A Survey of Advanced Encryption for Database Security: Primitives, Schemes, and Attacks

Buvana Ganesh[⋆] and Paolo Palmieri

School of Computer Science & IT,
University College Cork, Ireland
b.ganesh@cs.ucc.ie, p.palmieri@cs.ucc.ie

Abstract. The use of traditional encryption techniques in Database Management Systems is limited, as encrypting data within the database can prevent basic functionalities such as ordering and searching. Advanced encryption techniques and trusted hardware, however, can enable standard functionalities to be achieved on encrypted databases, and a number of such schemes have been proposed in the recent literature. In this survey, different approaches towards database security through software/hardware components are explored and compared based on performance and security, and relevant attacks are discussed.

1 Introduction

Ross Anderson famously stated that "you cannot construct a database with scale, functionality and security because if you design a large system for ease of access it becomes insecure, while if you make it watertight it becomes impossible to use" [6]. While compromises between security and functionality are common in many technological domains, getting the right balance is critical in databases, which are a primary way to store complex information on digital systems. Two factors make research into database security more relevant than ever: first, the ever increasing amount of information that is being generated, collected and stored, most of which sensitive, partly due to advances in the Internet of Things and the diffusion of personal devices such as smartphones and wearables; secondly, the trend towards outsourcing information storage to the cloud, following the Database-as-a-Service (DBaaS) paradigm, which introduces additional privacy risks [4].

Traditional cryptographic techniques cannot be easily applied to databases, due to their inherent nature as a way to structure information. While cryptography can provide an "external" layer of security, where the information is encrypted while at rest (for example through the use of file system or full disk encryption), this does not protect against any attack that targets a "live" database, for example via malicious queries. Trivially encrypting data within a database, instead, compromises the functionality of the database, as it prevents basic operations such as ordering, (partial) equality testing, search and many other. An alternative strategy to protect information is *differential privacy*, which adds noise to data but yields almost accurate results using statistical mechanisms, based on a privacy budget. Although corporations like Apple and Google favor

[⋆] Buvana Ganesh is supported by a PhD scholarship funded by the Science Foundation Ireland Centre for Research Training in Artificial Intelligence under Grant No. 18/CRT/6223.

this for ease of usage, it is not suitable for accurate search and retrieval, and is difficult to implement in dynamic (constantly changing) databases. On the other hand, research on advanced cryptographic mechanisms such as homomorphic or searchable encryption, combined with novel hardware security mechanisms, is increasingly pushing the boundaries of the security *vs.* functionality compromise in databases by allowing, for example, certain operations to be performed on encrypted data. Anderson's intuition remains true, and even the most advanced security schemes cannot entirely prevent leakage of information, which may allow an attacker to approximately reconstruct a database in less than linear time [27]. However, the adoption of these advanced mechanisms can undoubtedly enhance the security of database systems, while maintaining performance comparable to that of an unencrypted database.

In this paper, we discuss the novel cryptographic and security primitives that can be applied to databases (Section 3), and we survey the schemes in the literature that are employing these primitives for database security (Section 4). To the best of our knowledge, no previous survey covers secure databases with encryption and hardware. While a direct comparison of the schemes is challenging, due to the different levels of functionalities they provide, we categorise the schemes according to the security they claim to offer, and juxtapose their respective performances according to standard benchmark tests (Section 5). Finally, we discuss the main potential attacks in relation to the schemes, in order to gain valuable insight for the development of future schemes.

2 Terminology and Definitions

A database (DB) is an organised collection of data of any type, used for the storage, retrieval and management of information, which are performed through a database management system (DBMS). DBs are differentiated based on their structure and properties. Relational databases (RDB) are structured by predefined categories with a *Schema* of *Columns* and *Tables*. NoSQL databases are not as well-defined, and the data is malleable and eventually consistent, thus improving scalability and flexibility. Types include *Key-value*, *Graph*, *Wide Column* or *Document* oriented stores. NewSQL forms a class of RDBMS that seek to provide the same scalability of NoSQL and are distributed to enable faster distributed analytics. In terms of warehousing, Online Analytic Processing (OLAP) is utilized for large scale read-only multidimensional databases to create online comparative summaries of data. Online Transaction Processing (OLTP) helps in dynamic transactions and updates on smaller sizes of data with read and write access.

In the following, we refer to databases that are protected either cryptographically or through trusted hardware as encrypted databases (EDB), and their respective encrypted database management systems as EDBMS.

The aim of an EDB is to provide secure data outsourcing and sharing. When outsourced either to a cloud or third party, who performs computations on the database, the data should possess the necessary properties that enable such computations to be performed without decryption. These normally include search, retrieval, transactions and operations. The EDBMS should also enable access control and policies. A basic set of properties including full range of Queries, Storage, Memory management, Dynamic Updates, Access pattern hiding, etc. are to be considered for the construction.

In EDBMS, generally, the client acts as user interface for identification, authentication, key generation and sometimes query processing. Proxies, if used, mainly help in the modification of queries received by the server dedicated to database storage-retrieval. Additional servers with various purposes such as key distribution, computation, etc. may be present.

2.1 Threats and Adversaries

Attacks on unencrypted databases are commonplace in the news causing significant losses every year. Common threats include privilege abuse, image leaks from virtual machines, SQL injections, and disk theft [28]. The main way to secure unencrypted DBs is through the use of policies and access control. *Query control* [56], for instance, uses policies to ensure the query is sufficiently targeted. However, only cryptographic security can make data provably secure, even in the case of attacks on the DBMS components. Currently, most industry DB solutions only rely on *Transparent Data Encryption* for securing data at rest, and *Transport Layer Security* for data in transit. This implies that any computation can be performed only after decryption. EDBs aim to provide increased security, and prevent log, memory and access pattern based attacks.

The security literature categorises adversaries based on their abilities. In the context of databases, *honest but curious* adversaries are administrators or service providers (SPs) with access to a part (or sometimes full) history of queries performed. Most schemes assume this and it is often referred to as the “Snapshot attack” [28]. Persistent passive attackers compromise the DBMS server and passively observe all its operations. *Malicious* administrators, users or servers are capable of modifying the contents of a DB to manipulate the output. They can carry out *chosen document*, *query* or *keyword* attacks based on prior knowledge, as discussed in Section 5.2. Databases can be attacked by internal or external entities with various levels of prior knowledge from *fully known document*, *partially known document*, *known document subset* to *distributional knowledge*. Based on the level of prior knowledge, it is possible to recover queries, plaintext or partial plaintext (see Section 5.2).

