Security and Privacy Implications on Database Systems in Big Data Era: A Survey

G. Dumindu Samaraweera, *Student Member, IEEE* and J. Morris Chang, *Senior Member, IEEE*

Abstract—For over many decades, relational database model has been considered as the leading model for data storage and management. However, as the Big Data explosion has generated a large volume of data, alternative models like NoSQL and NewSQL have emerged. With the advancement of communication technology, these database systems have given the potential to change the existing architecture from centralized mechanism to distributed in nature, to deploy as cloud-based solutions. Though all of these evolving technologies mostly focus on performance guarantees, it is still being a major concern how these systems can ensure the security and privacy of the information they handle. Different datastores support different types of integrated security mechanisms, however, most of the non-relational database systems have overlooked the security requirements of modern Big Data applications. This paper reviews security implementations in today's leading database models giving more emphasis on security and privacy attributes. A set of standard security mechanisms have been identified and evaluated based on different security classifications. Further, it provides a thorough review and a comprehensive analysis on maturity of security and privacy implementations in these database models along with future directions/enhancements so that data owners can decide on most appropriate datastore for their data-driven Big Data applications.

Index Terms—Big data, database systems, attacks, threats, security, privacy, performance

1 Introduction

Every new wave of computing technology from mainframe era to Big Data era has accelerated data growth in numerous ways. Thus, the volume increase of data in a fast pace has been identified as one of the ongoing challenges for any database system [1]. Starting from the early stages, relational database systems have been considered as the key data management technology for many organizations and it has been served as the backbone for structured data. However, the increased volume and variety of the data types has led to existence of alternative database designs that can even facilitate semi-structured and unstructured data without compromising the performance of the database engine. As a result, NoSQL models have given the rise. However, despite the fact that usage of DBMS for data management, data analytics also plays a major role in any organization, particularly with fast growth of data. To facilitate such data analytics with large volume of data, the idea of combining strong Atomicity, Consistency, Isolation and Durability (ACID) guarantees of relational database systems together with performance guarantees of NoSQL models has been proposed and termed as NewSQL which is held to be one of the emerging database models for future data-driven applications.

When the organizations increase their usage of database systems as the key data management technology, especially with Big Data management, the security of the information managed by these systems becomes vital. Confidentiality, Integrity and Availability (CIA) are considered as the foundation of data security and privacy, but whether modern database systems can exhibit these properties in their architectures is still a major concern. On the other hand, moving database infrastructures from on-premise to distributed cloud-based architectures has increased the risk of security and privacy breaches. Thus, majority of organizations, do not store mission critical data in the cloud as they argue there is a higher degree of confidence of security when the data stored on-site [1]. Hence, utilizing the state-of-the-art performance benefits provided by the database systems for Big Data applications, without compromising the security, is the new challenge for modern-day database systems. There has been a lot of research in the comparison of different datastores over the past [2], [3], [4] based on performance and quality attributes; yet, there has been no security and privacy focused classification of different database models giving more emphasize on security/privacy aspects of database systems.

This article aims to fulfill this gap by providing a thorough and comprehensive analysis on maturity of security (and privacy) implementations of today's leading database models, and their competency for serving modern Big Data applications by investigating the existing security models of different database systems and current efforts of the research community towards strengthening these mechanisms. At first, authors have investigated and identified set of industry standard technical approaches and the mechanisms that can

• *The authors are with the Department of Electrical Engineering, University of South Florida, 4202 E. Fowler Avenue, Tampa, FL 33620 USA. E-mail: samaraweera@mail.usf.edu, chang5@usf.edu.*

Manuscript received 16 July 2018; revised 17 June 2019; accepted 16 July 2019. Date of publication 18 July 2019; date of current version 7 Dec. 2020. (Corresponding author: G. Dumindu Samaraweera.) Recommended for acceptance by L. Chen. Digital Object Identifier no. 10.1109/TKDE.2019.2929794

be utilized to implement security on database systems. Then, modern database systems (that are actively being discussed) have been classified in to multiple categories based on their usage and popularity. Thereafter, those datastores have been individually evaluated based on the identified security mechanisms and an extensive comparison has been provided. As per the key findings of this survey, even though relational database systems are facilitated with reasonably strong security mechanisms that can ensure the protection for most of the modern-day Big Data applications, a larger fraction of NoSQL and New SQL systems are still lacking strong security guarantees. Therefore authors believe that it is the right time to properly revisit the security offerings of modern database solutions toward designing a robust security framework for next generation database systems.

The rest of the survey is organized as follows: Section 2 presents summary of underlying technologies of different database systems and Section 3 discusses about the threats, vulnerabilities and adversarial models that can lead to data breaches in database systems. We extend the security discussion in Section 4 with a comprehensive analysis and an evaluation of security and privacy mechanisms available with leading data management systems. Finally, paper concludes with Section 5 providing further observations for future work.

2 Underlying Data Processing Mechanisms of Database Systems

Database systems have been evolving over the last few decades attributed to couple of driving factors, mainly, advances in hardware, increased volume expansion of data, emerging applications and so on. In order to understand the synergies of security mechanisms and its implementations, it is vital to look in to the underlying data processing technologies of these database systems on which the essential performance and security principles are performed and heavily rely on.

2.1 Database Transaction Models

The idea of transactions and their logical semantics were evolved with the data management techniques. A transaction is bundling of multiple operations on database state into a single set of sequence. When multiple users share the same set of data in a database, handling concurrent transactions have raised issues as it needs to ensure consistency and integrity of data. In late 1970s Jim Gray defined the most widely accepted transaction model and later it became popularized as ACID transactions [5]. ACID transactions offer guarantees of synchronous access to mutable database state. The *atomicity* property guarantees that either all or none of the updates of a transaction are committed. This is significant in replicated databases in order to maintain the consistency. The *consistency* property stipulate that all transactions must follow defined rules and restrictions of the database. The *isolation* property of a DBMS ensures that synchronous execution of transactions results in a system state that could be obtained if transactions were executed serially. Finally, the *durability* property guarantees that the updates (of a transaction) are intact once the transaction is committed.

With the increased level of scalability requirements of web applications, it became apparent that no ACID compliant database could ever satisfy the needs of handling large distributed volume of data. In 2000, Eric Brewer presented a conjecture explaining trade-offs in distributed systems, later popularized as Consistency, Availability, and Partition tolerance (CAP) theorem [6]. The CAP theorem states that it is possible to have at most only two of consistency, availability, and partition tolerance. *Consistency* defines that all replicas of the same data will carry the same value across the distributed system at a given instant. *Availability* means even in an event of failure, the database remains operational with the help of remaining live nodes in the distributed system. In contrast, *partition tolerance* defines that the system is designed to operate in the face of unplanned network outage between replicas. Later, as an alternative design, Basically Available, have a Soft state, Eventually consistent (BASE) model [7] has been proposed which was derived from the CAP theorem in which consistency and isolation in ACID transactions have given lower priority in order to favor the availability and scalability. Thus, ACID and BASE represent the two design considerations at the opposite ends of the consistency-availability spectrum and most of today's cloud based distributed systems use a mix of both approaches [8].

2.2 Data Management Systems

Over the last few decades, Relational Database Management Systems (RDBMS) were identified as the most suitable solution for large-scale storage and management (irrespective of their naturally fit to the relational data model), due to strong guarantees of ACID properties. The Oracle, MySQL, Microsoft SQL Server and PostgreSQL are some of the most popular relational database systems available today. However, with the increasing demand for Big Data systems that are typically composed with variety of data models in structured, semi-structured and unstructured representations, relational databases faced several challenges in terms of storage and performance. At first, they were required to cater the intensive needs of data access on database systems, making them to change the architecture from centralized to distributed in nature. Second, traditional relational databases impose challenges in maintaining the guaranteed performance due to the volume expansion of data in a much fast pace. This vacuum brings the existence of NoSQL (Non SQL) models.

The NoSQL systems usually comes with many added advantages compared to the relational databases including the support for unstructured data models, high concurrency, low latency, high flexibility, high scalability and availability. The term NoSQL was first appeared somewhere in late 1980s to name a relational database that did not have an SQL interface and it was then brought back in 2009 for naming an event introducing non-relational databases [2]. These NoSQL systems provide data partitioning and replication as in-built features and usually run on cluster computers deployed on commodity hardware that can provide horizontal scalability. There are different types of NoSQL data models that are actively being discussed and these can be categorized in to four basic types. 1) Key-Value Store having a big Hash Table of keys and values (e.g., Riak KV, Amazon DynamoDB) 2) Document-Oriented Store that stores documents made up of tagged elements (e.g., MongoDB, CouchDB) 3) Column-Oriented Store where each storage

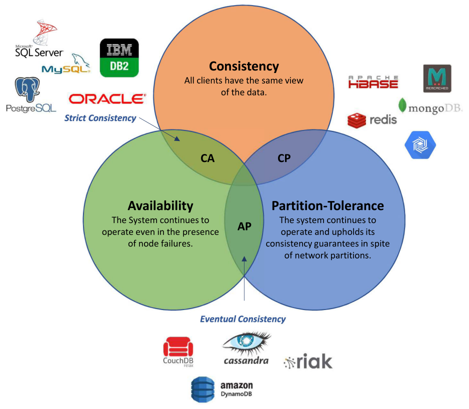
SQL Server
MySQL
IBM DB2
PostgreSQL
ORACLE
Strict Consistency
Consistency
All clients have the same view of the data.
CA
CP
Availability
The System continues to operate even in the presence of node failures.
AP
Partition-Tolerance
The system continues to operate and upholds its consistency guarantees in spite of network partitions.
HBASE
mongoDB
redis
Eventual Consistency
CouchDB
cassandra
riak
amazon DynamoDB

Fig. 1. Database systems according to the CAP theorem.

block contains data from only one column (e.g., Cassandra, HBase) 4) Graph Store which is a network database that uses edges and nodes to represent and store data (e.g., Neo4J, OrientDB).

As the name suggests, *key-value* systems store data as key-value pairs. However, these datastores differ widely in functionality and performance while some systems store data ordered on the key and others do not. Some keep entire data in the memory while others persist data into the disk. The most defining characteristics of key-value databases include real-time processing of Big Data, horizontal scalability across nodes in a cluster, reliability and availability. Hence, they can achieve extremely fast response times even with commodity type processors [9]. *Document-oriented* databases are used to manage semi-structured data typically in the form of key-value pairs as JSON [10] documents. Customarily, each document is an independent entity with varied and/or nested attributes, generally indexed by their primary identifiers as well as semi-structured document field values. Thereby, document datastores are ideal for applications that involve aggregates across document collections. Traditionally, relational database systems are row-oriented systems as their processing is row-centric and are designed to efficiently return rows of data. In contrast, *column-oriented* datastores are column-centric. Conceptually, it can be represented as a relational database having an index on every column, without incurring any additional overhead. Due to the inherent characteristics in the design, these column oriented databases have set of column families (nested key-value systems) and a column family may have any number of columns of any type of data, as long as the latter can be persisted as byte arrays [9]. Moreover, columns in a family are logically related to each other and physically stored together hence, they can be used in applications that are characterized by flexible database schema, sparse data, high speed insert and read operations. *Graph databases* on the other hand are applied in areas where relationship about data interconnectivity is more, or as important as, the data itself [11]. These relationships can be either static (or may be dynamic) nevertheless, introducing graphs as a modeling tool has several advantages for this type of data viz. more natural modeling of data, applying queries directly to the graph (e.g., finding shortest path) and so on. Hence, most of the social network applications are naturally modeled using graphs. Despite all the benefits, these NoSQL databases lose the support for ACID transactions as a trade-off for increased scalability and availability [12]. Hence, larger fraction of NoSQL databases consider BASE as the transaction model which was derived from Brewer's CAP theorem.

