UDK 004.6:004.632

NOSQL DATABASES. TECHNOLOGY FOR PROTECTING DATA FROM UNAUTHORIZED ACCESS

A.V. Kuchynski
Student of Belarusian State University of Informatics and Radioelectronics. Software engineer IBA-Group.

U.N. Hutkouski
Student of Belarusian State University of Informatics and Radioelectronics. Software engineer IBA-Group.

I.I. Piletski
PhD, Associate Professor of Informatics Department of the BSUIR

Belarusian State University of Informatics and Radioelectronics, Republic of Belarus
IBA-Group, Republic of Belarus
E-mail: alexkuchinskydev@gmail.com

A.V. Kuchynski
Pre-graduate student at Belarusian State University of Informatics and Radioelectronics. Certified AWS Architect, works as a Big Data engineer at IBA Group. Focuses on cloud computing, distributed frameworks, security of systems.

U.N. Hutkouski
Pre-graduate student at Belarusian State University of Informatics and Radioelectronics. Works as a project manager at IBA Group. Interested in full stack programming, high-availability and elastic architectures.

I.I. Piletski
PhD of Maths, associate professor of BSUIR. In the field of IT for more than 47 years. Participation in the development of several dozen major projects: the chief designer of the project, the chief architect of software and information support, the project manager, the head of the department, the head of the laboratory (Scientific Research Institute of Computer Science, Academy of Sciences of Belarus, IBA, BSUIR). The author of dozens of research and publications.

Abstract. Companies such as banks, medical facilities, government units does not want their data was available and used by third parties (It can be result of hackers' attacks, data leaks, and employee fraud). Just write encrypted data to your data storage is not enough. This data storage has to allow business users to work with its content performing CRUD (create, read, update, delete) operations and search queries. During encryption, we are using some function F(x) to change content of our data, getting cyphertext as a result, after encryption takes place we lose this ability of performing search and basic operations. We are proposing approach, which allows managing encrypted data by executing queries on it without decryption, results returned in encrypted form to client where he/she can decrypt it using self-managed keys.

Keywords: NoSQL, graph database, protection from unauthorized access, security.

Introduction. Basically there are two states of data when it's vulnerable for potential thieves:

1. Data at rest – is data that is not actively moving from device to device or network to network such as data stored on a hard drive, laptop, flash drive, or archived/stored in some other way. Data protection at rest aims to secure inactive data stored on any device or network, so in case if somebody physically takes disks from the computer they will not be able to reveal the data stored on it.

2. Data in transit - or data in motion, is data actively moving from one location to another such as across the internet or through a private network. Data protection in transit is the protection of this data while it's traveling from network to network or being transferred from a local storage device to a cloud storage device

There are two places where data can be encrypted:

1. On the client side (client-side encryption) - information is encrypted before it's send to database. Client itself chooses encryption algorithm and manages encryption keys. Data is transferred and stored already encrypted.
2. On the server side (server-side encryption) - server performs encryption of the data when it's written and decryption when it's read. In this case encryption keys can be stored by client or by server.

Choosing the place where data is encrypted defines security level:

1. Level 0. No encryption
2. Level 1. Server side encryption, server keeps keys.
3. Level 2. Server side encryption, keys are kept by client
4. Level 3. Client-side encryption.

Challenges of search in encrypted NoSQL storages. Firstly, when encryption takes place and we write encrypted data into database, we lose ability of standard search over this data (using of indexes). Such operations as: greater than, less than, equals, in range, etc. become not supportable as columns contains ciphertext [1].

Secondly, specification of NoSQL databases have to be considered. In this types of storages search is performed based on the key (almost all NoSQL databases are key-value storages). Value which is mapped to the key can be unstructured data (or I would prefer to name it multistructured), so there is no definite database scheme as in relational databases. So there is a question: over which fields exactly we are searching over? Because most of the times all business crucial information is contained in the key (several attributes can be concatenated) [1, 2].

It's also worth mentioning that special tools do exist for performing column search. This services are built on the top of the database, maintain their own indexes, define their own data schemas and mappings, provide separate interface for querying data (Example: Solr, Phoenix).