2.2 Provable Security

Different types of proofs are available to establish security, the most common being *game-based proofs*. Game-based proofs cover indistinguishability attacks such as *Known Plaintext Attack* (or Known Query Attack in DBs), *Chosen Plaintext Attack* (IND-CPA), Chosen Ciphertext Attack (IND-CCA). In this paper, we use the definitions provided in [36] for these classes of attacks. In game-based proofs two parties, the challenger and the adversary, play a cryptographic game. If the adversary cannot guess which of the two texts presented by the challenger is the right plaintext/ciphertext in polynomial time, then the encryption used is IND CPA/CCA secure respectively. This type is employed when one or more cryptosystems are used in a scheme.

Real/Ideal model also known as the simulation paradigm is a technique for provable security first described by Goldreich et. al. in [24]. A protocol P securely computes a functionality if for every adversary A in the real model, there exists an adversary (a simulator) S in the ideal model, such that a real execution and an ideal execution of

P is indistinguishable for the adversary. This is useful when input is provided by both honest and corrupt parties. Schemes which utilize privacy preserving data structures use such paradigms for proving security. Similarly, Universal Composibility (UC-Secure) framework works with an environment, an ideal functionality and a simulator to prove security when composing different protocols [12].

3 Security Primitives

The primitives mentioned here are used for different applications and sometimes as standalones to provide security. But in order to achieve a number of basic database functionalities, the techniques used need to be used in a modular manner or modified. Leakage in encryptions is unavoidable, since the result of a query using that functionality would leak the same property about the data. Analytical queries for deriving data summaries such as *Equality*, *Order*, *Max*, *Min*, *Round*, *Sum*, *Average*, *Limit*, *Count*, *Like*, *Stemming*, and *Wildcard* require order-preserving, homomorphic and searchable encryptions. Boolean queries *And*, *Or*, *Not* require interoperablity with conjunctive and disjunctive search support. In this section, we discuss the aforementioned mechanisms in relation to EDBs. We note, however, how most EDBs can execute only a subset of all possible queries, and therefore will use a a selection, rather than all, of these schemes.

3.1 Property Preserving Encryptions (PPE)

Schemes in the class of Property Preserving Encryptions (PPE) enable functions such as Order, Search, Arithmetics, Equality, Distance, etc. to be performed in EDBs. Three main families of schemes are discussed below: homomorphic encryption; searchable encryptions; and order preserving/revealing encryptions. Secure implementations of these functions are present in most of the surveyed schemes. While encryptions for other functions are available, such standalone methods usually imply significant leakages. For instance, Distance Recoverable Encryption were introduced by Wong et al [63] for calculating k-nearest neighbors in EDBs, using asymmetric scalar-product-preserving encryption. Tex et. al. [59] discuss case studies for encrypting SQL query logs using Xquery and use Distance preserving encryption with PPE.
Homomorphic Encryption (HE) – The property of an encryption that allows algebraic operations to be performed on ciphertext without decryption is called Homomorphic Encryption. A standard HE scheme comprises a minimum of 4 algorithms: Key Generation, Encryption, Decryption and Homomorphic Evaluation for the operations (often addition, $+$ and multiplication, $\cdot$). Based on the evaluations, a HE scheme can be *partially homomorphic* (PHE), supporting one of the two operations above, or *fully homomorphic* (FHE), supporting both operations. In order to improve performance, some schemes are *somewhat homomorphic* with just a limited set of operations before the noise makes the system unreliable. Also, as the encryptions are based on finite structures, in certain schemes homomorphic evaluation is possible only up to a modulus level, due to the increase in size of ciphertext, while others allow evaluations at all levels. Often, a leveled HE (LFHE) scheme is bootstrapped and extended to a fully homomorphic scheme.

Table 1. Types of Homomorphic Encryption

Scheme	**Type**	**Underlying Hard problem**	**Ops**	**Libraries**
Paillier [45]	PHE	Composite Residuosity	$+$	Palisade, Paillier
DGHV [18]	SWHE	LWE	$+, \cdot$	-
FV/BFV [21]	LFHE	LWE	$+, \cdot$	SEAL, Palisade
BGV [11]	LFHE	LWE with no bootstrapping	$+, \cdot$	Helib
HEAAN [15]	LFHE	Approximate LWE	$+, \cdot$	HEANN, Palisade, SEAL, Helib

Since Gentry's FHE scheme in 2009, Post Quantum Public Key Encryption (PKE) HE are the most commonly used and are based on NP-hard problems like learning with errors (LWE) and its variants. Symmetric HE are more prone to attacks. Overall, HE schemes are susceptible to size pattern based attacks (Sec. 5.2).
In recent years, commercial interest in HE has increased, leading to many open source libraries, such as Microsoft Seal [55], IBM's HELib [29], IARPA's PALISADE[1]. The libraries support variants of HE schemes such as BFV [21], HEAAN [15], Paillier [45]. We summarize the schemes and their implementations in Table 1.
Searchable encryptions (SE) – Searchable encryption, another PPE, is a type of functional encryption that allows Search on encrypted data. *Searchable Symmetric Encryption* (SSE) may have many or all of the search features, such as keyword, Boolean, and phrase search queries, as well as stemming, wildcard, and approximate-match searching. SSE provides provable security because of the predetermined leakage function, but requires a scan of the entire DB for every query, unless some Indexing is provided. SSE focuses on data outsourcing, where the data owner is the client. The client parses each input document and performs keyword extraction, followed by a probabilistic *setup algorithm* that outputs the EDB (to be hosted on the server) and the master Key. The *search protocol* requires the EDB, master key and the query (keyword) w as the input. To search for w, the client generates and sends a *trapdoor*, which the server uses to run the search and recover pointers to appropriate documents. *Dynamic SSE* introduced by Kamara et al. [35], are stateful and thus reduce leakage. The setup generates a state to be passed to the Search, Read or Write protocol. Dynamic schemes ensure forward or backward privacy, or both.
Boneh et al. proposed *Public Key Encryption with Keyword Search* (PEKS) based on public key infrastructure using bilinear pairing. The general structure is similar to SSE, but with PKE. The schemes can be for single keyword, conjunctive or subset search. Zhou et al explain on the evolution of PEKS in their paper [66]. A *Private Information Retrieval* (PIR) protocol allows a query to retrieve an element from the DB without revealing the access patterns, relations between columns and their storage blocks. [34] and [31] are inspired by PIR. The schemes which form the basis for SSE systems are in Table 2.
Order Preserving Encryptions (OPE) – Range queries are critical for DBs and Order Preserving Encryptions provide this function. They are numeric and well ordered, therefore lead to significant leakage. Despite this, OPEs are widely studied as detailed in Table 3. Boldyreva et al.'s OPE scheme is the most commonly used due to its ease of implementation [9]. The algorithm constructs a deterministic *Random Order Preserving Function* (ROPF) on a domain of integers where the range is recursively bisected. This