The NewSQL on the other hand is a class of modern RDBMS that brings the benefits of performance and scalability of NoSQL while still maintaining the ACID guarantees of relational database systems. Organizations that handle high-profile data which requires strong consistency requirements (such as financial and/or order processing), are unable to admit the direct benefits of NoSQL due to the property of eventual consistency. In order to challenge this barrier, the idea of combining both relational and non-relational database architectures was proposed. NewSQL datastores meet many of the requirements for modern data management in cloud infrastructures, as it brings the best of both relational and non-relational architectures. The term NewSQL was first appeared in 2011 in a research paper discussing the rise of new database systems as challenges to established vendors [13]. Even though, different NewSQL systems vary greatly in their internal architectures, these datastores seem to be one of the promising database technologies in the near future. Most of the NewSQL systems are completely new and are written from the scratch with a distributed architecture in mind [13]. The VoltDB which is the commercial version of research project H-Store [14] and Google Spanner are considered to be the most prominent database systems in this category while Clustrix, NuoDB are also considered as commercial SQL compliant datastores under the roof of NewSQL. However, it is worth to note that no NewSQL systems (currently) are as general purpose as traditional relational SQL database systems set out to be. In addition, most of these systems are in-memory architectures in which may be inappropriate to directly use for volumes exceeding few petabytes [15].

2.3 Data Models and Processing Techniques

In a broader term, a database is simply a collection of data stored in a logically coherent manner so that the retrieval of data is efficient. The model of the database describes the logical structure and typically resolve the functionality of the database. When the workload of database grows, it is necessary to scale out and distribute the workload among multiple servers and this process is termed as horizontal scalability. One of the main disadvantages with relational model is the lack of support for horizontal scalability because when a relational database system is scaled out, it can become overwhelmingly complex. Even though they offer limitless indexing features with strong SQL support while having built-in data integrity, they were unable to share the common Big Data characteristics of Volume, Velocity and Variety (3Vs).

The NoSQL datastores are primarily designed with eventual consistency algorithms in mind hence they do not provide support for ACID transactions. But, these systems have strong performance guarantees that can handle massive volumes of data in terms of Big Data analytics. In addition, it is well understood that one data model does not fit into all requirements of today's data-driven applications. Hence, some of the datastores put availability first (e.g., Cassandra, DynamoDB) and some put flexibility first (e.g., MongoDB, CouchDB) while some of them are focused on alternative data models (e.g., Neo4j). This categorization is depicted in Fig. 1.

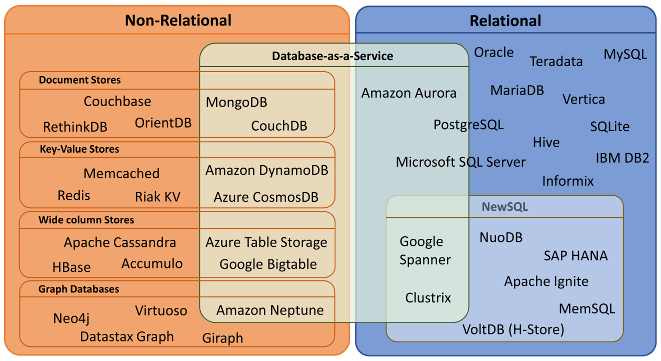

Fig. 2. Summary of database landscape.

2.3.1 Transaction Processing versus Analytical Processing

Historically, database systems were mainly utilized for On-line Transaction Processing (OLTP) where they access and process comparatively only small portions of the entire database and, therefore can be executed quite fast (e.g., sales order processing or banking transactions). However, the lack of support for ACID transactions made many NoSQL datastores not suitable for on-line transaction processing. For an example, financial applications that handle large number of short on-line transactions (Insert/Update/Delete), need strong consistency requirements. In such circumstance, most NoSQL database systems cannot cope with OLTP. Lately, another new usage of database systems has evolved and popularized as Business Intelligence (BI). Applications that support BI integrations rely on long term running On-line Analytical Processing (OLAP) queries (e.g., statistical databases) that process substantial portions of the data to produce analytical reports (e.g., aggregated sales statistics). In oder to process OLAP transactions, NoSQL datastores require lots of application code support. Applications that deals with greater amount of archive data or that have complex queries which use data aggregations, have issues with NoSQL models as they do not have direct support for joins and different levels of indexing.

In practice, most of the organizations with high rate of mission-critical transactions have split their data into two different systems, making one database for OLTP transactions while other (data warehouse) serving for OLAP queries. Despite the capability of decent transaction rates, there are many disadvantages of this separation including data freshness issues due to the delay caused by periodic synchronizations and excessive resource consumptions due to maintaining two separate data processing systems. As real-time data analytics play an increasingly important role in operations, most modern-day organizations are seeking to provide access to data across enterprises, by avoiding data silos to whatever limits possible. Earlier attempts to execute both types of transactions on operational OLTP database such as SAP EIS project [16], were dismissed as OLAP query processing led to resource contentions and severely hurt the mission-critical OLTP queries [17]. In order to fulfill this gap, the NewSQL datastores were primarily designed to utilize the main-memory database architectures. At first glance, the current explosion of data volume seems contradicting with the premise of keeping all transactional data memory resident. However, some studies [17] demonstrated that transactional data volume is limited in size and it favors in-memory data management even for larger commercial enterprises. For this reason, NewSQL database systems received much attention from many of the today's data-driven applications that requires business intelligence. Fig. 2 summarizes the database landscape of modern-day big data applications.

2.3.2 Disk-Based versus In-Memory Systems

Another common classification of data models is categorizing them either as disk-based or or in-memory data processing systems [18]. Most of the traditional relational database systems were developed to work on disk-based architectures where data processing (or larger portion of it) happens on disk. The introduction of Solid State Disks (SSDs) was highly favorable for disk-based database systems as the performance of SSDs were on orders of magnitude superior to magnetic disk devices. Regardless of the data model, none of those disk-based systems were able to support data analytics in real-time, as they need very high transactional processing. With the development of multi-core CPU architectures and availability of large amounts of main memory, created new breakthroughs with faster access making it viable to build in-memory systems where significant part of the database fits into the memory.

In order to take the full advantage of a large memory system, in-memory datastores requires an architecture that is aware that the database is completely memory resident. Traditional database systems almost habitually cache data in main memory to minimize disk IO. But, this is pointless in an in-memory system since database is already resides in memory. Thus, it requires to have cache-less architecture. On the other hand, since whole database is in memory, there should be some alternative persistence mechanisms to ensure that there is no data loss due to power failures. In order to facilitate this, in-memory systems generally use some combination of techniques such as replicating data within the cluster, writing complete images (snapshots/ check points) to disk and writing out transaction records to append-only disk files. MonetDB [19], [20] is one of the most influential database systems in the category of in-memory OLAP datastores. The SAP-SE's TREX [21] is another project under the same category, utilizing a columnar storage. On the other hand, VoltDB and Timesten can be categorized as dedicated OLTP main memory systems. Table 1 summarizes the top ranked (ranking is based on DB Rankings [22]) most popular disk-based and in-memory data processing/management systems that are available today.

The next section discusses the database security risks, threats and vulnerabilities with a discussion on different threat/adversarial models.

3 Database Security Risks, Threats, and Vulnerabilities

Security is an important part of any datastore especially in the cloud paradigm. Despite the different benefits offered by divergent database architectures (either relational or non-relational), ensuring data confidentiality, integrity and availability in any system is one of the important aspects in

TABLE 1
Classification of Disk-Based and In-Memory Database Systems

	Disk-based Systems	In-Memory Systems
Relational	Oracle [23]	Informix [24]
	MySQL [25]	Oracle TimesTen [26]
	SQL Server [27]	
	PostgreSQL [28]	
	DB2 [29]	
NoSQL	DynamoDB [30]	Redis [31]
	Riak KV [32]	Memcached [33]
	Cassandra [34]	MongoDB [35]
	Hbase [36]	Aerospike [37]
	Accumulo [38]	ArangoDB [39]
	Google Bigtable [40]	Hazelcast [41]
	Couchbase [42]	
	CouchDB [36]	
	OrientDB [43]	
	Neo4j [44]	
	Amazon Neptune [45]	
NewSQL	Google Spanner [46]	SAP HANA [47]
	Vertica [48]	VoltDB [49]
		MemSQL [50]
		Apache Ignite [51]
		NuoDB [52]
		Hekaton [53]

database security. Today, data security is of relatively greater concern than expanding capacity and moving to the cloud for enterprise information systems [54]. Moreover, majority of organizations do not store their mission-critical data in the cloud simply because of the security and privacy concerns. Several discussions have been going on [55], [56], [57] related to the latest security mechanisms and evolving trends to protect database systems against potential vulnerabilities/threats. Different types of cryptographic mechanisms, secret-key based methods, digital signatures and certificates are some of the means that are currently available to protect database systems. However, when moving a database system from on-premise to a cloud computing environment where dynamically scalable and virtualized resources are available for use over the Internet, ensuring database security (and privacy) is a challenging task. On the other hand, while it is a challenging task, it is one of the major necessities in today's Big Data applications than ever.

Continuing large scale compromises in database systems that manage sensitive information have influenced the active research on design of new technologies for securing information beyond the typical security mechanisms available in database systems. On the other hand, with the requirements of modern Big Data applications, various protocols have also been proposed for securely outsourcing data to a third party database servers based on strong cryptographic primitives such as fully homomorphic encryption (FHE), oblivious RAM, searchable symmetric encryption, order preserving encryption and so on. However, on the flip side, some of the recent work [57], [58], [59] have demonstrated successful attacks specially on encrypted databases and found that these systems are still vulnerable. Hence we envisage the requirement of having formal understanding of performance and security trade-off in database systems giving emphasis on different attacking strategies.

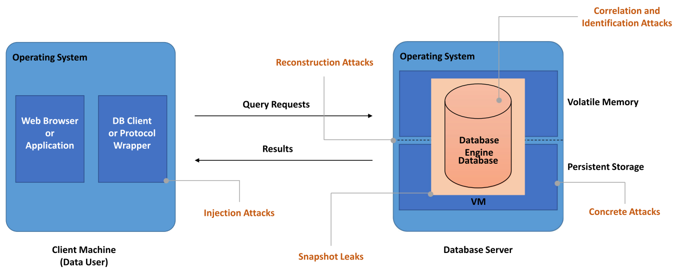

Fig. 3. Typical abstraction of database server deployment.

3.1 Adversarial Models

As suggested in the literature [60], [61] the strongest threat model in database systems is the active attacker who fully compromise the database server (e.g., administrator of a cloud service provider) and perform arbitrary malicious database operations. However, as discussed by Grubbs et al. [57] such attacks are difficult to defend against and instead latest security models focus on passive attacks that do not interfere with the functionality of database but passively observes all its operations (honest-but-curious model). This can include observing the queries issued by the data-user and how these queries access the data in the database. Typical abstraction of database deployment is described in Fig. 3 that is used to explain the threat models and attacks exist in actual database systems in production environment. It is also worth to note here that, most of these theoretical threat models are conceptions. They are not really digging into analyzing the actual material available in an event of database compromise (e.g., database log files, VM snapshot leaks) to look how they infer sensitive information. Hence, in this classification we have also considered the importance of these auxiliary information when discussing attack strategies in database systems.

For most of the attacks that are focused on violating confidentiality of data, adversary is honest-but-curious who has some means of access to the database server or is residing at the server-side sniffing the communication. However, for injection attacks attacker can be at the client-side who is injecting the malicious code to a remote web access request (through an API), when processed by the database client or protocol wrapper. Comparatively, in most of the attacks that are focused on privacy breaches, adversary can be a legitimate data-user who has unrestricted access to the database (e.g., data analyst) [62].