Basic principles. During construction process of the system which supports search over encrypted data it would be preferable to keep up in mind the following principles:

– Transparency – implementation of encryption/decryption mechanism and mechanism for search over encrypted data shouldn't require changes of existing components in the system or if changes takes place it shouldn't be huge (shown on figure 1).

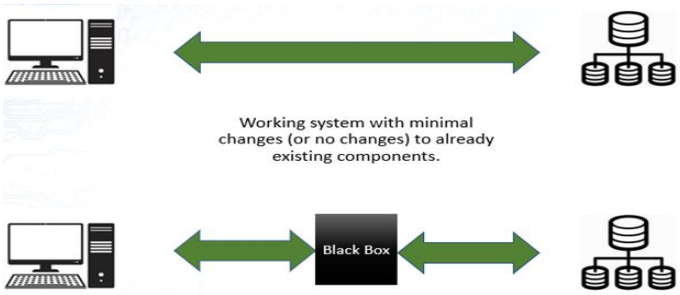

Figure *1*. – Illustration of transparency concept

– SQL-like queries. We discussed previously search constraints of NoSQL (key-value) databases. In this types of storages we can search specific entities only by key or part of the key. Examples of get methods for reading from NoSQL database:

- table.get("row_identificator")
- table.scan()
- table.scan("A*")

Otherwise, SQL-like query allows to search over all columns:

- SELECT * FROM People Where name like 'A*'
- SELECT * FROM People WHERE age < 25 and 0.14*year_salary > 5000

– Proper type of algorithm. There are different classes of encryption algorithms which allows to perform specific operations on encrypted data:

- Homomorphic encryption algorithms - allows execute arithmetic operations on two encrypted values without decryption (without key)
- Order revealing encryption - allows execute comparison operations on two encrypted values without decryption (without key)

– Rationalism between security level of algorithm and space efficiency/performance.

Proposed framework. The main problem of encrypted data in database is: we cannot use default-indexing tool, which is based on simple values comparison. Theoretically, ORE (order revealing encryption) algorithms have special function Compare(enc1, enc2) which allows us to know which value is greater:

Order Revealing Encryption (ORE): Three algorithms:

$(sk, pk) \leftarrow$ **Keygen** outputs a secret key and a public "comparison" key

$c \leftarrow E_{sk}(x)$ outputs ciphertext

$b \leftarrow Compare(pk, c_1, c_2)$ outputs a bit

Correctness: $x_1 \le x_2 \Leftrightarrow Compare(E_{sk}(x_1), E_{sk}(x_2)) = 1$ **(w.h.p.)**

Our framework is aimed to make it possible to build indexes for encrypted data using graph database and overriding comparison function (Compare(enc1, enc2)) so this indexes can be balanced and searched. Graph database also stores data entities itself. Framework components (shown on Figure 2):

–Client – user application performing queries and CRUD operation.

–Proxy – proxy-server which handles incoming requests and responses. Performs encryption and decryption of query and response.

–Query Engine – parses query and as a result producing execution plan in form of tree where leaf - it's one condition (unit, condition like a < 5); and node - logical primitives (and, or).

–Query Unit – library containing search functions (find less than, greater than, equals, etc.), functions for maintaining indexes in graph database (insert node, balance tree, etc.).

–Database – graph database.

Technical overview of the components:

–**Client Application** interacts with database via http protocol, currently available functionality:

- Write data to the database
- Send queries to the database

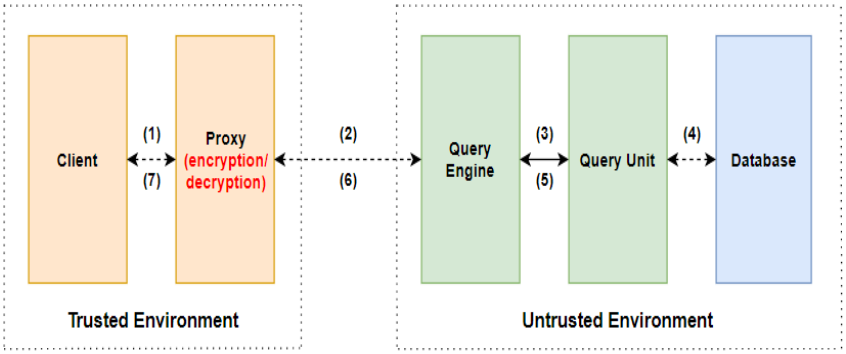

Figure *2*. – High-level framework architecture

–**Proxy** checks incoming and outcoming responses. If there is a special symbols in url, it changes the content of the request (encrypts data to be inserted, encrypts query, decrypts the result of the query) and passes it further to destination or back to client (when it's response). Proxy takes care of encryption/decryption and key is now stored inside the proxy. Proxy also supports caching of encryption/decryption values, so encryption/decryption function will not be used if this value already was encrypted/decrypted in this session. If value is supposed to be indexed it's encrypted with use of OPE algorithm, else 3DES is used [3, 4].

–**Query Engine** builds tree and performs tree traversal executing each leaf, then each node which defines logic operation is applied on the results of each child. Leafs are runned in parallel. Query example: *select * from db.table where (c < 10 and r > 90) or (a = 5 and (m in (9, 50) or k <= 0))*. Execution plan of this query in form of the tree shown on Figure 3.

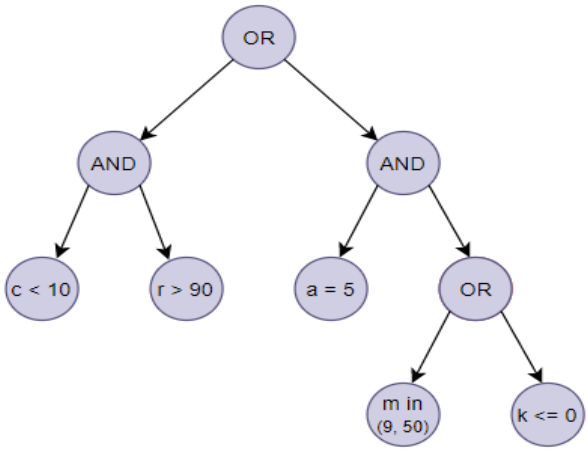

Figure *3*. – Execution plan for the query.

–**Query Unit** is just a library of functions which are used by Query Engine and when insertion of new data takes place.

–**Database**. Indexes are built using AVL Tree data structure. It allows perform search for complexity log(n) (shown on Figure 4).

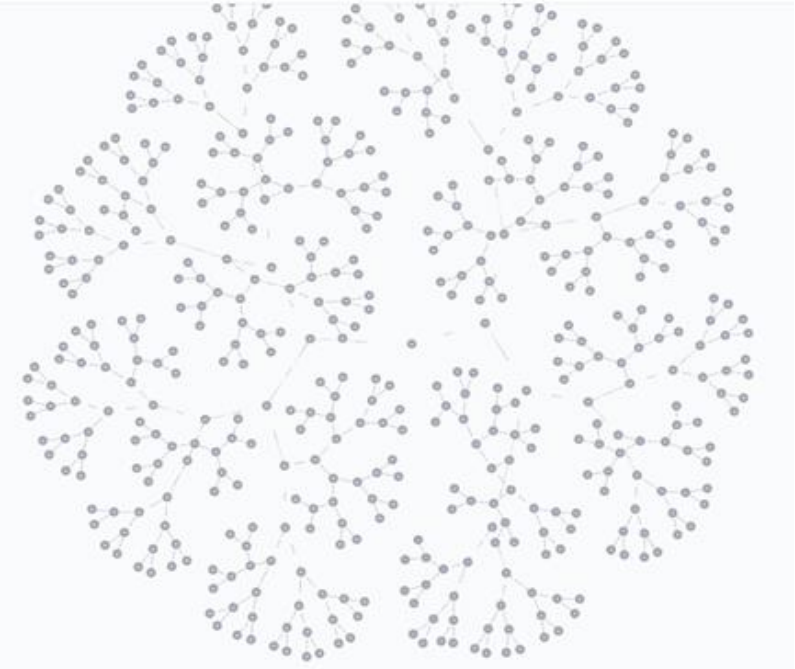

Figure *4*. –View of AVL index tree

For each index column there is an index tree. Tree is updated on each insert operation (update, delete in future). Each node of the tree represents encrypted value. Nodes which are not in the tree - data entities. Inserted entity gets reference(s) to node in the tree (shown on Figure 5).