Table 2. Searchable Symmetric Encryption (SSE) schemes

SSE - Year	Feature	Operations
Song et al '00 [57]	Embedding information in pseudo-random bit streams	Single word search
OSPIR OXT'13 [33]	Malicious, Oblivious PRF, Cross Tags(OXT)[14], IND CPA	Conjunctive, Boolean queries
ShadowCrypt '14 [30]	Efficiently searchable encryption with DOM	Standard Full text search
Mimesis Aegis '14 [39]	Efficiently deployable efficiently searchable encryption	Standard Full text search
Sophos '16 [10]	Pseudo-random functions and trap-door permutations	L-Adaptively Secure forward private scheme
Pouliot et al '16 [51]	Bloom Filters	Weighted Graph Matching attacks on EDESE
SisoSPIR '16 [31]	Oblivious Transfer, Secure node search. Real/Ideal	Distributed, Full SSE, Range queries
Saha et al '19 [53]	Ring LWE, IND CPA	Conjunctive, Disjunctive and Threshold queries

Table 3. Order Preserving/Revealing Encryptions (OPE) schemes

OPE	**Feature**	**Security**
Agrawal'09 [5]	Data Buckets transformed to Target distribution	Leaks more than Order
ROPF '09 [9]	ROPF with a hyper-geometric distribution. Modular ROPF by adding an integer modulo M	Leaks > half the bits
mOPE '13 [49]	Mutable - refreshes Cipher text using Merkle hashing	IND-OCPA
ORE '16 [40]	Left encryption: Domain's permutation and hashed key of plaintext. Right: encryptions of the comparison with every other value in the domain	Leaks d-bits of first differing block
Dyer et al'17 [19]	Approximate Integer Common Divisor problem PKE	Window One-wayness
FHOPE '18 [41]	Appx Greatest Common Divisor based Symmetric Fully Homomorphic OPE	IND-HOCPA

method reveals almost half of the bits in the ciphertext. Security for such encryptions is proved by Indistinguishability under Ordered Chosen Plaintext attacks (IND-OCPA) where an OPE leaks nothing but the order of the ciphertext. Other simulation-based security proofs are provided where IND-OCPA is considered insecure (Sec. 5.2).

3.2 Oblivious RAM

Initially introduced for secure multi-party computation, Oblivious Random Access Memory enables the oblivious execution of the RAM algorithm while hiding the access pattern from the server and memory. ORAM continuously reshuffles and re-encrypts (Symmetric or FHE) the data in blocks as they are accessed with fresh randomness using a position map with bandwidth and block size as parameters. The blocks are stored in binary search trees. *Path ORAM* [58] is the most used and practical version of ORAM. Here, when a block is read from the server, the entire path to the mapped leaf is read, the requested block is remapped to another leaf, and then the path that was just read is written back to the server. Garg et. al. [23] use *TWORAM* with two rounds of oblivious exchange between client and server. It does not support inserts/deletes or hides result sizes. In ORAM based schemes, the message length from the server to the user, as the result of query execution, is proportional to the number of records matching the query, therefore causing leakage of the size pattern. Therefore ORAM-based schemes are vulnerable to size pattern-based attacks, e.g., count attack (Sec. 5.2).

3.3 Enclaves

Enclaves are software programs written into a trusted portion of the hardware for secure storage and computation of data. The hardware manufacturer becomes a trusted third party, with potential access to the keys and code. Some of the commercially available Trusted Execution Environments are Intel Software Guard Extensions (SGX) [16], ARM's TrustZone [3] to be used in devices like Raspberry Pi and AMD Secure Memory Encryption which is enabled at BIOS and uses a single key generated by the AMD Secure Processor at boot for transparent encryption. Our focus shall rest on SGX because of its usage in a majority of the Hardware based EDBs.

SGX is a set of new x86 instructions that enable code isolation within virtual containers. The attack models assume untrusted servers. The three main functionalities are: *Isolation*, *Sealing* and *Attestation*. SGX isolates enclave code and data in the Processor Reserved Memory (PRM). Cryptographic keys are owned by the trusted processor. SGX uses AES-GCM to encrypt with additional authenticated data for sealing. An enclave can derive a Seal Key, specific to the enclave identity, from the Root Seal Key and this can be used to encrypt/authenticate and store data in untrusted memory. A special signing key and instructions are used for attestation to ensure unforgeable reports Attestation can be done locally or remotely.

Enclaves occupy a fixed and limited part of the memory and so an application is built around the SGX. Intel becomes compulsory hardware to run SGX,which is . The performance degrades around 4 times when a DBMS is run on the SGX.

4 Secure Database Schemes

In this section, we give a brief discursive description of current encrypted database schemes and their implementation, focusing on security and functionality. In previous work, Fuller et. al. [22] provided a systematization of knowledge for cryptographically protected database search, but do not include hardware attacks. We divide the schemes into three categories – encrypted conventional databases, privacy-preserving data structure based schemes, and hardware-based schemes. The schemes were chosen for the survey based on novelty, relevance, extent of functionality and implementation.

For key management, most schemes assume that only the clients possess the master key. Proxies or trusted execution environments hold the encryption schemes for the particular components. For members of the organisation, access control is the primary way to ensure key distribution if the scheme supports multiple users. Also, the number of servers used in each model is normally not specified.

