compression error bound?

A. End-to-end Performance

TODO

B. Scalability

TODO

C. Sensitivity

TODO

VII. DISCUSSION

TODO

VIII. CONCLUSION

TODO

REFERENCES

[1] "Data breaches - updated list," https://tech.co/news/data-breaches-updated-list, accessed on June 2, 2023.

[2] "Spideroak," 2007, website: https://spideroak.com/.

[3] "Sync," 2011, website: https://www.sync.com/.

[4] "Tresorit," 2011, website: https://tresorit.com/.

[5] "Keybase," 2014, website: https://keybase.io/.

[6] "Preveil," 2015, website: https://www.preveil.com/.

[7] (2023, May 20) Chatgpt confirms data breach. [Online]. Available: https://securityintelligence.com/articles/chatgpt-confirms-data-breach/

[8] (2023, May 15) Tiktok data breach timeline. [Online]. Available: https://firewalltimes.com/tiktok-data-breach-timeline/

[9] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, 2004, pp. 563–574.

[10] R. R. Al-Dahhan, Q. Shi, G. M. Lee, and K. Kifayat, "Survey on revocation in ciphertext-policy attribute-based encryption," *Sensors*, vol. 19, no. 7, p. 1695, 2019.

[11] M. B. Aliyu, "Efficiency of boolean search strings for information retrieval," *American Journal of Engineering Research*, vol. 6, no. 11, pp. 216–222, 2017.

[12] W. Almesberger *et al.*, "Linux network traffic control—implementation overview," 1999.

[13] F. Angius, M. Gerla, and G. Pau, "Bloogo: Bloom filter based gossip algorithm for wireless ndn," in *Proceedings of the 1st ACM workshop on Emerging Name-Oriented Mobile Networking Design-Architecture, Algorithms, and Applications*, 2012, pp. 25–30.

[14] A. Beimel, Y. Ishai, and T. Malkin, "Reducing the servers computation in private information retrieval: Pir with preprocessing," in *Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings 20*. Springer, 2000, pp. 55–73.

[15] M. Bellare, M. Fischlin, A. O'Neill, and T. Ristenpart, "Deterministic encryption: Definitional equivalences and constructions without random oracles," in *Advances in Cryptology–CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings 28*. Springer, 2008, pp. 360–378.

[16] J. Black, P. Rogaway, and T. Shrimpton, "Black-box analysis of the block-cipher-based hash-function constructions from pgv," in *Crypto*, vol. 2442. Springer, 2002, pp. 320–335.

[17] D. Boneh, D. Mazieres, and R. A. Popa, "Remote oblivious storage: Making oblivious ram practical," 2011.

[18] L. Bošnjak, J. Sreš, and B. Brumen, "Brute-force and dictionary attack on hashed real-world passwords," in *2018 41st international convention on information and communication technology, electronics and microelectronics (mipro)*. IEEE, 2018, pp. 1161–1166.

[19] M. Brandenburger, C. Cachin, M. Lorenz, and R. 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)*. IEEE, 2017, pp. 157–168.

[20] N. G. Bronson, J. Casper, H. Chafi, and K. Olukotun, "A practical concurrent binary search tree," *ACM Sigplan Notices*, vol. 45, no. 5, pp. 257–268, 2010.

[21] W. Chen, T. Hoang, J. Guajardo, and A. A. Yavuz, "Titanium: A metadata-hiding file-sharing system with malicious security," *Cryptology ePrint Archive*, 2022.

[22] H. E. L. Contributors, "Helib: Homomorphic encryption library," https://github.com/homenc/HElib, 2021.

[23] H. Corrigan-Gibbs and D. Kogan, "Private information retrieval with sublinear online time," in *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*. Springer, 2020, pp. 44–75.

[24] E. Dauterman, E. Feng, E. Luo, R. A. Popa, and I. Stoica, "Dory: An encrypted search system with distributed trust," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 1101–1119.

[25] ——, "Dory: An encrypted search system with distributed trust," in *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, 2020, pp. 1101–1119.

[26] D. Demmler, A. Herzberg, and T. Schneider, "Raid-pir: practical multi-server pir," in *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*, 2014, pp. 45–56.

[27] J. Doerner and A. Shelat, "Scaling oram for secure computation," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 523–535.

[28] I. Dropbox, "Dropbox," *http://www. dropbox. com*, 2014.

[29] P. Ferragina and G. Vinciguerra, "The pgm-index: a fully-dynamic compressed learned index with provable worst-case bounds," *Proceedings of the VLDB Endowment*, vol. 13, no. 8, pp. 1162–1175, 2020.