3.2 Attacks on Database Systems

In general, attacks on database systems can be categorized into two main classes. The first category discusses about how confidentiality of data can be compromised while the second category discusses about how data privacy can be revealed.

3.2.1 *Attacks based on Confidentiality of Data*

a) Injection Attacks: SQL injection is one of the typical attacks [63] that works on inserting malicious code into the query statements when application passes them to the database client. Most of the databases store performance statistics as system level diagnostic tables that can be used for database tuneups and to resolve diagnosed issues. These tables sometimes store timestamped list of currently executing queries (e.g., information_schema, performance_schema database in MySQL) where an attacker can easily obtain a list of queries made by other users. Moreover, Ron et al. [63] discussed about the same in a context where NoSQL databases and demonstrated that an attacker can even bypass the authentication mechanism and extract data illegally by injecting a malicious code.

b) Leakage-abuse/Inference and Reconstruction Attacks: This is an attacking strategy where adversary exploits some leakage to recover the query information. In 2016, kellaris et al. developed generic reconstruction attack model [59] which can recover significant fraction of the search keys with a good probability in a polynomial time. In their study, they have categorized the attacking strategy into two main classes based on query access pattern and communication volume. Reconstruction attack using query access pattern refers to the server learning which records are returned as a result of a particular query. In contrast, the reconstruction using communication volume refers to the server learning how many records are returned as a result of a query.

This attack is even possible with encrypted databases (EDB). Roughly speaking, most EDBs rely on some kind of property-preserving encryption (PPE) mechanisms (e.g., deterministic, order-preserving) which enables them to execute various database operations. However, still these solutions are prone to leak some amount of information. This has steered various reconstruction attack models [58], [64], [65], [66] where the attacks are even possible with partial information about a single record in the DB.

c) Concrete Attacks: This refers to the theft of persistent storage (disk theft). ACID compliant databases use on-disk log files in order to facilitate roll-back operations for most recent transactions. By using standard forensic techniques, these log files can be used to reconstruct the past query transactions issued on the database [67]. Furthermore, in [68] Grubbs et al. revealed that the timing of queries carries sensitive information which can be extracted from log files that support replicated transactions. Typically, these attacks can be mitigated/minimized using data-at-rest encryption mechanisms.

d) Snapshot Leaks: Today's database systems are increasingly deployed on Virtual Machines (VM) hence they are exposed to the threat of VM image leakage attacks [69], [70]. In this scenario, attacker obtains an image of the virtual machine and hence reveals the point-in-time state of the entire persistent and/or volatile memory. By accessing individual pages in the cache, attacker can reveal the information about past queries. In [57] Grubbs et al. have performed this attack on MySQL database and revealed the ability to dump the whole memory of the MySQL process.

e) Full System Compromise: This is the attack in which rooting the database system and gain full access to the database and OS states. This can be a persistent passive or an active attack but as mentioned earlier as well, passive attacks are more common.

3.2.2 Attacks based on Privacy of Data

One of the major threats in terms of privacy in database systems is linking different types of datasets together to reveal unique fingerprint of an individual or sensitive information (also known as re-identification). These type of attacks can be categorized into two subclasses and often they are insider attacks.

a) Correlation Attacks: In this class of attack, values in a dataset is linked with other sources of information to create more unique and informative entries. For an example, if one published database lists user information with medication prescriptions and another lists user information with pharmacies visited, once both are linked the correlated database can have information such as which patient bought its medication from which pharmacy [71]. Hence, final correlated dataset can have more information per user.

b) Identification Attacks: In identification attacks, an adversary tries to find out more information about a particular individual by linking entries in a database. This can be considered as the most threatening type of data privacy attack as it has more impact on an individual's privacy. For instance, if an employer searches for occurrences of its employees in a pharmacy customer database, it may reveal some information about medical treatments and illnesses of its employees.

In terms of mitigating these attacks, data anonymization or data pseudonymization techniques can play a big role in a way such that linkage of datasets are still feasible, but identifying an individual from that dataset becomes hard. Following section provides a comprehensive assessment of security mechanisms in leading database systems with a discussion on how to mitigate these threats.

4 Data Protection Mechanisms in Database Systems

In terms of mitigating the risks, database systems are equipped with different types of security mechanisms. There are sufficient number of surveys [2], [4], [56], [72] carried out in the past to compare the security implementations in RDBMSs and NoSQL datastores. Compared to these RDBMS models, database security is overlooked by many of the NoSQL and NewSQL datastores. As more attention has given for the performance of the database engine, some of these systems even do not facilitate at least sufficient authentication mechanisms (e.g., Redis). On the other hand, distributing data over multiple servers in different data centers provides more avenues for security breaches. This section reviews existing security mechanisms giving emphasis on industry standard security and privacy best practices and concepts [73] along with current efforts of the database and cryptographic communities to extend these existing mechanisms (Fig. 4 summarizes the classification of these database security mechanisms).

4.1 Authentication, Authorization, and Access Control Mechanisms

Authentication is the mechanism that identify (and verify) the users associated with a database system, before allowing them to access data/resources. This can be provided in different ways ranging from single user authentication to mutual authentication of user with database server [74]. A typical implementation is password-based authentication model allied with a user login. Some database systems have its own integrated authentication mechanisms while rest of the systems employ some other mechanisms such as user certificates and integrated directory services, where database

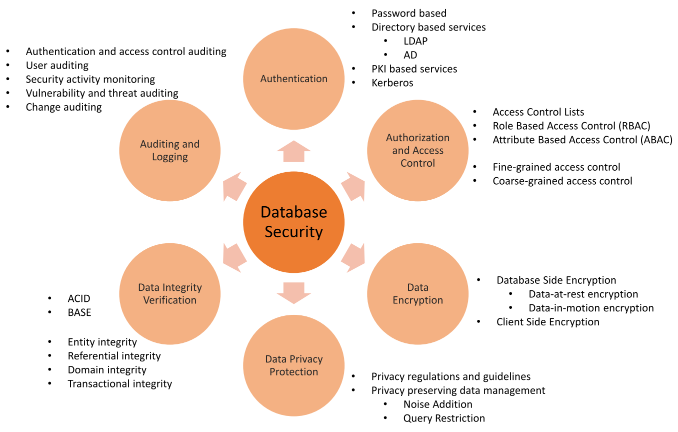

Fig. 4. Classification of database security mechanisms.

users (and user roles) are authenticated through an organizational level directory service like Lightweight Directory Access Protocol (LDAP) or Active Directory and Kerberos servers. Multi-factor authentication and certificate based authentication are some of other well known authentication techniques while Secure Sockets Layer (SSL) and Kerberos are widely used authentication protocols.

Authorization plays a major role in any database system in security perspective. Once the identity of the user has been verified, it is then required to map/grant the user to the resources within the database system. Authorization is the mechanism through which it can be ensured that only authorized database users/roles are allowed to access defined set of objects or the entire database. It is usually performed through controlling a set of policies/permissions associated with each user. Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role Based Access Control (RBAC) are three of conventional access control models [55]. Beside these models, Policy Based Access Control (PBAC) and Attribute Based Access Control (ABAC) have gained more attention during the recent past which can satisfy various other security requirements within an application domain. Relational database systems usually have RBAC mechanisms hence they implement authorization at the table level while most of the NoSQL database systems allow column-family level authorization.

Today, most of the relational database systems are equipped with some means of authentication mechanisms. As an example, Oracle database [23] has a powerful set of authentication mechanisms including means to authenticate through network using protocols like Kerberos, PKI-based services or directory-based services. As of the case of NoSQL datastores, not every NoSQL datastore comes with authentication mechanisms and some of them are not strong enough. For an example, in Redis [31] admin password is sent in clear-text for admin functions and data access does not support authentication [4]. However, Cassandra, MongoDB, HBase are some of the NoSQL datastores that provide comparatively stronger authentication mechanisms.

In terms of authorization, most RDBMSs customarily equipped with role-based access control mechanisms. These systems usually allow authorization at the table level, however systems like PostgreSQL [28] even allow per user based row-level security (RLS), broadly termed as fine-grained authorization/access control. In PostgreSQL, by default, database tables do not have any policies. Therefore, if a user has some level of access privilege to a particular table, all rows within that table are equally available for querying or updating. But with RLS, row level security can be defined so that only specified rows will be available for querying and updating. These fine-grained access control mechanisms allow database administrators to define object level security within the datastore and in terms of relational database systems, these can be further classified into row level or cell level. In the matter of NoSQL database systems, there is no schema associated with; hence they store heterogeneous data together. Thus, most of them cannot provide authorization at the table or object level instead, they allow mechanisms such as column-family level authorization. However, wide-column stores like Apache Accumulo [38] provide cell-based access control mechanism using an access control list (ACL). Apart from that, almost all the NewSQL datastores that have been surveyed, are also enabled with fine-grained role-based access control mechanisms. This can be most probably attributed to the availability of relational models in the NewSQL datastores. On the other hand, systems such as Apache Ignite [51] do not provide any sort of authorization features. As the case of most NewSQL systems, Ignite also utilizes main memory as the default storage and processing tier, hence they might have not invested more on authorization since the system completely runs on memory.

It is noteworthy that some of the non-relational datastores completely operate on cloud as services hence they inherently absorb the identity and access management (IAM) systems implemented at the cloud infrastructure level. Amazon DynamoDB [30], one of the leading key-value stores and Amazon Neptune [45] a NoSQL graph database, both utilize the identity and access management services provided by Amazon Web Services. Moreover, systems such as Google Cloud Bigtable [40]-a wide-column store-and Google Spanner [46]-a distributed NewSQL database-both utilize the inherent features of Google Cloud identity and access management services to implement the database authentication and authorization. Moreover, it is also noteworthy that distributed (sharded) database systems need to have additional layer of properly managed access control policies to maintain consistent authorization throughout the cluster in order to ensure unrestricted access to legitimate/authorized users [75].

4.2 Encryption Mechanisms

Encryption is the mechanism which ensure the confidentiality of data in a database system such that malicious intruders and unauthorized parties cannot access any valuable information. In order to secure data by means of encryption, it is required to protect them not only when they are at rest but also in transit or in motion. Data at rest refers to the data that has been flushed from the memory and written to the disk. Data Encryption Standard (DES) and Advance Encryption Standard (AES) are two of widely used algorithms for data-at-rest encryption. Data in transit (motion) usually refers to the data that is in communication or is being exchanged in a communication. With the existence of modern distributed architectures, data in transit (communication) can be further classified in to two categories;

a) Client to server communication: Almost all datastores allow remote connections to database so that clients can remotely connect to the database to retrieve or process data. However, this connection needs to be secure and private hence channel must be encrypted.

b) Inter-node communication: Some of the relational database systems and most of the NoSQL and NewSQL datastores are equipped with distributed processing mechanisms and also equipped with different integrated replication strategies where nodes in a database cluster needs to communicate between each other in order to synchronize data. This communication can also be eavesdropped, hence, needs to have a server-server encryption mechanism.