4.1 Encrypted Conventional Databases

CryptDB [50] is an DBMS that supports all of the standard SQL over encrypted data without requiring any client-side query processing, modifications to existing DBMS or legacy applications and offloads virtually all query processing to the server, using a proxy server to process encrypted queries. CryptDB uses the security of existing cryptosystems like AES-CBC/CMC, ROPF[9] and Paillier, forming an *Onion Layering Model* where encryptions and their respective keys are layered based on the type of queries. Therefore, CryptDB cannot perform Complex Join operations and lacks rich functionality though it primarily focuses on relational data. Although CryptDB shows good performance with TPC-C queries, it is 26% slower than performance on MySQL.
Monomi [60], based on the same Onion Layering model, delegates Queries between a trusted client and an untrusted server using a query planner and designer, based on the encryption and order of the Queries. The scheme is optimized based on Cost of Storage and Transaction, yet reserves as many tasks as possible for the server. This improves interoperability. But the data leaks order related properties in steady state and on querying and the adversary is assumed to have full access to the DB.
Arx [48] assumes the non-malicious non-curious CSP model with intrusion detection to prevent attackers from observing and logging queries over time, but fears "steal-and-run" attacks. Arx has the same building blocks structure along with *ArxAgg* for addition and two new DB indices, *ArxRange* for Range queries and *ArxEq* for equality queries, which uses treap like structure, whose nodes destroy and rebuild themselves for untraceability. Queries on these indices reveal only a limited per-query access pattern. Arx is built against leakage attacks based on frequency and order. Overheads are only about 6% to 15% more than on specific unencrypted data and deployed on MongoDB and similar databases to show functionality. Arx uses a two-round protocol to hide the relationship between the node values of the range query index and the DB rows holding that value to avoid snapshot attacks. Nevertheless, the leakage is sufficient to recover the values in the index using a variant of the bipartite matching attacks.
Seabed [46], built by Microsoft, uses Additively Symmetric Homomorphic Encryption (ASHE) for performing analytics on big data. The adversary is assumed to be with honest but curious. To prevent frequency attacks, Splayed ASHE (SPLASHE) uses deterministic encryption with padding for infrequent plaintext values, rather than a dedicated column in the schema. Seabed is built on Apache Spark. The performance-schema will leak a query histogram for only the frequently occurring values. However, a partial histogram could be reconstructed from the logs. With other leakage about frequent values from query patterns, damaging cross-column inference attacks can be performed. [28]
DBMask [54] can efficiently handle large databases even when user dynamics change. The Proxy server acts as a mask for the data attribute based models with access control and cryptographic constructors like Attribute based Group Key Management(AB-GKM) are introduced for functionality in DBMask. No changes are made to the DBMS engine. An access tree is formed and thresholds are set with Shamir's Secret Sharing polynomials to derive group keys for querying. Privacy Preserving Numerical comparisons, key searches, joins are derivatives Hence, DBMask can perform access control and predicate matching at the time of query processing by adding predicates to the query being executed.

P-McDB [17] – Cui et al. proposed a dynamic SSE scheme for Privacy Preserving Multi Cloud DB which uses multiple clouds for storage and search, Index and witness and, re-randomize and shuffle. P-McDb is meant for multiple users and data cannot be re-encrypted on user revocation. In this system, the clouds are Honest but curious, capable of injecting malicious records. It supports partial searches of encrypted records. The communication between them could introduce more delays. The model is efficient enabling full search. The dual cloud system prevents inference attacks.
Encrypted NoSQL databases – EncKV [65] uses Symmetric Searchable Encryption and ORE [40] to accomplish the primary task of the *Key Value Stores*. EncKV is tested on the Redis Cluster with the Amazon EC2 public cloud platform. *Graph Databases* for Encryption for Approximate Shortest Distance Queries (GRECS) [42] uses somewhat homomorphic and symmetric encryptions for secure Graph operations. GraphSE2 [38] provides scalable social search by decomposing queries and using interchangeable protocols, and demonstrate using Azure Cloud. Wiese et al [62] present CloudDB guard adapting an Onion Layering model with PPE for *Wide Column Stores* and test their scheme on the Enron email dataset. The scheme presents optimised reading, writing and storage efficient schemes with low performance loss.

4.2 Privacy Preserving Data Structure based Schemes

Structures such as B-trees, B+ trees are used in DBs for indexing purposes. Queries return observable states in memory, along with which parts of the DB are touched (called access patterns), frequency. To avoid this, EDBs often rely on modified data structures or mechanisms such as Inverted Index, Oblivious Index, Bloom filter trees, AVL Trees or SE based index, ORAM, etc.
Blind Seer [47] – Pappas et al provide a framework (BLoom filter INDex SEarch of Encrypted Results) for developing secure and oblivious search using Bloom filter search trees, garbled circuits and HE. *Garbled circuits* (GC) [64] enable oblivious exchange of information between two parties. Bloom Filters are helpful in privately checking if an element belongs to a set, using hashes allowing no false negatives. A DB is permutated with a Pseudo-random Generator on a Random string XOR'ed with the Record which forms a *Bloom filter search tree*. The DB and Search tree are encrypted using an *additive homomorphism* in the server and sent to the Index server along with the public key. A Query gets transformed into a *Boolean Circuit*, which the index server securely computes, garbled and sent to the server, where the data is retrieved from the Search Tree and sent back. The transaction queries can be executed in constant time, but the periodic re-indexing that merges the temporary Bloom filter list to the tree on updations is expensive.
Blind Seer supports analytical Boolean queries and provides security against Access pattern leakage. Bloom filter search tree supports range queries without OPE. Blind Seer claims to be 1.2 - 3 times slower than MySQL.
Oblix [43] – Mishra et al proposed a highly scalable Doubly *Oblivious Search Index* using SGX and Doubly Oblivious RAM. The client's access to the server's memory and to its own local memory are obliviously accessed and hence the double obliviousness. The attack model assumes a malicious attacker, performing any hardware attacks on the enclave and code modifications but the processor cannot be harmed. Oblix supports

multi user access to data without revealing the other party's query results or changes. It creates Doubly Oblivious Sorted Multi-maps (DOSM) to store key-value pairs and Doubly Oblivious Data Structures (DODS) to store the EDB. The AVL binary trees are modified to rebalance only after a fixed number of nodes have been accessed in the previous phase, so the adversary can predict when rebalancing begins, and cache the nodes accessed. Oblix achieves protection against modification attacks and access pattern leakage for both Data and code via Merkle hash trees by using Intel SGX's built-in integrity tree and by employing a separate hash tree for data stored outside.