[30] P. Gasti and K. B. Rasmussen, "On the security of password manager database formats," in *Computer Security–ESORICS 2012: 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings 17*. Springer, 2012, pp. 770–787.

[31] C. Gentry, K. A. Goldman, S. Halevi, C. Julta, M. Raykova, and D. Wichs, "Optimizing oram and using it efficiently for secure computation," in *Privacy Enhancing Technologies: 13th International Symposium, PETS 2013, Bloomington, IN, USA, July 10-12, 2013. Proceedings 13*. Springer, 2013, pp. 1–18.

[32] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *Journal of the ACM (JACM)*, vol. 43, no. 3, pp. 431–473, 1996.

[33] S. Gueron, S. Johnson, and J. Walker, "Sha-512/256," in *2011 Eighth International Conference on Information Technology: New Generations*. IEEE, 2011, pp. 354–358.

[34] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan, "One server for the price of two: Simple and fast single-server private information retrieval," *Cryptology ePrint Archive*, 2022.

[35] Y. Himeur, A. Sayed, A. Alsalemi, F. Bensaali, A. Amira, I. Varlamis, M. Eirinaki, C. Sardianos, and G. Dimitrakopoulos, "Blockchain-based recommender systems: Applications, challenges and future opportunities," *Computer Science Review*, vol. 43, p. 100439, 2022.

[36] S. Idreos and M. Callaghan, "Key-value storage engines," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2667–2672.

[37] P. Jiang, Q. Wang, J. Cheng, C. Wang, L. Xu, X. Wang, Y. Wu, X. Li, and K. Ren, "Boomerang:{Metadata-Private} messaging under hardware trust," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 877–899.

[38] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Financial Cryptography and Data Security: FC 2010 Workshops, RLCPS, WECSR, and WLC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers 14*. Springer, 2010, pp. 136–149.

[39] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3183713.3196909

[40] T. Kraska, A. Beutel, E. H. Chi, J. Dean, N. Polyzotis, and L. Zhu, "Consistency guarantees for parallel incremental data processing," in *Proceedings of the 2019 International Conference on Management of Data*. ACM, 2019, pp. 1271–1288. [Online]. Available: https://dl.acm.org/doi/10.1145/3299869.3319882

[41] T. K. Kuppusamy, V. Diaz, and J. Cappos, "Mercury:{Bandwidth-Effective} prevention of rollback attacks against community repositories," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 673–688.

[42] K. Kurosawa, "How to correct errors in multi-server pir," in *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part II*. Springer, 2019, pp. 564–574.

[43] A. H. Lashkari, F. Mahdavi, and V. Ghomi, "A boolean model in information retrieval for search engines," in *2009 International Conference on Information Management and Engineering*. IEEE, 2009, pp. 385–389.

[44] C. Li, W. Ma, and L. Qin, "Efficient and accurate approximate query processing on data warehouses with learned indexes," in *Proceedings of the 2019 International Conference on Management of Data*. ACM, 2019, pp. 1717–1734. [Online]. Available: https://dl.acm.org/doi/10.1145/3299869.3319886

[45] S. Li and M. Gastpar, "Converse for multi-server single-message pir with side information," in *2020 54th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2020, pp. 1–6.

[46] H.-Y. Lin, S. Kumar, E. Rosnes, A. G. i Amat, and E. Yaakobi, "Multi-server weakly-private information retrieval," *IEEE Transactions on Information Theory*, vol. 68, no. 2, pp. 1197–1219, 2021.

[47] Y. Ma, K. Zhong, T. Rabin, and S. Angel, "Incremental {Offline/Online}{PIR}," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1741–1758.

[48] S. J. Menon and D. J. Wu, "Spiral: Fast, high-rate single-server pir via fhe composition," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 930–947.

[49] M. H. Mughees, H. Chen, and L. Ren, "Onionpir: Response efficient single-server pir," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2292–2306.

[50] A. MySQL, "Mysql," 2001.

[51] F. Olumofin and I. Goldberg, "Revisiting the computational practicality of private information retrieval," in *International Conference on Financial Cryptography and Data Security*. Springer, 2011, pp. 158–172.

[52] S.-C. Park and M.-S. Lee, "A survey of access control models in database management systems," *Journal of Computing Science and Engineering*, vol. 9, no. 2, pp. 57–78, 2015.

[53] S. Patel, G. Persiano, K. Yeo, and M. Yung, "Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via

hashing," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 79–93.

[54] G. Persiano and K. Yeo, "Limits of preprocessing for single-server pir," in *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2022, pp. 2522–2548.