4.2.1 Industry Established Solutions

Most relational database systems available today are equipped with the mechanisms to protect both data-at-rest and data-in-transit. Some of these encryption technologies are more specific for a given database system and some of them are mostly applied by many vendors. Transparent Data Encryption (TDE) is one of such technology employed by many vendors such as Microsoft, IBM and Oracle to provide protection for data-at-rest. Oracle database and Microsoft SQL Server are some of the popular relational database systems that use TDE as primary data encryption mechanism in which they basically implement protection at file level, by encrypting database both on the hard drive and backup media. The encryption key used by these technologies can be either a symmetric key which is secured using a certificate stored in the master database or an asymmetric key provided by a key management service. Apart from that, in most of the cases TDE employ either AES [76] or 3DES [77] encryption algorithm in order to encrypt data. However, many NoSQL solutions such as Riak [32], Redis [31], Memcached [33] and CouchDB [36] are initially designed to be worked on secure and trusted environments, hence they do not provide any sort of encryption mechanisms. Nevertheless, NoSQL datastores such as Cassandra and HBase now facilitate TDE (with their enterprise version) to provide encryption for data-at-rest. While most of the database solutions deliver inbuilt mechanisms to encrypt data, some systems such as Accumulo and Neo4j (even though they do not have integrated encryption mechanisms) provide the necessary features to integrate them with third party on-disk encryption tools to ensure security for data-at-rest. On the other hand, as they are still in the evolving stage, NewSQL solutions such as Apache Ignite, VoltDB and NuoDB do not provide any mechanisms to protect data-at-rest other than relying on third party tools.

As the protection for data-at-rest is implemented at the database engine, it is also equally important to ensure the protection when data being exchanged or in communication between database server and client applications or other nodes within the same cluster. Traditionally, most of the database systems employed firewall policies, operating system level configurations or organizational level virtual private networks (VPN) to ensure security of these inter-node communications as most of the time they have been deployed in on-premise trusted environments. However, when datastores become more and more distributed and their deployment architecture changes from on-premise to cloud infrastructures, special mechanisms are required to ensure protection for data-in-transit. Most database systems including NoSQL and NewSQL, now supports encryption for data-in-transit by using Transport Layer Security (TLS) [78].

Apart from the network level encryption mechanisms there are some other set of technologies where data is encrypted at the client side transparently by the data connection layer so that data then remains encrypted over the network, in memory and on the drive. With SQL Server 2016 (Azure SQL Database), Microsoft introduced a technology called Always Encrypted [79], which belongs to this category of protection where both encryption for data-at-rest and data-in-transit can be ensured. Hence, it provides a clear-cut separation between those who own the data and those who manage the data, especially with cloud based services. Further, it ensures that on-premise database administrators, cloud database operators, or other high-privileged but unauthorized users cannot access the sensitive information hence minimize the risk of concrete attacks.

4.2.2 Solutions Provided by Cryptographic Community

Several studies have also been carried out to protect database from curious insiders and malicious outsiders by encrypting the content at the client side using different approaches. In 2011, Popa et al. presented CryptDB [60] an encrypted query processing mechanism that works on relational database systems. The main idea was, client encrypt the original data at a middle-ware application at client-side in a trusted vicinity and store them in the database located in an untrusted environment in such a way that it can query over the encrypted data. Their design was bundled with layered architecture of encryption schemes, which enables execution of SQL equality checks, order comparisons, aggregates and joins. This idea has given the momentum for research on security-aware database systems and CryptDB ensures that in an event of database server get compromised (full system compromise), most of the data is secured. Later, multiple CryptDB based frameworks [80], [81], [82] were able to serve in different dimensions making them well-suited for outsourced production databases with third party service providers.

Different approaches have also been proposed to implement secure encrypted NoSQL datastores. BigSecret [83] is a framework that enables secure outsourcing and processing of encrypted data over key-value stores where indexes are encoded in a way that allow comparisons and range queries. In another quite different approach, Yuan et al. [84] proposed an encrypted, distributed, and searchable key-value store with a secure data partition algorithm that distributes encrypted data evenly across a cluster of nodes. In Secure-NoSQL, Ahmadian et al. [85] looked in to the aspects of ensuring both confidentiality and integrity of data on a document store NoSQL data model. Also, Macedo et al. [86] presented a generic NoSQL framework and set of libraries supporting data processing and cryptographic techniques that can be used with existing NoSQL engines.

It is noteworthy that good fraction of above solutions are rely on PPE based schemes such as [87], [88] which makes them vulnerable for various inference attacks. Hence, there is another line of work focusing on secure hardware along with trusted execution environments such as *enclaves* (e.g., Intel SGX [89]) to enable secure query processing. The

Cipherbase [90] and TrustedDB [91] are some of the works that can harness the power of enclaves by placing part of the db engine inside some allocation of trusted hardware. Recently, Priebe et al. [92] has proposed EnclaveDB that guarantees confidentiality and integrity of data by hosting all sensitive data in an enclave memory.

4.3 Ensuring Data Integrity

Data integrity is a fundamental concept which enables the protection for data from unauthorized modification (unintentionally or maliciously). It refers to the accuracy and consistency of data stored in a database system and verifies that the data has remained unaltered in transit from creation to the reception of data. Consistency model of a database system defines how well that datastore can ensure data integrity. Database systems with strong ACID guarantees can ensure higher level of data integrity compared to other consistency models such as BASE. In practice, data integrity can be enforced in a database system by series of integrity constraints or rules. In relational database systems integrity constraints are an inherent part of the system and they can be generally classified into three types as 1) entity integrity, 2) referential integrity and 3) domain integrity. Entity integrity is basically the concept of primary key where every value in a particular column (or combination of columns) can be identified using a unique (and not null) value. Comparatively, referential integrity means the concept of foreign key. This helps to define the relationship between tables. Domain integrity concerns the validity of entries for a specific data column by ensuring the appropriate data type, format etc. Maintaining data integrity in a database system means making sure that data remains intact and unchanged throughout the entire life cycle.

Whenever data is processed at the database, there is a risk of data cloud get corrupted/changed either accidentally or maliciously. With all the different types of integrity constraints, relational database systems can minimize the chances for accidental data corruption. However, due to the heterogeneous nature and schema-less architecture of NoSQL datastores and as larger fraction of their consistency model is "eventual consistent", most of the NoSQL datastores are unable to facilitate data integrity. On the other hand, it is also hard for them to ensure referential integrity and transactional integrity because of their design constraints. But, there are some NoSQL databases that are capable of providing data integrity. Document datastores like MongoDB and graph datastores such as Neo4j, Virtuoso and Amazon Neptune now provide the strong support for ACID guarantees making them compatible for data integrity validations. Nevertheless, as NewSQL datastores primarily designed to ensure strong ACID guarantees, most of today's NewSQL database systems are able to provide data integrity.

While, combining different integrity constraints in a database system can minimize the risk of accidental data corruption or update, it is hard for these constraints itself to ensure all the requirements that satisfy data integrity. Therefore, in practice, organizations usually enforce other mechanisms such as implementing regular data-backup policies, ensuring proper functioning of IT network and well-defined security policies etc. in order to safeguard data integrity. On the other hand, especially to provide protection from malicious activities on the database, it is equally important to have mechanisms not only within DBMS but also beyond DBMS level such as network layer. In such circumstance, most database systems employ transport layer security protocols (TLS/SSL) to facilitate and ensure data integrity beyond DBMS level.

4.4 Inference Control Mechanisms and Maintaining Privacy of Sensitive Information

Maintaining data privacy is one of the key challenging tasks with any database system. However, it is even more challenging with cloud-based distributed architectures as when database systems are hosted in a public cloud, curious cloud operators might have the access to private data. Hence, it is required to implement proper and fine-grained mechanisms to protect data privacy in database systems. In a broader sense, data privacy (or information privacy) is the necessity to preserve and protect personal (or sensitive) information from being accessed/disseminated by a third party [93]. As per Agrawal et al. [94] privacy can be identified as the right of individuals to determine for themselves when, how and up to what extent information about them is communicated to others. Typically, privacy preserving data protection mechanisms (such as encryption, authentication and information masking) determine what data within a database system can be shared with others and which should be restricted. Privacy can be in the form of different types of data; 1) on-line privacy which contains personal data shared during on-line transactions 2) Financial privacy that contains any financial information 3) Medical privacy which contains privileged medical information such as medical treatments 4) Location privacy that shares location-based data and 5) Political privacy which contains political preferences.

However, as most modern database management systems do not consider privacy as a key feature, it is not an explicit characteristic of the underlying data model upon which these systems are built. On the other hand, due to the volume expansion of data in a fast pace, it is quite difficult for a general purpose data management system to provide real-time filtering mechanism to define what data is sensitive and what is not. With the introduction of encryption and access control mechanisms, database designers were able to ensure some level of data privacy. However, it is well understood that these mechanisms itself does not guarantee the security and privacy for outsourced database systems [57], especially when the systems are deployed on public cloud infrastructures. Even though none of the database systems available today are capable enough to provide complete, separate or integrated mechanisms to safeguard data privacy, some work has been carried out by the database research community towards developing privacy preserving data management techniques, as discussed next.

4.4.1 Privacy-Preserving Data Management Techniques

Broadly, there are three classes of techniques dealing with privacy preserving data management [95]. First class is dealing with the techniques when data to be released to third parties. These techniques have nothing much to do with database systems as once data are released, database systems do not have any control over it. They usually incorporate data sanitization with the use of data anonymization techniques such as k-anonymity [96]. The second class of techniques are related to the context of data mining in database systems. Even though a database is sanitized by removing private data, strong data mining techniques may allow some features to recover the original information from the database. As a solution, several different approaches have been proposed to achieve privacy preserving data mining by modifying or perturbing data so that it is no longer represent the original information [97]. However, one of the major problems with these techniques is the quality of the resulting database. When data undergo too many modifications, the resultant database may not be much useful. Several techniques have also been developed to address this problem by estimating the errors introduced by the modifications [98]. Moreover, in a context where privacy preserving distributed data mining, several techniques have been proposed based on encryption methods where multiple data owners can work together without releasing original data [97]. The third and final class of privacy preserving data management techniques is dealing with the DBMSs specifically tailored to support privacy policies and standards like W3Cs Platform for Privacy Preferences Project (P3P) [99] initiative. In [94], authors have introduced the concept of Hippocratic databases, basically a privacy protection mechanism for relational database systems. However, implementing such system poses several challenges, even though articulating a privacy preserving DBMS is quite straightforward. Moreover, it is worth to note that in order to implement a production ready privacy preserving database solution, it might require to have a combined approach of data anonymization along with privacy preserving data mining.

4.4.2 Prerequisites for Implementing Privacy-Preserving Database Systems

As suggested by Bertino et al. [100], in a context where tailor-made privacy preserving DBMS solutions, it is crucial that once data being collected, privacy promises be enforced by the information systems managing them. In their study, they have discussed set of requirements towards developing privacy preserving DBMS solution that can be utilized to support wide range of privacy policies. Following key points highlight the most important requirements for a privacy preserving database solution.

a) Support for Rich Privacy Related Meta-data: In mechanisms such as P3P often requires the data users to specify the intended purpose of the data retrieved by them in order to ensure privacy guarantees. Thus, to facilitate access to such meta-data, privacy preserving DBMSs should implement the mechanisms to store privacy specific meta-data in the database together with the data. Further, it should be associated with the data according to a range of possible granularities with the adequate flexibility and without degrading the overall performance of the datastore.

b) Support for Attribute-based Access Control: Most database systems usually equipped with role-based access control mechanisms. However, RBAC does not provide the possibility of specifying application dependent user profiles for use in privacy enforcement. Hence, there should be mechanisms to extend the support for attribute-based or purpose-based access control mechanisms in privacy preserving DBMSs.

c) Fine-grained Access Control to Data: In order to implement a comprehensive privacy preserving DBMS solution, a fine-grained access control mechanism is of utmost importance. In conventional relational databases, only way to have some level of fine granularity in access control is with use of *Views*. However, in order to implement a privacy enhanced DBMS solution, these *View* mechanisms should be extended to the level of each tuple or set of tuples that are being protected and these should be implemented per user basis.

d) Privacy-preserving Information Flow: In most distributed database systems, information/data flow across different domains. Thus, it is important that all privacy policies associated with these data also traverse along with the data when they move within organization or across different organizations. The main idea is to assure that if data have been collected under a given privacy promise of an individual, this should also be enforced when data are passed to different parties.

e) Protection from Insider Attacks: The misuse of privileges by the legitimate high privileged users, is one of another privacy breach exists in database systems that has not received much attention. This can be mitigated by implementing per-user based layered encryption mechanisms or adoption of user access profiling techniques.