4.3 Trusted Hardware Based Schemes

TrustedDB [8] – Bajaj et al were one of the first to propose an EDBMS based on Trusted hardware. TrustedDB is built on IBM 4764 series of secure co-processors (SCPU) with *Sensitive attributes* processed by SQLite and a commodity Server for rich MySQL DBs. Query processing engines are run on both the server and SCPU. Private attributes can only be decrypted by the client or by the SCPU. The main CPU DBMS is an unmodified MySQL 14.12 engine, but can be substituted. An ample amount of changes are required to integrate the hardware to cooperate with each other.
CipherBase [7] is an extension of Microsoft SQL Server, designed to protect data against admins with root access privileges. Cipherbase integrates a custom designed FPGA to act as a secure DB processor for the *Trusted Module*(TM) a submodule for core operations over encrypted Data, alongside an Untrusted DB Server Module(UM), The Cipherbase query plan runtime system aids in shipping tuples encrypted with AES-ECB, AES-CBC or ROPF[9], from UM to TM then to decrypt, process, and re-encrypt these tuples in the TM, and ship results back. It provides end to end functionality. Computations runs an order of magnitude slower than regular processors.
ObliDB [20] is an enclave-based oblivious DB engine that efficiently runs general relational read workloads with an SGX implemented. ObliDB can store data with no obliviousness or indexed, or both combined. Based on the query's characteristics, like the amount of DB covered and choice of storage, the query planner chooses one of the available select algorithm for Select, Aggregate and Join queries. ObliDB uses Path-ORAM to cover the access patterns of the queries to avoid the disadvantages of CipherBase and TrustedDB. It is 2.6% slower than SparkSQL.
Stealthdb [61] creates three enclaves on the server: For client authentication, query pre-processing and operation. Based on the importance, columns are marked with an Encrypted or Unencrypted index. StealthDB with encrypted IDs incurs 4.2 times the overhead to PostgreSQL 9.6, for even large DBs.
EnclaveDB [52] is built with an enclave based on Hekaton, optimized for OLTP workloads where data fits in the available memory. The model assumes threats from any party controlling the DB server. It does not consider Access pattern attacks. The implementation uses AES-GCM, a high-performance AEAD scheme, and Oblivious transfers with Software Guard Extensions for security.

Table 4. Performance

EDBMS	Focus	Type	Benchmark	Code
CryptDB'11 [50]	S	RDBMS/NoSQL	TPC-C 31, TPC-H 4	[50]
Monomi'13 [60]	S/C	RDBMS/NoSQL	TPC-C 31, TPC-H 19	[60]
Arx'17 [48]	S	RDBMS/MongoDB	TPC-C 30	-
Seabed'16 [46]	S/C/H	Apache Spark	Ad Analytics, MDX API, TPC-DS 99	-
P-McDB'19 [17]	S	RDBMS, Cloud	TPC-H	-
DBMask'16 [54]	S/C	RDBMS	TPC-C, CRIS	-
Oblix'18 [43]	S	Search Index, Key-Value, Scalable	ZeroTrace, Google Key Transparency, Signal	-
Blind Seer [47]	S/C	Search Index	US Census and "Call of the wild" based	-
TrustedDB'13 [8]	TH	SQLite/MySQL	TPC-H	-
Cipherbase'13 [7]	TCH	MS SQL Server	TPC-C 31	-
ObliDB'17 [20]	TH	MongoDB	Big Data	[20]
StealthDB'17 [61]	TH	PostgreSQL	TPC-C 31	[61]
EnclaveDB'18 [52]	TH	Hekaton	TPC-C, TATP	-

Focus(Query Handling) - S: Server, C: Client, TH: Trusted Hardware

5 Discussion

In this section, we discuss and compare the schemes which we presented and categorized above. Section 5.1 discusses performances, compared based on run time and ability to execute queries in *benchmarks* including TPC-C, TPC-H, and TATP, Enron email dataset, Big Data [2]. Benchmarks simulate a complete computing environment to execute queries with varying degrees of complexity, centred around transactions, decision support systems, OLAP, etc. Table 4 lists the benchmarks relevant to each scheme.

Section 5.2 discusses the schemes from a security point of view, and identifies the main attacks. The security components included in each scheme are listed in Table 5.

5.1 Performance

The performance comparison presented here is based on the results included in the original papers. It would be outside the scope of this review, as well as technically challenging to reproduce the original results independently: the software based schemes considered have different basic components that are not easily integrated, while the hardware based schemes make use of equipment that is not readily available and whose configurations are hard to reconstruct. The code is available as open source only for the schemes mentioned in the Table 5.1, which is not sufficient for elaborate analysis. Comparisons in terms of complexity is usually done with respect to MySQL or other popular DBMS and are stated in the scheme descriptions.

Conventional EDB schemes like CryptDB, DBMask are feasible because of their modular structure and are faster as they resemble traditional DBs. These schemes, however, try to process full queries in the proxy or the server instead of delegating a portion to the client. This reduces the functionality. For example, a query with computation and comparison cannot be performed together as the encryptions do not allow this. In comparison, privacy-preserving data structure based systems are not immediately prac-

Table 5. Security Comparisons

EDBMS	**Adv**	**Encryptions, ORAM**	**Hardware**	**Security**
CryptDB	HbC	Paillier[45], AES, ROPF [9], mOPE[49]	-	IND CPA
Monomi	HbC	Paillier[45], ROPF[9], AES	-	IND-CPA
Seabed	HbC	Splayed Additively SHE	-	SS/ IND-CPA
DBMask	HbC	AB-GKM, AES, ROPF [9], Song SSE[57]	-	SS
P-MCDB	Mal	AES-ECB, Asghar OPE,	-	Real/Ideal
Arx	HbC	GC, AES, mOPE, ArxEq, ArxAgg	-	SS/IND-CPA
Blind Seer	Mal	GC, Bloom Filter Tree	-	SS
Oblix	Mal	D-Oblivious sorted multimaps/Data Structure, DORAM	Enclave	Real/Ideal
TrustedDB	HbC	AES,RSA,SHA	SCPU	Hardware
Cipherbase	HbC	AES, Paillier, Column level, ROPF, ORAM	FPGA	IND-CPA
ObliDB	Mal	Path-ORAM	SGX	Real/Ideal
EnclaveDB	Mal	AES	SGX	IND-CPA/CTXT
StealthDB	HbC	AES GCM	SGX	SS

HbC: Honest but Curious, Mal: Malicious, SS: Semantic Security

tical, due to their focus on obscuring the access pattern, including Oblix, Blind Seer, SisoSPIR. Here, every query has to search the entire DB to retrieve even one element.