[55] B. Pinkas and T. Reinman, "Oblivious ram revisited," in *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*. Springer, 2010, pp. 502–519.

[56] B. Pinkas and T. Schneider, "Oblivious ram: Improved lower bounds and optimal constructions," in *Proceedings of the 31st Annual International Cryptology Conference (CRYPTO)*, 2009, pp. 502–519. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-03356-8_28

[57] R. Poddar, G. Ananthanarayanan, S. Setty, S. Volos, and R. A. Popa, "Visor: Privacy-preserving video analytics as a cloud service," in *Proceedings of the 29th USENIX Conference on Security Symposium*, 2020, pp. 1039–1056.

[58] R. A. Popa, C. M. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: Protecting confidentiality with encrypted query processing," in *Proceedings of the twenty-third ACM symposium on operating systems principles*, 2011, pp. 85–100.

[59] L. Ren, C. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. Van Dijk, and S. Devadas, "Constants count: Practical improvements to oblivious {RAM}," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 415–430.

[60] T. Shen, J. Jiang, Y. Jiang, X. Chen, J. Qi, S. Zhao, F. Zhang, X. Luo, and H. Cui, "Daenet: making strong anonymity scale in a fully decentralized network," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2286–2303, 2021.

[61] T. Shen, J. Qi, J. Jiang, X. Wang, S. Wen, X. Chen, S. Zhao, S. Wang, L. Chen, X. Luo *et al.*, "{SOTER}: Guarding black-box inference for general neural networks at the edge," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 723–738.

[62] E. Shi, W. Aqeel, B. Chandrasekaran, and B. Maggs, "Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time," in *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part IV 41*. Springer, 2021, pp. 641–669.

[63] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li, "Oblivious ram with o ((logn) 3) worst-case cost." in *Asiacrypt*, vol. 7073. Springer, 2011, pp. 197–214.

[64] E. Stefanov, M. v. Dijk, E. Shi, T.-H. H. Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path oram: an extremely simple oblivious ram protocol," *Journal of the ACM (JACM)*, vol. 65, no. 4, pp. 1–26, 2018.

[65] E. Stefanov, E. Shi, and D. Song, "Towards practical oblivious ram," *arXiv preprint arXiv:1106.3652*, 2011.

[66] D. Stehlé and R. Steinfeld, "Faster fully homomorphic encryption," in *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer, 2010, pp. 377–394.

[67] E. Tsyrklevich and B. Yee, "Dynamic detection and prevention of race conditions in file accesses," in *12th USENIX Security Symposium (USENIX Security 03)*, 2003.

[68] J. Wang, S.-F. Sun, T. Li, S. Qi, and X. Chen, "Practical volume-hiding encrypted multi-maps with optimal overhead and beyond," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2825–2839.

[69] Z. Wang, R. Hu, T. Yu, and C. Chen, "Design and implementation of a database client blacklist mechanism," in *2019 International Conference on Computing, Communications and Intelligence Systems (ICCCIS)*. IEEE, 2019, pp. 71–74.

[70] E. W. Weisstein, "Fermat's little theorem," *https://mathworld. wolfram. com/*, 2004.

[71] K. Wilson and K. Wilson, "Onedrive," *Everyday Computing with Windows 8.1*, pp. 71–74, 2015.

[72] J. Wu, L. Ping, X. Ge, Y. Wang, and J. Fu, "Cloud storage as the infrastructure of cloud computing," in *2010 International conference on intelligent computing and cognitive informatics*. IEEE, 2010, pp. 380–383.

[73] P. Wu, J. Ning, J. Shen, H. Wang, and E.-C. Chang, "Hybrid trust multi-party computation with trusted execution environment," in *The Network and Distributed System Security (NDSS) Symposium*, 2022.

[74] Y. Xia, X. Yu, M. Butrovich, A. Pavlo, and S. 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*, 2022, pp. 12–17.

[75] H. Xu, Z. Cai, D. Takabi, and W. Li, "Audio-visual autoencoding for privacy-preserving video streaming," *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 1749–1761, 2021.

[76] K. Yeo, "Lower bounds for (batch) pir with private preprocessing," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2023, pp. 518–550.

[77] Y. Zhong, H. Li, Y. J. Wu, I. Zarkadas, J. Tao, E. Mesterhazy, M. Makris, J. Yang, A. Tai, R. Stutsman *et al.*, "{XRP}:{In-Kernel} storage functions with {eBPF}," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 375–393.

[78] W. Zhou, Y. Cai, Y. Peng, S. Wang, K. Ma, and F. Li, "Veridb: An sgx-based verifiable database," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2182–2194.