4.4.3 Inference Control in Statistical Databases

In a different context yet related to the same, there is another line of work discussing about privacy preserving data management in statistical databases. Typically, Statistical Database (SDB) system enables its users to retrieve aggregate statistics (e.g., count, sum, sample mean etc.) for a subset of entities presented in the database [101]. In today's data driven applications, data analytics (with OLAP) plays a vital role in terms of statistical information extraction for decision making purposes. Current approaches for data security cannot guarantee privacy of individuals when providing general purpose access (for internal users) especially for OLAP queries in a database system. Common mechanisms like access control policies can limit the access to a particular database, but once an inside analyst has access to data, these policies cannot really control how data is used. As demonstrated by many insider attacks [102], [103], [104] allowing unrestricted access to data is one of the major causes of privacy breaches. Therefore, providing security on statistical databases has already become a growing public concern. Over the time, several techniques have been proposed by the research community for preventing statistical database compromise and those can be mainly categorized into two classes.

a) Noise Addition: In this method, all data in the datastore are available for the use but only approximate values will be returned rather than exact. The primary focus in noise addition techniques is to mask the true values of the sensitive data by adding some level of noise/error to it. This is usually done in a controlled way so as to balance the competing needs of privacy and information loss [105]. Based on the how noise is added, these techniques can be further classified (Figs. 5a and 5b).

- *Data Perturbation:* In this approach the original content in the database is replaced by a perturbed database where the statistical queries are performed.

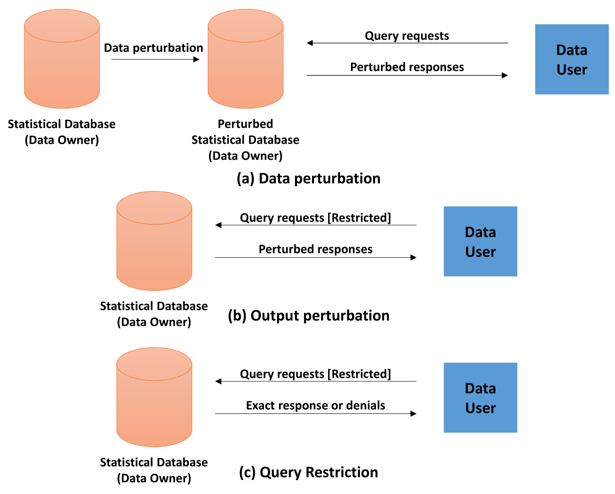
Data perturbation
Query requests
Perturbed responses
Data User
Statistical Database (Data Owner)
Perturbed Statistical Database (Data Owner)
(a) Data perturbation
Query requests [Restricted]
Perturbed responses
Data User
Statistical Database (Data Owner)
(b) Output perturbation
Query requests [Restricted]
Exact response or denials
Data User
Statistical Database (Data Owner)
(c) Query Restriction

Fig. 5. Techniques used in statistical databases to protect privacy.

- *Output Perturbation:* Queries are evaluated on the original data and the noise is added to the results of the queries.

b) Data Restriction: Techniques that restrict data statistics can be broadly divided into three classes namely Global Recording, Suppression and Query Restriction [105]. Global recording transforms an attribute into another domain (e.g., defines a set of ranges for numerical values and then replace each single value with its corresponding range). Suppression is the technique that replaces the value of an attribute in one or more records by a missing value. Finally in query restriction technique, users are not provided with micro data directly, instead they can ask queries through a channel. These queries are either answered exactly or are rejected. The decision of which queries to answer is made by using different techniques/parameters such as query set size, query set overlap so on [101].

In general, noise addition perturbation methods work by multiplying or adding a stochastic/randomized number to confidential quantitative attributes in a database. Typically, this stochastic value is chosen from a normal distribution with zero mean and a very small standard deviation. Additive noise methods were first introduced in late 1980s by Kim et al. [106] and this idea was brought back with improvements [107] and later multiplicative noise approach and its variants were proposed [108]. In 2005 Dwork et al. introduced Differential Privacy (DP) [109], [110] that utilizes Laplace noise addition, yet the most promising technique with strong formal guarantee of privacy. This method enforces confidentiality by returning perturbed aggregated query results from databases such that users of the database cannot distinguish if particular data item has been altered or not. Because of its desirable privacy guarantees, DP has received growing attention from the research community and various mechanisms have been proposed over the couple of years towards implementing DP for SQL queries [62], [111], [112], [113], [114]. Following Fig. 5 shows a summary of techniques used in statistical databases to maintain privacy of data.

By exploring these different privacy aspects, it is evident that protecting private data in a database system is an important concern. But, ensuring data privacy with such set of complex issues, is still a considerable challenging task. Therefore, in order to cope with with today's data driven applications, these data management systems should have comprehensive mechanisms to protect the privacy of data in terms of unauthorized data access, sharing of data, misuse and reproduction of individual information.

4.5 Auditing and Monitoring Mechanisms in Database Systems

Generally, database auditing and monitoring refers to the recording of individual and collective actions performed by database users or system events [115]. It is usually associated with generating (automated) audit trails that logs series of events occurred in a database system such as which database object or data record was touched by which user/account. These event logs are much important in an event of forensic analysis of security events. While all other different security mechanisms are trying to mitigate the occurrence of malicious attacks, in a case of security breach, these audit trails can be used to identify the root cause of the incident. Hence, most of the information security and privacy standards such as Health Insurance Portability and Accountability Act of 1996 (HIPAA), Payment Card Industry Data Security Standard (PCI-DSS), Family Educational Rights and Privacy Act (FERPA) and European Union Data Protection Directive, require the existence of these audit trails in datastores that goes in production environment. In practice, database auditing and monitoring can be classified into several different categories [115].

a) Authentication and Access Control Auditing: Process of identifying the information of who accessed which systems and what components, including when and how.

b) Subject/user Auditing: Process of identifying what activities (e.g., insert, update, delete etc.) have been performed by the users/administrators of the database system.

c) Security Activity Monitoring: Process of identifying and flagging any suspicious, abnormal or unusual activity/access to sensitive data.

d) Vulnerability and Threat Auditing: Process of identifying the vulnerabilities in the database and monitor for users attempting to exploit them.

e) Change Auditing: Implementing baseline policy for different database objects, configurations, schemas, users and privileges and then track deviations from that baseline.

In order to facilitate above list of different security audits, database systems usually maintain several types of logs. Implementation of these logging and monitoring mechanisms varies from system to system. Some cloud-based, service oriented database systems like Amazon DynamoDB, Azure Cosmos DB and Google Bigtable take the advantage of cloud infrastructure level diagnostic and logging tools in order to implement the database logging mechanisms while most of the other database systems usually have integrated logging mechanisms. In some systems like Apache Ignite, even though they do not have integrated logging mechanisms, those can be configured with third party logging libraries and frameworks such as Log4j [116] and SLF4J [117] to enable auditing and logging.

4.6 Are Today's Database Systems Ready to Take the Challenge?

By considering diverse security characteristics available in today's popular database systems, a summary of findings are listed on Table 2. It is worthy of note here that though

TABLE 2
Summary of Comparison of Database Systems Based on Security and Privacy

No	Database & Vendor/Developer	Authentication	Authorization and Access Control	Encryption			Consistency Model	Auditing and Logging
				Data-at-Rest	Data-in-Transit			
					Client-Server communication	Internode communication		
	Relational Database Systems (RDBMS)							
1	Oracle *by Oracle Corporation*	Yes. Implemented through OS or network-based authentication either using SSL, Kerberos, PKI or directory services.	Yes. Different system and object level privileges. User roles and profiles with fine-grained access control.	Yes. Using Transparent Data Encryption (TDE) released with version 12c.	Yes. Using AES and 3DES with Diffie-Helman key negotiation algorithm.		ACID	Yes. Different types of auditing mechanisms available.
2	MySQL *by MySQL AB and lately acquired by Oracle Corporation*	Yes. Several authentication methods.	Yes. Role based access control.	Yes, with enterprise edition.	Yes, using SSL.	Yes, with MySQL Cluster CGE version.	ACID	Yes.
3	Microsoft SQL Server *by Microsoft Corporation*	Yes. Through OS or mixed authentication mode.	Yes. Role based permissions.	Yes. Using Transparent Data Encryption (TDE).	Yes, using SSL.	Yes, using SSL.	ACID	Yes. Several levels of auditing are available.
				Always Encrypted technology is available with SQL Server 2016 where data is encrypted at the client transparently by the data connection layer without any code changes and data then remains encrypted over the network, in memory, and on the drive.				
4	PostgreSQL *by PostgreSQL Global Development Group*	Yes. Different types of authentication methods including GSSAPI, SSPI and LDAP authentication.	Yes, Role based permissions. Introduced per user Row-Level Security (RLS) with version 9.5 and above.	Yes, using encryption with 128bit AES in XTS mode.	Yes, with the client-side encryption feature.	Yes, using SSL.	ACID	Yes, different ways to generate comprehensive audit trails of different database operations with PGAudit tool.
5	DB2 *by IBM Corporation*	Yes, implemented through either OS or a domain controller or a Kerberos security system.	Yes, Role based permissions implemented with different privileges and authority levels.	Yes, with DB2 Native Encryption and IBM InfoSphere Guardium Data Encryption tools.	Yes, using SSL.		ACID	Yes, different categories of audits can be performed through db2audit tool.
	Key-value Stores							

TABLE 2
Continued

6	Redis *by Salvatore Sanfilippo*	Simple authentication. Passwords are stored in cleartext inside redis.conf	No.	No.			BASE	No.
7	Amazon Dynamo DB *by Amazon*	Yes, though Identity and Access Management (IAM) service offered by AWS.	Yes, using permissions policies.	Yes, 256bit AES managed with AWS Key Management Service.	No. Client needs to encrypt them at client side or to use encrypted connections. Or to use DynamoDB Encryption Client.	Yes, using SSL.	BASE	Can be implemented with AWS CloudTrail.
8	Memcached *by Danga Interactive*	Yes, implemented using Simple Authentication and Security Layer (SASL) which is available with version 1.4.3 and above.	No.	N/A (it does not store data on disk)	No.	No.	BASE	No.
9	Azure Cosmos DB *by Microsoft Corporation*	Yes, using master keys and resource tokens.	Yes, using Azure IAM with role-based access control and integration with Active Directory.	Yes. Keys managed by Cosmos DB Management Service.	Yes, using SSL/TLS 1.2		BASE	Yes, with Azure Diagnostic Logs.
10	Riak KV *by Basho Technologies*	Yes, using password, or Pluggable Authentication Module (PAM), or using client certificate.	Yes, using different permissions.	No.	Yes, using HTTPS or encrypted Protocol Buffers traffic.	Yes, using SSL.	BASE	No.
Wide Column Stores								
11	Apache Cassandra *by Avinash Lakshman & Prashant Malik and lately became a project of Apache Software Foundation*	Yes, using password-based authentication, or external mechanism such as Kerberos or LDAP.	Yes, using role-based permissions.	Yes, using Transparent Data Encryption (TDE), however this supports in Datastax Enterprise version only.	Yes, using SSL.		BASE	Available with Datastax Enterprise version.
12	HBase *by Apache Software Foundation*	Yes, using Simple Authentication and Security Layer (SASL) which supports Kerberos.	Yes, using role-based and attribute-based access control using groups and Access Control Lists (ACL).	Yes, using transparent encryption with built in extensible cryptographic codec and key management framework.	Yes, using TLS/SSL.	Yes, using SASL for inter component replication.	BASE	No. However, it can be enabled with other tools with in the Hadoop eco system.