Hardware-based solutions require utilities like SGX from all parties, which is not always ideal. Storage and Memory in SGX are not scalable over time. Moreover, making the hardware and the encryptions work together using an existing DBMS framework can be difficult, as stated in [8], and [7].

The distinct features of each scheme make them difficult to collectively compare and rank. A secure scheme may not be fully functional and vice versa. Each scheme has a different number of servers, presence or absence of proxies, integration of protocols, hardware, etc. Therefore, each of the three categories are compared using some standard parameters and benchmarks in Table 4

Industry: Encrypted Databases have become commercialised for cloud computing platforms majorly and products are available for use, such as Bitglass Cloud Encryption, CipherCloud, McAfee MVISION Cloud from Skyhigh Networks, Microsoft Always Encrypted (SQL Server and Azure Plus enclaves), Netskope, Symantec CloudSOC, Google Encrypted BigQuery. A majority of the products feature secure search and retrieval using a combination of access control and established cryptosystems like AES-256. Schemes like Bitglass, also support Range Queries. However, advanced EDBs are still in research and experimental stages, e.g. Encrypted BigQuery.

5.2 Attacks and Security

EDBs support properties requiring ciphertext manipulation, thereby refusing IND-CCA. Hence most schemes prove security through Real/Ideal or IND-CPA though this does not guarantee the absence of leakage, which is discussed here as any piece of information that the user derives more than what the returned query result implies. It can occur online and offline. We classify leakage based on where it occurs: in the memory, or during computation.

Table 6. Attacks based on Leakage levels and the affected schemes[13]

Leakage	**Attacks**	**Schemes**
$\mathcal{L}_4$ - Full-text Substitution Cipher	Full DB Reconstruction	Song et. al. [57]
$\mathcal{L}_3$ - PPE - Data Distribution (Occurrence leakage)	Inference, IKK [32], Count, Record Injection	Conventional EDB: CryptDB[50], Cipherbase[7], Monomi[60], Seabed[46]
$\mathcal{L}_2$- DET, Appended PRF - Access, Size and Search Patterns	IKK, Count, Record Injection	Unencrypted Indexes - Blind seer [47], Arx [48], etc
$\mathcal{L}_1$ - HE/ORAM - Communication Volume	Count	Encrypted Index/Result Length Hiding - Oblix [43], SisoSPIR [31]

Leakage from Memory – *Inference attacks*, termed IKK attack, proposed by Islam et al. [32] use frequency analysis on SSE with only access pattern disclosure and full document/partial query knowledge. Naveed et. al. [44] performed inference attacks on EDBs and state that it is better to offload data to the client and perform queries locally. If the data distribution of a particular column is known, the column can be reconstructed in $O(N^4 \log N)$ queries where N is the number of entries. The attack is performed using frequency analysis on Static DB without any queries. Another passive attack applicable for most schemes is the *Count attack*, where an adversary could recover queries by counting the number of matched records even if the encrypted records are semantically secure. The attacker with full knowledge of query distribution, sees how many records are returned in response to a query and identifies it if the number is unique, hence matching every returned record with that keyword [13].
Leakage from Computation – *Leakage abuse attacks* are common and cannot be eliminated in SE and PPE. In order to evaluate leakages from searchable encryptions, Curtmola et. al. define a series of leakage levels which was then characterised by Cash et al.[13] as $\mathcal{L}_1$–$\mathcal{L}_4$, from least to most leakage. We present leakage levels and related attacks in Table 6.
Any snapshot of the system contains information about search tokens, past queries, workloads, and access patterns from logs in the byte level. With this knowledge, the attacker who compromised the disk can reconstruct queries that were used to modify the DB [28]. Prior knowledge of the plaintext at various degrees can lead to *Injection* of malicious records to reveal data or even without the knowledge, the record can simply alter the results of the analytics performed on the data. Only some schemes, like Oblix, can handle injections and malicious entities (Table 5).
By using just the *volume* of the range queries, databases can be reconstructed in at most $O(N^4 logN)$ time for N unique entries in the DB [26]. Lacharite et al. [37] consider Full and Approximate *Database Reconstruction* using range queries and access pattern leakage from just $O(N)$ queries. Grubbs et al. [27] perform ϵ-Approximate Database reconstruction (ADR) through access pattern leakage relating it closely to statistical learning theory and *no Query distribution Knowledge* with ϵ admissible error. On neglecting the extremities, ADR can be performed within $O(\epsilon^{-1} log(\epsilon^{-}1))$ time, which implies that the complexity is invariant of the number of items in the DB but focuses on the error ϵ.
Attacks on Hardware – Side channel attacks, as well as denial of service are applicable to the TEE as well [16]. The licensing allows Intel to force itself as an intermediary

for their enclaves. The software used for isolating the memory to create the SGX can be made malicious using privilege escalation. Cache timing attacks observe the time differences between accessing a cached and uncached memory location Private caches can partially prevent this. Simple power analysis can correlate power consumption and type of query executed. Memory mapping attacks uses address retrieval for page tables. Some attacks can be solved by pairing with an ORAM or PIR.

5.3 Future research directions

As highlighted in our discussion, research gaps and open questions are evident in all three types of schemes: malicious attackers are often not defendable in the constructions; EDBs face issues with the dynamic updations; search features like wildcard, stemming, concatenation are not refined or available altogether. While SSEs focus on search, only schemes like SisoSPIR [31] give a near complete and secure framework.