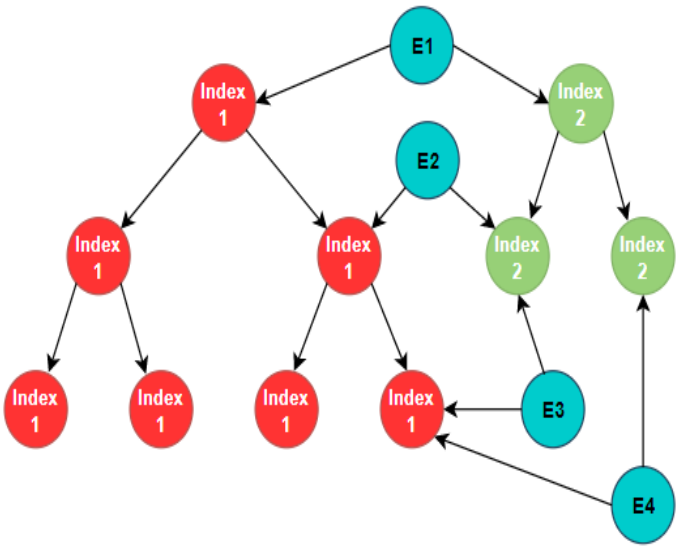

Figure 5. –View of index and entities nodes

Conclusion. Proposed framework allows performing search queries on encrypted data without need to decrypt it on query stage. Data is stored securely, thus cannot be used by unauthorized users. Architecture uses NoSQL graph database as a storage for primarily data and index structures. Order revealing encryption used for fields which have to be searched.

References

[1.]NoSQL Data Architecture & Data Governance: Everything You Need to Know [Электронный ресурс] / Режим доступа: https://www.dataversity.net/nosql-data-architecture-data-governance-everything-need-know/ Дата доступа: 24.02.2020.

[2.]The NoSQL Ecosystem [Электронный ресурс] / Режим доступа: https://www.aosabook.org/en/nosql.html Дата доступа: 24.02.2020.

[3.]Secure Parallel Processing of Big Data Using Order-Preserving Encryption on Google BigQuery [Электронный ресурс] / Режим доступа: https://arxiv.org/pdf/1608.07981.pdf Дата доступа: 24.02.2020.

[4.]What is 3DES encryption and how does DES work? [Электронный ресурс] / Режим доступа: https://www.comparitech.com/blog/information-security/3des-encryption/ Дата доступа: 24.02.2020.

NOSQL DATABASES. TECHNOLOGY FOR PROTECTING DATA FROM UNAUTHORIZED ACCESS

А.В. Кучинский
Студент БГУИР.
Инженер-программист
IBA Group.

В.Н. Гутковский
Студент БГУИР.
Инженер-программист
IBA Group.

И.И. Пилецкий
Доцент кафедры информатики БГУИР, кандидат физико-математических наук, доцент, старший научный сотрудник

Белорусский государственный университет информатики и радиоэлектроники, Республика Беларусь
IBA-Group, Республика Беларусь
E-mail: alexkuchinskydev@gmail.com

Аннотация. Компании (банки, мед. учреждения, государственные компании) не хотят, чтобы их информация (в случае хакер-атаки, утечки данных, неосторожности обслуживающего персонала и т.д.) была доступна злоумышленниками и использовалась 3-й стороной. Просто сложить зашифрованные данные в хранилище недостаточно. Это хранилище должно использоваться и давать возможность работать со своим содержимым посредством обработки поисковых запросов. При шифровании мы меняем реальные значения на результат работы некой функции, одновременно с этим мы теряем возможность поиска по этим данным. Мы предлагаем подход, при котором можно бы было осуществлять поиск в зашифрованных данных без выполнения операции дешифрования при непосредственном поиске.

Ключевые слова: NoSQL, графовая база данных, защита от несанкционированного доступа, безопасность.