TABLE 2
Continued

13	Accumulo *by Apache Software Foundation*	Yes, using Simple Authentication and Security Layer (SASL) and GSSAPI to support Kerberos.	Yes, with cell-based access control using ACLs.	No. However, can be combined with HDFS Transparent Security to provide encryption at rest.	Yes, using SSL.	No.	BASE	Yes, fixed event logging.
14	Azure Table Storage *by Microsoft Corporation*	Yes, using identity-based authentication with resource tokens.	Yes, using role-based access control with Azure Active Directory (AD).	Yes, using Azure Disk Encryption.	Yes, using SSL.		BASE	Yes, with Azure Activity Logs and Diagnostics Logs.
15	Google Cloud Bigtable *by Google Inc.*	Yes, using Google Sign-in, or Firebase authentication with tokens, or OpenID connect with tokens.	Yes, using Google Cloud Identity and Access Management (IAM). Fine-grained access control with role-based permissions.	Yes, using AES and different key management options like Customer-managed encryption keys (CMEK) or Customer-supplied encryption keys (CSEK)	Yes, using TLS/SSL.	Yes, using TLS.	BASE	Audit logs available only for admin activity and planned for changes in future.
Document Stores								
16	MongoDB *by MongoDB Inc.*	Yes, supports multiple authentication mechanisms including Salted Challenge Response Authentication Mechanism (SCRAM) and certificate based (x.509) authentication. MongoDB Enterprise supports LDAP and Kerberos.	Yes, role-based access control along with permissions.	Available with MongoDB Enterprise version only and it uses AES-256 CBC or AES-256 GCM mode.	Yes, using TLS/SSL with minimum of 128-bit key length for all connections.		ACID	Auditing is available only for MongoDB Enterprise and it has the features to audit and log DDLs, replica sets and shared cluster, CRUD operations along with authentication and authorization.
17	Couchbase *by Couchbase Inc.*	Yes, using different mechanisms including certificate-based authentication (x.509), password-based authentication, LDAP based and PAM-based authentication.	Yes, using role-based access control.	No. Can be implemented with 3rd party on-disk encryption software-vendors.	Yes, using TLS/SSL.		BASE	Yes, system-management tasks can be audited using integrated audit tools.
18	CouchDB *by Apache Software Foundation*	Yes, using either basic HTTP authentication, or cookie authentication, or proxy authentication, or OAuth authentication.	Yes, can be configured in to role-based access control. (However, CouchDB handles permissions on per database basis)	No.	Yes, using SSL.	Can be implemented using HTTPS connections.	BASE	Yes, logging is available for some of the system events but not for all database operations such as CRUD.

TABLE 2
Continued

19	OrientDB *by OrientDB Ltd.*	Yes, using password-based authentication, or Kerberos authentication, or using LDAP. OrientDB Enterprise edition supports Symmetric Key authentication.	Yes, using role-based permissions.	Yes, using AES and DES algorithms. However, encryption at rest is not supported on remote protocol yet.	Yes, using SSL.	BASE	Available only with Enterprise Edition.
20	RethinkDB *by Rethinkdb*	Yes, using password-based authentication, or OAuth authentication.	Yes, using set of permissions implemented as read, write, connect and config.	No.	Yes, using TLS.	BASE	No.
Graph Datastores							
21	Neo4j *by Neo Technology*	Yes, using password-based, or LDAP based authentication, or Kerberos authentication and single sign-on.	Yes, using role-based access control and permissions.	No. Can be used with 3rd party volume encryption mechanisms.	Yes, using TLS/SSL.	ACID	Yes, enabled with two types of logging for inspection of queries and security events.
22	Giraph *by Apache Software Foundation*	Yes, using Simple Authentication and Security Layer (SASL).	No.	No.	No.	BASE	No.
23	Datastax Enterprise Graph *by DataStax Inc.*	Yes, using password-based authentication, or LDAP based authentication, or Kerberos based authentication.	Yes, using role-based access control with permissions.	Yes, using Transparent Data Encryption with AES, DES, Blowfish and RC2 algorithms.	Yes, using SSL.	BASE	Yes, with different options to log system events and database operations.
24	Virtuoso *by OpenLink Software*	Yes, using either basic HTTP authentication, PKI, OpenID or OAuth authentication mechanisms.	Yes, using role-based security and access control list that governs the permissions.	No.	Yes, using SSL.	ACID	Yes, using check point audit trail for database transactions.
25	Amazon Neptune *by Amazon*	Yes, using Identity and Access Management (IAM) service offered by AWS.	Yes, using IAM permission policies.	Yes, using AES-256 encryption with AWS Kay Management Service (AWS KMS)	Yes, using TLS.	ACID	Yes, collection of audit functions for management operations.

TABLE 2
Continued

NewSQL Databases							
26	SAP Hana *by SAP SE*	Yes, using basic password-based, or Kerberos based authentication, or using SAML, or x.509 certificate-based authentication including single-sign on.	Yes, using object-based authorization mechanism with different privilege levels.	Yes, using AES-256 in CBC mode algorithm to encrypt data.	Yes, using TLS/SSL.	ACID	Yes, multiple levels of management, security and database event logs.
27	Apache Ignite *by Apache Software Foundation*	Yes, using basic password-based authentication.	No.	No.	Yes, using SSL and TLS.	ACID	No. Can be configured with 3rd party logging libraries to enable auditing and logging.
28	MemSQL *by MemSQL Inc.*	Yes, using basic password-based authentication, or Kerberos authentication, or SAML authentication.	Yes, using fine-grained role-based access control using policies.	Yes, using Linux Unified Key Setup (LUKS).	Yes, using SSL.	ACID	Available only for MemSQL Advanced Security Option license holders.
29	VoltDB *by VoltDB Inc.*	Yes, can be enabled with configuration file. It can also be configured to work with Kerberos.	Yes, using procedure-based access control.	No.	No, however it can be configured using secure tunnel.	ACID	Yes, supports logging of management events.
30	Google Spanner *by Google*	Yes, using OAuth authentication mechanism.	Yes, using permissions and roles offered by Google Cloud Identity and Access Management (IAM).	Yes, using AES256 or AES 128 encryption algorithms with Google's Key Management Service.	Yes, using TLS/SSL.	ACID	Yes, it is supported as a part of Cloud Audit Logging which generates different types of data access logs and activity event logs.
31	NuoDB *by NuoDB Inc.*	Yes, using basic password-based authentication or LDAP based authentication mechanisms.	Yes, using standard SQL roles and privileges with administrative accounts.	No. Can be used with 3rd party encryption engines.	Yes, NuoDB uses network encryption along with Secure Remote Password protocol (SRP) (RFC-2945) for mutual authentication between application clients and NuoDB nodes.	ACID	Yes, different logging categories including database operations, security and management events.
32	Clustrix *by Clustrix Inc.*	Yes, using password-based authentication.	Yes, using permission based (database level or table level) access control mechanism.	Yes, using AES 256-bit encryption algorithm.	Yes, using SSH access for cluster access. Database access is not encrypted.	ACID	Yes, different set of logging for queries and management events.

there are hundreds of different database systems available, for this survey it was considered only the most popular database systems in each different category according to the DB-Engines rankings [22]. As the summary of results in Table 2 implies, the first two columns were grouped according to the different data storage models based on their storage architecture and popularity. The security criterion/mechanisms that were investigated are listed in the rest of the columns. Additionally, the encryption mechanisms have been further classified into two different groups to have a broader intuition about how data encryption mechanisms have been implemented on different database systems. Moreover, the consistency model explains how strong the devised mechanism for data integrity in these database solutions.

It is noted that in overall relational database systems have very strong set of security assurances compared to other data models. All datastores that have been surveyed under the category of relational database model have demonstrated required standard security mechanisms which can ensure better protection for the data. Moreover, systems like Microsoft SQL Server has outperformed most established systems and presented some additional offerings such as client-side encryption mechanisms. Along with such client-side encryption tools, it ensures that data remains encrypted not just over the network, but also in memory and on the drive as well. It is well understood that availability of such integrated security mechanisms have influenced much to establish relational model as the most prominent data model for handling complex web-based applications during the last few decades.

On the flip side, it can be seen that most of the NoSQL models do not have sufficient mechanisms to ensure data security. Majority of them have simple password based client-side authentication mechanisms but it is clear that rest of the security mechanisms (such as authorization, access control, encryption etc.) are not appeared in most of the NoSQL systems. In the case of key-value systems Redis provides password based authentication however, these passwords are stored in plain-text set by system administrators and it does not provide authentication by default (listens all connections on port 6,739). In fact, it also does not provide any sort of encryption, access control or logging mechanisms. It is further observed that only DynamoDB has the integrated mechanisms to provide data encryption while rest of the systems do not have such mechanism other than relying on third party SSL/TLS implementations to protect the data transmission over the network.

However, in the category of wide-column datastores, all of the surveyed databases have demonstrated at least some combination of multiple security mechanisms. But still, Cassandra only provides comparatively weak password based authentication where passwords are stored just using MD5 hash, and inter-node communication in Cassandra does not have authentication and encryption by default. Thereby, it is somewhat vulnerable for malicious attackers who might have access to the communication network (they have a separate Datastax enterprise version which supports TDE). In the case of HBase, it does not support high level auditing and logging facilities.

From the survey of document-oriented databases Couchbase, CouchDB and RethinkDB do not have integrated mechanisms to provide encryption for data-at-rest even though they have slightly different implementations for rest of the security mechanisms. On the other hand, majority of the graph databases do not facilitate most of the security mechanisms except some means for authentication. In the case of Apache Giraph, it has none of the security mechanisms except simple authentication. Moreover, it is also noteworthy that most of the NoSQL solutions only provide very basic built-in support for network level security (inter-node and client server) instead they recommend to integrate third party solutions such as VPN or SSL/TLS based mechanisms for data communication. Most of the databases support auditing and logging at database/table level but they lack the provision for automated auditing features in their open-source releases. Hence, in overall, NoSQL systems still requires much attention to improve the security; at least by providing several different built-in data protection mechanisms.

In a context where NewSQL systems, it is observed that even though the NewSQL systems are still serving/performing at their learning curve, they have sufficiently high set of security mechanisms compared to NoSQL data models. Yet, Apache Ignite, one of the popular database in this category does not even have integrated mechanism to protect data in terms of access control, data encryption and auditing. In addition, VoltDB and NuoDB do not support this functionality either, even though larger fraction of other NewSQL databases support encryption at-rest.

Finally, it is also worthy to note that most of the cloud-based database services that have been surveyed (such as Azure Cosmos DB, Google Bigtable, Amazon DynamoDB) are having complete fine-grained set of security mechanisms making them well-suited for secure Big Data applications. Moreover, because of the integrated security mechanisms, the value and the popularity of NewSQL databases have risen, making numerous avenues for today's data-driven applications in Big Data paradigm.

5 Conclusion and Avenues for Enhancing Security in Database Systems

Over the past 15 years, cloud-computing has emerged as a distributed computing paradigm which can cater the immense requirements of database systems of modern data-driven applications. In par with this new wave of technology, a lot of different new database architectures such as NoSQL and NewSQL have emerged. However, the continued role of relational databases still has a significant impact on today's promising database architectures because of their integrated implementations of security mechanisms compared to other database models. Information security is one of the top priorities of today and many organizations store their mission-critical data still on-premises with relational databases where they believe it is safer. Hence, most of the non-relational datastores do not fit in to a potential avenue in enterprise level integrations even they are more heightened for on-cloud distributed operations.