Artificial intelligence and machine learning algorithms on DBs are gaining interest. In general, they require vectorization for efficient computation and tackling natural language processing problems. Secure matrix multiplication is necessary to deploy neural networks even after reducing the activation functions to polynomials like ML Confidential [25]. NewSQL DBs require privacy preserving matrix multiplication, while NoSQL structures demand array multiplication, graph operations as well.

Finally, in order to enable meaningful comparison between schemes, a general framework for leakage quantification would prove very useful, and would contribute to improve security of new schemes.

6 Conclusions

In this survey, we highlight how several existing schemes are vulnerable to a number of known attacks. Finding the delicate compromise between performance and security, conforming to *Anderson's Law*, can only give a viable but imperfect solution. Our discussion points to future research directions, aimed on one hand to secure schemes in the shorter term, and on the other hand to improve the real-world applicability of more secure schemes in the long term.

This survey focuses in particular on the tools required to build an EDB, along with the pros and cons of using the different methods. When designing a future EDB, the aim can be to increase security, functionality or both. As the conventional systems are weaker in security because of PPE leakage, index based schemes are preferred, though they are not immediately viable. An index based scheme supporting access pattern hiding, updates, range queries, injection, etc, would represent a significant breakthrough.

References

1. PALISADE Lattice Cryptography Lib. (ver.1.9.2). `http://palisade-crypto.org`
2. Tpc benchmarks, `http://www.tpc.org/information/benchmarks.asp`
3. ARM security technology building a secure system using trustzone technology (rev. C). Tech. rep., ARM (2009)

4. Agrawal, D., El Abbadi, A., Emekçi, F., Metwally, A.: Database management as a service: Challenges and opportunities. In: IEEE ICDE. pp. 1709–1716 (2009)
5. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order-preserving encryption for numeric data. In: ACM SIGMOD International Conference on Management of Data. pp. 563–574 (2004)
6. Anderson, R.J.: Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley Publishing, 2 edn. (2008)
7. Arasu, A., Eguro, K., Joglekar, M., Kaushik, R., Kossmann, D., Ramamurthy, R.: Transaction processing on confidential data using cipherbase. In: IEEE ICDE. pp. 435–446 (2015)
8. Bajaj, S., Sion, R.: Trusteddb: A trusted hardware-based database with privacy and data confidentiality. IEEE Trans. Knowl. Data Eng. 26(3), 752–765 (2014)
9. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: EuroCrypt 2009. pp. 224–241 (2009)
10. Bost, R.: $\sum o\varphi o\varsigma$: Forward secure searchable encryption. In: ACM SIGSAC CCS. pp. 1143–1154. ACM (2016)
11. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Electron. Colloquium Comput. Complex. 18, 111 (2011)
12. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Foundations of Computer Science, FOCS 2001. pp. 136–145. IEEE (2001)
13. Cash, D., Grubbs, P., Perry, J., Ristenpart, T.: Leakage-abuse attacks against searchable encryption. In: ACM SIGSAC CCS. pp. 668–679. ACM (2015)
14. Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.C., Steiner, M.: Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In: CRYPTO 2013, vol. 8042, pp. 353–373 (2013)
15. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: AsiaCrypt 2017. LNCS, vol. 10624, pp. 409–437. Springer (2017)
16. Costan, V., Devadas, S.: Intel SGX explained. IACR Cryptol. ePrint Arch. 2016, 86 (2016)
17. Cui, S., Song, X., Asghar, M.R., Galbraith, S.D., Russello, G.: Privacy-preserving searchable databases with controllable leakage. CoRR abs/1909.11624 (2019)
18. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: EuroCrypt 2010 LNCS, vol. 6110, pp. 24–43. Springer (2010)
19. Dyer, J., Dyer, M.E., Djemame, K.: Order-preserving encryption using approximate common divisors. J. Inf. Secur. Appl. 49 (2019)
20. Eskandarian, S., Zaharia, M.: Oblidb: Oblivious query processing for secure databases. PVLDB 13(2), 169–183 (2019)
21. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. 2012, 144 (2012)
22. Fuller, B., Varia, M., Yerukhimovich, A., Shen, E., Hamlin, A., Gadepally, V., Shay, R., Mitchell, J.D., Cunningham, R.K.: Sok: Cryptographically protected database search. In: IEEE Security & Privacy. pp. 172–191 (2017)
23. Garg, S., Mohassel, P., Papamanthou, C.: TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In: CRYPTO. LNCS, vol. 9816, pp. 563–592. Springer (2016)
24. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game, or a completeness theorem for protocols with honest majority. In: Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, pp. 307–328. ACM (2019)
25. Graepel, T., Lauter, K.E., Naehrig, M.: ML confidential: Machine learning on encrypted data. In: Information Security and Cryptology - ICISC 2012. vol. 7839, pp. 1–21. Springer (2012)
26. Grubbs, P., Lacharité, M., Minaud, B., Paterson, K.: Pump up the volume: Practical database reconstruction from volume leakage on range queries. In: ACM CCS. pp. 315–331 (2018)