However, organizations are still exploring the different possibilities to move towards data management technologies other than the relational model. As with the performance attributes provided by different NoSQL models, there are many outperforming alternatives for relational database systems that are bundled with lot of additional benefits. However, ensuring security on these systems is a challenging task. This study has mainly focused on security and privacy implementations on different database solutions and as of the findings of this survey suggested, it is high time for most of the NoSQL datastores to revisit their security mechanisms and remodel them as fine-grained secure solutions. Most importantly, many of the NoSQL database systems lack encryption mechanisms that support security for data-at-rest and data-in-transit, which is one of the crucial requirements for datastores in the cloud-based production environment. Hence, it is worth to note that exhaustive studies on secure non-relational database systems have prominent opportunities and great potential for future research in security-aware database systems.

On the other hand, there are several factors which drive the choice of storage infrastructure for different kind of data. Business analytics is one of the key considerations in today's applications. As traditional relational model does not fit well with business analytics, NewSQL datastores has the potential to cater this demand. On an information security perspective, as most NewSQL database systems are still evolving, their guarantees for data security are considerably low compared to the relational database systems. Moreover, most of them are in-memory solutions and they have relatively overlooked the requirements of data security and privacy. Hence, sophisticated security provisions are still needed for NewSQL datastores. Furthermore, it was revealed that tightening security on these systems should not degrade the performance of the datastore irrespective of the demand for real-time transactions.

In such circumstance, continuing to look for ways to build cryptographic primitives and systems that achieve better security and privacy in stronger threat models while preserving performance is the future research direction for next generation database systems. Finally, over the past decade several new exciting technologies including Hadoop have been introduced and those technologies have had great influence in database systems. Thus, as some of the literature suggested, these open-source database solutions should no longer be seen as "new" approach. Instead these should be matured as viable alternatives for existing traditional database systems where these too can fit in to the actual production environment. Hence, most promising approach to popularize those systems is to strengthen the security and privacy guarantees of these database systems.

References

[1] E. King, "The 2016 enterprise data management," 2016. [Online]. Available: http://www.dbta.com/DBTA-Downloads/ResearchReports/The-2016-Enterprise-Data-Management-Survey-6555.aspx. Accessed: Jan. 1, 2018.

[2] J. R. Lourenço, B. Cabral, P. Carreiro, M. Vieira, and J. Bernardino, "Choosing the right NoSQL database for the job: A quality attribute evaluation," *J. Big Data*, vol. 2, no. 1, 2015, Art. no. 18.

[3] M. A. Mohamed, O. G. Altrafi, and M. O. Ismail, "Relational vs. NoSQL databases: A survey," *Int. J. Comput. Inf. Technol.*, vol. 3, no. 03, pp. 598–601, 2014.

[4] K. Grolinger, W. A. Higashino, A. Tiwari, and M. A. Capretz, "Data management in cloud environments: NoSQL and NewSQL data stores," *J. Cloud Comput.*, vol. 2, no. 1, 2013, Art. no. 49.

[5] G. Harrison, *Next Generation Databases: NoSQL, NewSQL, and Big Data*. Jan. 2015.

[6] E. A. Brewer, "Towards robust distributed systems," in *Proc. 19th Annu. ACM Symp. Principles Distrib. Comput.*, 2000, Art. no. 7.

[7] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier, "Cluster-based scalable network services," *ACM SIGOPS Operating Syst. Rev.*, vol. 31, pp. 78–91, 1997.

[8] E. Brewer, "Cap twelve years later: How the "rules" have changed," *Comput.*, vol. 45, no. 2, pp. 23–29, 2012.

[9] V. N. Gudivada, D. Rao, and V. V. Raghavan, "NoSQL systems for big data management," in *Proc. IEEE World Congress Services*, 2014, pp. 190–197.

[10] D. Crockford, "The application/json media type for JavaScript object notation (JSON)," 2006.

[11] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, no. 1, 2008, Art. no. 1.

[12] N. Leavitt, "Will NoSQL databases live up to their promise?" *Comput.*, vol. 43, no. 2, pp. 12–14, Feb. 2010.

[13] G. Harrison, *Next Generation Databases: NoSQLand Big Data*. New York, NY, USA: Apress, 2015.

[14] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. Jones, S. Madden, M. Stonebraker, Y. Zhang, et al., "H-store: A high-performance, distributed main memory transaction processing system," *Proc. VLDB Endowment*, vol. 1, no. 2, pp. 1496–1499, 2008.

[15] J. Piekos, "SQL vs. NoSQL vs. NewSQL: Finding the right solution," 2015. [Online]. Available: http://dataconomy.com/2015/08/sql-vs-nosql-vs-newsql-finding-the-right-solution/. Accessed: Jan. 1, 2018.

[16] J. Doppelhammer, T. Höppler, A. Kemper, and D. Kossmann, "Database performance in the real world: TPC-D and SAP R/3," *ACM SIGMOD Rec.*, vol. 26, pp. 123–134, 1997.

[17] A. Kemper and T. Neumann, "HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots," in *Proc. IEEE 27th Int. Conf. Data Eng.*, 2011, pp. 195–206.

[18] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang, "In-memory big data management and processing: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 7, pp. 1920–1948, Jul. 2015.

[19] M. B. V., "MonetDB," 2002. [Online]. Available: https://www.monetdb.org/. Accessed on: Jan. 01, 2018

[20] S. Manegold, M. L. Kersten, and P. Boncz, "Database architecture evolution: Mammals flourished long before dinosaurs became extinct," *Proc. VLDB Endowment*, vol. 2, no. 2, pp. 1648–1653, 2009.

[21] C. Binnig, S. Hildenbrand, and F. Färber, "Dictionary-based order-preserving string compression for main memory column stores," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 283–296.

[22] DB-Engines, "DB-engines ranking," 2018. [Online]. Available: https://db-engines.com/en/ranking. Accessed on: May 20, 2018

[23] O. Corporation, "Oracle database," 1979. [Online]. Available: https://www.oracle.com/database/index.html. Accessed on: May 20, 2018

[24] IBM, "IBM informix," 2001. [Online]. Available: https://www.ibm.com/analytics/informix. Accessed on: Jul. 01, 2018

[25] O. Corporation, "MySQL," 1995. [Online]. Available: https://www.mysql.com/. Accessed on: May 30, 2018

[26] O. Corporation, "Oracle TimesTen," 1996. [Online]. Available: https://www.oracle.com/database/timesten-in-memory-database/index.html. Accessed on: Jul. 01, 2018

[27] Microsoft, "SQL server," 1989. [Online]. Available: https://www.microsoft.com/en-us/sql-server/default.aspx. Accessed on: May 20, 2018

[28] P. G. D. Group, "PostgreSQL," 1996. [Online]. Available: https://www.postgresql.org/. Accessed on: May 20, 2018

[29] IBM, "IBM DB2," 1983. [Online]. Available: https://www.ibm.com/analytics/us/en/db2/. Accessed on: Jul. 01, 2018

[30] Amazon, "Amazon DynamoDB," 2012. [Online]. Available: https://aws.amazon.com/dynamodb/. Accessed on: May 30, 2018

[31] S. Sanfilippo, "Redis," 2009. [Online]. Available: https://redis.io. Accessed on: May 30, 2018

[32] B. Technologies, "RIAK KV," 2010. [Online]. Available: http://basho.com/products/riak-kv/. Accessed on: May 30, 2018

[33] D. Interactive, "Memcached," 2003. [Online]. Available: https://memcached.org/. Accessed on: Jun. 10, 2018

[34] A. S. Foundation, "Cassandra," 2008. [Online]. Available: http://cassandra.apache.org/. Accessed on: May 30, 2018

[35] M. Inc, "MongoDB," 2009. [Online]. Available: https://www.mongodb.com/. Accessed on: May 30, 2018.

[36] A. S. Foundation, "CouchDB," 2005. [Online]. Available: http://couchdb.apache.org/. Accessed on: May 30, 2018

[37] Aerospike, "Aerospike," 2010. [Online]. Available: https://www.aerospike.com/. Accessed on: Jul. 01, 2018

[38] A. S. Foundation, "Apache Accumulo," 2008. [Online]. Available: https://accumulo.apache.org/. Accessed on: Jun. 05, 2018
[39] A. GmbH, "ArangoDB," 2011. [Online]. Available: https://www.arangodb.com/. Accessed on: Jul. 02, 2018
[40] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, 2008, Art. no. 4.
[41] Hazelcast, "Hazelcast," 2009. [Online]. Available: https://hazelcast.com/. Accessed on: Jul. 01, 2018
[42] C. Inc., "Couchbase," 2010. [Online]. Available: https://www.couchbase.com/. Accessed on: Jul. 01, 2018
[43] O. Ltd, "OrientDB," 2010. [Online]. Available: https://orientdb.com/. Accessed on: May 30, 2018
[44] N. Technology, "Neo4J," 2007. [Online]. Available: https://neo4j.com/. Accessed on: May 30, 2018
[45] Amazon, "Amazon Neptune," 2017. [Online]. Available: https://aws.amazon.com/neptune/. Accessed on: Jun. 05, 2018
[46] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al., "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, 2013, Art. no. 8.
[47] S. SE, "SAP HANA," 2010. [Online]. Available: https://www.sap.com/products/hana.html. Accessed on: Jul. 02, 2018
[48] A. P. Stonebraker and Michael, "Vertica," 2005. [Online]. Available: https://www.vertica.com/. Accessed on: Jul. 01, 2018
[49] V. Inc, "VoltDB," 2015. [Online]. Available: https://www.voltdb.com/. Accessed on: May 20, 2018
[50] M. Inc, "MemSQL," 2013. [Online]. Available: https://www.memsql.com/. Accessed on: May 20, 2018
[51] A. S. Foundation, "Apache Ignite," 2015. [Online]. Available: https://ignite.apache.org/. Accessed on: May 25, 2018
[52] NuoDB, "NuoDB," 2008. [Online]. Available: http://www.nuodb.com/. Accessed on: May 20, 2018
[53] M. Inc., "Hekaton," 2014. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/sql-server-in-memory-oltp-internals-for-sql-server-2016?view=sql-server-2017. Accessed on: Jul. 02, 2018
[54] SAP, "Data 2020: State of big data study data sources, connectivity & IT frameworks," 2017. [Online]. Available: https://news.sap.com/wp-content/blogs.dir/1/files/SAPData-2020-Study Infographic.pdf/. Accessed: Jan. 1, 2018.
[55] E. Bertino and R. Sandhu, "Database security-concepts, approaches, and challenges," *IEEE Trans. Depend. Sec. Comput.*, vol. 2, no. 1, pp. 2–19, Jan.–Mar. 2005.
[56] S. Srinivas and A. Nair, "Security maturity in NoSQL databases-are they secure enough to haul the modern it applications?" in *Proc. Int. Conf. Advances Comput. Commun. Informat.*, 2015, pp. 739–744.
[57] P. Grubbs, T. Ristenpart, and V. Shmatikov, "Why your encrypted database is not secure," in *Proc. 16th Workshop Hot Topics Operating Syst.*, 2017, pp. 162–168.
[58] P. Grubbs, M.-S. Lacharité, B. Minaud, and K. G. Paterson, "Learning to reconstruct: Statistical learning theory and encrypted database attacks," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 1–62.
[59] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic attacks on secure outsourced databases," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1329–1340.
[60] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *Proc. 23rd ACM Symp. Operating Syst. Principles*, 2011, pp. 85–100.
[61] R. Poddar, T. Boelter, and R. A. Popa, "Arx: A strongly encrypted database system," *IACR Cryptology ePrint Archive*, vol. 2016, 2016, Art. no. 591.
[62] N. Johnson, J. P. Near, and D. Song, "Towards practical differential privacy for SQL queries," *Proc. VLDB Endowment*, vol. 11, no. 5, pp. 526–539, 2018.
[63] A. Ron, A. Shulman-Peleg, and A. Puzanov, "Analysis and mitigation of NoSQL injections," *IEEE Secur. Privacy*, vol. 14, no. 2, pp. 30–39, Mar./Apr. 2016.
[64] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 644–655.
[65] M.-S. Lacharité, B. Minaud, and K. G. Paterson, "Improved reconstruction attacks on encrypted data using range query leakage," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 297–314.
[66] F. B. Durak, T. M. DuBuisson, and D. Cash, "What else is revealed by order-revealing encryption?" in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1155–1166.
[67] P. Frühwirt, P. Kieseberg, S. Schrittwieser, M. Huber, and E. Weippl, "InnoDB database forensics: Reconstructing data manipulation queries from redo logs," in *Proc. 7th Int. Conf. Availability Rel. Secur.*, 2012, pp. 625–633.
[68] P. Grubbs, R. McPherson, M. Naveed, T. Ristenpart, and V. Shmatikov, "Breaking web applications built on top of encrypted data," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1353–1364.
[69] T. Garfinkel and M. Rosenblum, "When virtual is harder than real: Security challenges in virtual machine based computing environments," in *Proc. 10th Conf. Hot Topics Operating Syst.*, 2005, pp. 20–20.
[70] T. Ristenpart and S. Yilek, "When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2010, pp. 1–18.
[71] M. Jensen, "Challenges of privacy protection in big data analytics," in *Proc. IEEE Int. Congress Big Data*, 2013, pp. 235–238.
[72] J. R. Palanco, *NoSQL Security*. 1 Ed. Amsterdam, Netherlands: Elsevier Inc., 2011.
[73] U. C. Framework, "Database security requirements guide," 2017. [Online]. Available: https://www.stigviewer.com/stig/database_security_requirements_guide/. Accessed on: Jan. 15, 2019
[74] R. Duncan, "An overview of different authentication methods and protocols," *SANS Institute*, 2001.
[75] N. Delessy, E. B. Fernandez, M. M. Larrondo-Petrie, and J. Wu, "Patterns for access control in distributed systems," in *Proc. 14th Conf. Pattern Lang. Programs*, 2007, Art. no. 3.
[76] U. S. N. I. of Standards and T. (NIST), "Announcing the advanced encryption standard (AES)," 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf. Accessed: Jan. 1, 2018.
[77] T. Nie and T. Zhang, "A study of DES and Blowfish encryption algorithm," in *Proc. IEEE Region 10 Conf.*, 2009, pp. 1–4.
[78] E. Rescorla, *SSL and TLS: Designing and Building Secure Systems*, vol. 1. Boston, MA, USA: Addison-Wesley Reading, 2001.
[79] Microsoft, "Always Encrypted (Database Engine)," 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine?view=sql-server-2017. Accessed on: Jun. 25, 2018
[80] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," *Proc. VLDB Endowment*, vol. 6, pp. 289–300, 2013.
[81] J. Li, Z. Liu, X. Chen, F. Xhafa, X. Tan, and D. S. Wong, "L-EncDB: A lightweight framework for privacy-preserving data queries in cloud computing," *Knowl.-Based Syst.*, vol. 79, pp. 18–26, 2015.
[82] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan, "Big data analytics over encrypted datasets with seabed," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 587–602.
[83] E. Pattuk, M. Kantarcioglu, V. Khadilkar, H. Ulusoy, and S. Mehrotra, "BigSecret: A secure data management framework for key-value stores," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, 2013, pp. 147–154.
[84] X. Yuan, X. Wang, C. Wang, C. Qian, and J. Lin, "Building an encrypted, distributed, and searchable key-value store," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, 2016, pp. 547–558.
[85] M. Ahmadian, F. Plochan, Z. Roessler, and D. C. Marinescu, "SecureNoSQL: An approach for secure search of encrypted NoSQL databases in the public cloud," *Int. J. Inf. Manage.*, vol. 37, no. 2, pp. 63–74, 2017.
[86] R. Macedo, J. Paulo, R. Pontes, B. Portela, T. Oliveira, M. Matos, and R. Oliveira, "A practical framework for privacy-preserving NoSQL databases," in *Proc. IEEE 36th Symp. Reliable Distrib. Syst.*, 2017, pp. 11–20.
[87] F. Kerschbaum, "Frequency-hiding order-preserving encryption," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 656–667.
[88] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Proc. Annu. Cryptology Conf.*, 2011, pp. 578–595.
[89] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proc. 2nd Int. Workshop Hardware Architectural Support Secur. Privacy*, 2013, vol. 10, Art. no. 10.

[90] A. Arasu, S. Blanas, K. Eguro, M. Joglekar, R. Kaushik, D. Kossmann, R. Ramamurthy, P. Upadhyaya, and R. Venkatesan, “Secure database-as-a-service with cipherbase,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 1033–1036.

[91] S. Bajaj and R. Sion, “TrustedDB: A trusted hardware-based database with privacy and data confidentiality,” *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 3, pp. 752–765, Mar. 2014.

[92] C. Priebe, K. Vaswani, and M. Costa, “EnclaveDB: A secure database using SGX,” in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 264–278.

[93] K. Barker, M. Askari, M. Banerjee, K. Ghazinour, B. Mackas, M. Majedi, S. Pun, and A. Williams, “A data privacy taxonomy,” in *Proc. Brit. Nat. Conf. Databases*, 2009, pp. 42–54.

[94] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, “Hippocratic databases,” in *Proc. 28th Int. Conf. Very Large Databases*, 2002, pp. 143–154.

[95] A. Aldini, R. Gorrieri, and F. Martinelli, *Foundations of Security Analysis and Design III: FOSAD 2004/2005 Tutorial Lectures*, vol. 3655. Berlin, Germany: Springer, 2005.

[96] L. Sweeney, “k-anonymity: A model for protecting privacy,” *Int. J. Uncertainty Fuzziness Knowl.-Based Syst.*, vol. 10, no. 05, pp. 557–570, 2002.

[97] J. Vaidya and C. Clifton, “Privacy-preserving data mining: Why, how, and when,” *IEEE Secur. Privacy*, vol. 2, no. 6, pp. 19–27, Nov./Dec. 2004.

[98] C. Clifton, “Using sample size to limit exposure to data mining,” *J. Comput. Secur.*, vol. 8, no. 4, pp. 281–307, 2000.

[99] W3C, “The Platform for Privacy Preferences 1.0 (P3P1.0) Specification,” 2002. [Online]. Available: https://www.w3.org/TR/P3P/. Accessed on: Jun. 25, 2018

[100] E. Bertino, J.-W. Byun, and N. Li, “Privacy-preserving database systems,” in *Proc. Int. School Found. Secur. Anal. Des. III*, 2005, pp. 178–206.

[101] N. R. Adam and J. C. Worthmann, “Security-control methods for statistical databases: A comparative study,” *ACM Comput. Surv.*, vol. 21, no. 4, pp. 515–556, 1989.

[102] M. Hosenball, “Swiss spy agency warns U.S., britain about huge data leak,” 2012. [Online]. Available: https://reut.rs/2SEZCdw. Accessed on: Jan. 15, 2019

[103] C. Terhune, “Nearly 5,000 patients affected by UC Irvine medical data breach,” 2015. [Online]. Available: https://www.latimes.com/business/la-fi-uc-irvine-data-breach-20150618-story.html. Accessed on: Jan. 15, 2019

[104] J. Vijayan, “Morgan stanley breach a reminder of insider risks,” [Online]. Available: https://securityintelligence.com/news/morgan-stanley-breach-reminder-insider-risks/. Accessed on: Jan. 15, 2019

[105] L. Brankovic and H. Giggins, *Statistical Database Security*. Berlin, Germany: Springer, 2007, pp. 167–181.

[106] J. J. Kim, “A method for limiting disclosure in microdata based on random noise and transformation,” in *Proc. Section Surv. Res. Methods*, 1986, pp. 303–308.

[107] P. Tendick, “Optimal noise addition for preserving confidentiality in multivariate data,” *J. Statistical Planning Inference*, vol. 27, no. 3, pp. 341–353, 1991.

[108] J. Kim and W. Winkler, “Multiplicative noise for masking continuous data,” *Statist.*, vol. 1, pp. 1–18, 2003.

[109] C. Dwork, “Differential privacy: A survey of results,” in *Proc. Int. Conf. Theory Appl. Models Comput.*, 2008, pp. 1–19.

[110] C. Dwork, A. Roth, et al., “The algorithmic foundations of differential privacy,” *Found. Trends® Theoretical Comput. Sci.*, vol. 9, no. 3/4, pp. 211–407, 2014.

[111] F. D. McSherry, “Privacy integrated queries: An extensible platform for privacy-preserving data analysis,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 19–30.

[112] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler, “GUPT: Privacy preserving data analysis made easy,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 349–360.

[113] A. Narayan and A. Haeberlen, “DJoin: Differentially private join queries over distributed databases,” in *Proc. 10th USENIX Conf. Operating Syst. Des. Implementation*, 2012, pp. 149–162.

[114] K. Nissim, S. Raskhodnikova, and A. Smith, “Smooth sensitivity and sampling in private data analysis,” in *Proc. 39th Annu. ACM Symp. Theory Comput.*, 2007, pp. 75–84.

[115] P. Huey, “Oracle database security guide,” 2017. [Online]. Available: https://docs.oracle.com/cd/E1188201/network.112/e36292/toc.htm. Accessed: Jan. 1, 2018.

[116] A. S. Foundation, “Apache Log4j,” 2001. [Online]. Available: https://logging.apache.org/log4j/2.x/. Accessed on: Jun. 25, 2018

[117] C. Gulcu, “Simple logging facade for Java,” 2013. [Online]. Available: https://www.slf4j.org/manual.html. Accessed on: Jun. 25, 2018

G. Dumindu Samaraweera received the BSc degree in computer systems and networking from Curtin University, Australia, and MSc degree in enterprise application development from Sheffield Hallam University, United Kingdom, in 2009 and 2013, respectively. He started his carrier as a systems analyst/software engineer and then served as an electrical engineer, currently reading for the PhD degree in electrical engineering. His current research interests include cloud computing, security/privacy preserving database systems, and cyber security. He is an associate member of the Institution of Engineers, Sri Lanka, member of BCS (United Kingdom), and a student member of the IEEE.

J. Morris Chang received the BSEE degree from the Tatung Institute of Technology, Taiwan, and the MS and PhD degrees in computer engineering from North Carolina State University. He is currently a professor with the Department of Electrical Engineering, University of South Florida. His industrial experience includes positions at Texas Instruments, Taiwan, Microelectronics Center of North Carolina, and AT&T Bell Laboratories, Pennsylvania. He was on the faculty of the Department of Electrical Engineering, Rochester Institute of Technology, Rochester, the Department of Computer Science, Illinois Institute of Technology, Chicago, and the Department of Electrical and Computer Engineering, Iowa State University, IA. His research interests include cyber security, wireless networks, energy-aware computing, and object-oriented systems. Currently, he is a handling editor of the *Journal of Microprocessors and Microsystems* and the associate editor-in-chief of *IEEE IT Professional*. He is a senior member of the IEEE.