27. Grubbs, P., Lacharite, M.S., Minaud, B., Paterson, K.G.: Learning to Reconstruct: Statistical Learning Theory and Encrypted Database Attacks. In: IEEE Security & Privacy. pp. 1067–1083 (2019)
28. Grubbs, P., Ristenpart, T., Shmatikov, V.: Why your encrypted database is not secure. In: 16th Workshop on Hot Topics in Operating Systems. pp. 162–168 (2017)
29. Halevi, S., Shoup, V.: Algorithms in helib. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 554–571. Springer (2014)
30. He, W., Akhawe, D., Jain, S., Shi, E., Song, D.X.: Shadowcrypt: Encrypted web applications for everyone. In: ACM SIGSAC. pp. 1028–1039. ACM (2014)
31. Ishai, Y., Kushilevitz, E., Lu, S., Ostrovsky, R.: Private large-scale databases with distributed searchable symmetric encryption. In: CT-RSA. vol. 9610, pp. 90–107. Springer (2016)
32. Islam, M.S., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In: NDSS. The Internet Society (2012)
33. Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M., Steiner, M.: Outsourced symmetric private information retrieval. In: ACM SIGSAC CCS'13. pp. 875–888. ACM (2013)
34. Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M., Steiner, M.: Outsourced symmetric private information retrieval. In: ACM SIGSAC CCS'13. pp. 875–888. ACM (2013)
35. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) the ACM CCS'12. pp. 965–976. ACM (2012)
36. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, 2nd Edition. CRC Press (2014)
37. Lacharité, M., Minaud, B., Paterson, K.G.: Improved reconstruction attacks on encrypted data using range query leakage. In: 2018 IEEE Security & Privacy. pp. 297–314 (2018)
38. Lai, S., Yuan, X., Sun, S., Liu, J.K., Liu, Y., Liu, D.: Graphse^2: An encrypted graph database for privacy-preserving social search. In: ACM Security AsiaCCS. pp. 41–54. ACM (2019)
39. Lau, B., Chung, S.P., Song, C., Jang, Y., Lee, W., Boldyreva, A.: Mimesis aegis: A mimicry privacy shield-a system's approach to data privacy on public cloud. In: 23rd USENIX Security Symposium. pp. 33–48. USENIX Association (2014)
40. Lewi, K., Wu, D.J.: Order-Revealing Encryption: New Constructions, Applications, and Lower Bounds. In: ACM SIGSAC- CCS'16. ACM Press (2016)
41. Liu, G., Yang, G., Wang, H., Xiang, Y., Dai, H.: A Novel Secure Scheme for Supporting Complex SQL Queries over Encrypted Databases in Cloud Computing. Security and Communication Networks (Jul 2018)
42. Meng, X., Kamara, S., Nissim, K., Kollios, G.: GRECS: Graph encryption for approximate shortest distance queries. In: 22nd ACM SIGSAC. ACM (2015)
43. Mishra, P., Poddar, R., Chen, J., Chiesa, A., Popa, R.A.: Oblix: An efficient oblivious search index. In: 2018 IEEE Symposium on Security and Privacy. pp. 279–296 (2018)
44. Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property preserving encrypted databases. In: 22nd ACM SIGSAC-CCS '15. ACM Press (2015)
45. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EuroCrypt 1999. pp. 223–238 (1999)
46. Papadimitriou, A., Bhagwan, R., Chandran, N., Ramjee, R., Haeberlen, A., Singh, H., Modi, A., Badrinarayanan, S.: Big data analytics over encrypted datasets with seabed. In: 12th USENIX Symposium on OS Design and Implementation. USENIX Association (2016)
47. Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Choi, S.G., George, W., Keromytis, A., Bellovin, S.: Blind seer: A Scalable Private DBMS. In: 2014 IEEE Security & Privacy. pp. 359–374. IEEE (2014)
48. Poddar, R., Boelter, T., Popa, R.A.: Arx: an encrypted database using semantically secure encryption. Proceedings of the VLDB Endowment 12(11), 1664–1678 (Jul 2019)
49. Popa, R.A., Li, F.H., Zeldovich, N.: An ideal-security protocol for order-preserving encoding. In: 2013 IEEE Symposium on Security and Privacy. pp. 463–477 (2013)

50. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: Cryptdb: processing queries on an encrypted database. Commun. ACM 55(9), 103–111 (2012)
51. Pouliot, D., Wright, C.V.: The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In: ACM SIGSAC. pp. 1341–1352. ACM (2016)
52. Priebe, C., Vaswani, K., Costa, M.: Enclavedb: A secure database using SGX. In: 2018 IEEE Symposium on Security and Privacy. pp. 264–278 (2018)
53. Saha, T.K., Rathee, M., Koshiba, T.: Efficient private database queries using ring-lwe somewhat homomorphic encryption. J. Inf. Secur. Appl. 49 (2019)
54. Sarfraz, M.I., Nabeel, M., Cao, J., Bertino, E.: Dbmask: Fine-grained access control on encrypted relational databases. Trans. Data Priv. 9(3), 187–214 (2016)
55. Microsoft SEAL (release 3.5). `https://github.com/Microsoft/SEAL` (Apr 2020), microsoft Research, Redmond, WA.
56. Shay, R., Blumenthal, U., Gadepally, V., Hamlin, A., Mitchell, J., Cunningham, R.: Don't even ask: Database access control through query control. SIGMOD Rec. 47(3), 17–22 (2018)
57. Song, D.X., Wagner, D.A., Perrig, A.: Practical techniques for searches on encrypted data. In: 2000 IEEE Symposium on Security and Privacy. pp. 44–55 (2000)
58. Stefanov, E., van Dijk, M., Shi, E., Chan, T.H., Fletcher, C.W., Yu, X., Devadas, S.: Path ORAM: an extremely simple oblivious RAM protocol. J. ACM 65(4), 18:1–18:26 (2018)
59. Tex, C., Schäler, M., Böhm, K.: Towards meaningful distance-preserving encryption. In: 30th Intl. Conf. on Scientific and Statistical Database Management, SSDBM. pp. 2:1–2:12 (2018)
60. Tu, S., Kaashoek, M.F., Madden, S., Zeldovich, N.: Processing analytical queries over encrypted data. Proceedings of the VLDB Endowment 6, 289–300 (Mar 2013)
61. Vinayagamurthy, D., Gribov, A., Gorbunov, S.: Stealthdb: a scalable encrypted database with full SQL query support. PoPETs 2019(3), 370–388 (2019)
62. Wiese, L., Waage, T., Brenner, M.: Clouddbguard: A framework for encrypted data storage in nosql wide column stores. Data Knowl. Eng. 126, 101732 (2020)
63. Wong, W.K., Cheung, D.W., Kao, B., Mamoulis, N.: Secure knn computation on encrypted databases. In: ACM SIGMOD '09. pp. 139–152 (2009)
64. Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science. pp. 160–164. IEEE Computer Society (1982)
65. Yuan, X., Guo, Y., Wang, X., Wang, C., Li, B., Jia, X.: Enckv: An encrypted key-value store with rich queries. In: ACM Asia CCS. pp. 423–435 (2017)
66. Zhou, Y., Li, N., Tian, Y., An, D., Wang, L.: Public key encryption with keyword search in cloud: A survey. Entropy 22(4), 421 (2020)