The ISC Int'l Journal of Information Security
ISeCure

November 2022, Volume 14, Number 3 (pp. 113–121)
http://www.isecure-journal.org

PRESENTED AT THE ISCISC'2022 IN RASHT, IRAN.

SecureKV: Secure Searchable Outsourcing of Key-Value Databases to the Public Cloud [☆]

Maryam Saeedi Sadr [1], and Mohammad Ali Hadavi [1,*]
[1] *Faculty of Electrical and Computer Engineering, Malek Ashtar University of Technology, Tehran, Iran.*

ARTICLE INFO.

Keywords:
NoSQL, Key-Value Database, Security, Confidentiality, Data Outsourcing, Query Processing, Multi-Cloud

Type:
Research Article

doi:
10.22042/isecure.2022.14.3.12

dor:
20.1001.1.20082045.2022.14.3.12.6

ABSTRACT

The use of NoSQL data and its storage in the Cloud is growing rapidly. Due to the accumulation of data in the Cloud, data security against untrusted service providers as well as external attackers becomes a more serious problem. Over the past few years, there are some efforts to secure the outsourcing of NoSQL data, especially column-based and document-based models. However, practical solutions for secure outsourcing of key-value databases have not been identified. This paper attempts to introduce SecureKV as a secure method for outsourcing key-value databases. This method employs a multi-Cloud storage scenario to preserve outsourced data confidentiality. Besides security issues, the proposed method supports executing major key-value queries directly on outsourced data. A prototype of the Redis database management system has been implemented to show the efficiency and effectiveness of the proposed method. The results imply that, besides security issues, it is efficient and scalable enough in executing key-value-specific queries.

1 Introduction

With the growing demand for large dataset processing and the advantages of storing data on the cloud, companies and individuals are increasingly willing to outsource their data to the cloud. However, data outsourcing incurs serious security issues such as data confidentiality and integrity, user privacy, and enforcing access control policies. These challenges are mainly due to the lack of trust in the cloud service providers. Over the past several years, much research has been done on the security of relational data storage in the cloud [1–3]. While relational databases are designed for structured data, and aspects such as scalability and data distribution are less important in their design, data is changing to *big data*, i.e., semi-structured/unstructured data, which must be stored and processed in a scalable and distributed manner. NoSQL databases are designed to store massive amounts of semi-structured/unstructured data used in data analysis and decision-making in various areas such as industry, science, and medicine. Given the benefits of cloud storage such as cost-saving, on-demand self-service, rapid elasticity, and scalability, the desire of organizations and individuals to store their data on the cloud has increased. There are four types of NoSQL data models: document-oriented, key-

[*] Corresponding author.

[☆] The ISCISC'2022 program committee effort is highly acknowledged for reviewing this paper.

Email addresses: sadr.s.maryam@mut.ac.ir, hadavi@mut.ac.ir

ISSN: 2008-2045 © 2022 ISC. All rights reserved.

value pairs, column-oriented, and graph. There is some reported research on the security of document and column-oriented models when data is outsourced to the cloud [4–6]. However, secure outsourcing of the key-value data with its considerable advantages of flexibility, performance, scalability, continuous availability, and the great potential of high-level support of rich queries and multiple data models [7], has been mostly neglected. To the best of our knowledge, there is no appropriate solution to search on securely outsourced key-value pairs. Existing solutions generally have the following limitations:

- They are not designed to efficiently support specific key-value queries.
- They often outsourced key-value data by changing the data model to tabular data.
- They are limitations to support all key-value data types such as *String*, *List*, *Hash*, *Set*, and *Sorted set*.
- Contrarily to the big data nature, they usually require a predefined schema before outsourcing.
- They focus on data confidentiality and have limitations to providing access confidentiality, as well as pattern confidentiality [8–10].

Considering the above limitations, we propose SecureKV, a multi-cloud-centric key-value store with rich queries and supporting key-value data types. SecureKV uses the idea of intrinsic separation between keys and values. The SecureKV suggests that keys and values are stored in separate servers. SecureKV efficiently supports a variety of key-value-specific queries. In this method, the data owner can choose the encryption algorithm and the number of cloud servers, which store data values, to trade off between performance, availability, and security. Considering the popularity of Redis as a key-value DBMS, we develop a prototype of our method on Redis and experiment with executing its major queries on the string data type. The proposed solution can be easily extended to support other key-value DBMSs. **Paper organization:** Section 2 provides an overview of the related work. Section 3 introduces the system architecture and elaborates on the processing of key-value queries. The implementation and performance evaluation results are given in Section 4. Finally, Section 5 summarizes the paper.

2 Related Work

This section briefly reviews recent studies on the security of relational and NoSQL database outsourcing. **Security in relational data outsourcing:** Encryption and fragmentation are two main approaches to achieving data confidentiality in the cloud [2]. There are also some solutions with the simultaneous use of several encryption algorithms to support different queries on encrypted outsourced data [1]. Cryptographic techniques can also provide data integrity by detecting data tampering [2]. While encryption can be effective in many environments, it brings several complications in scenarios where fine-grained data retrieval needs to be supported. Instead, data fragmentation is used when associations among data values are confidential, rather than the values themselves [2]. Secret sharing has also been used in several studies to preserve privacy in database outsourcing. Agrawal *et al.* [11] use Shamir's secret sharing scheme to preserve outsourced data confidentiality. Hadavi *et al.* [3] introduce a new approach that enables clients to efficiently search among shared secrets. Attasena *et al.* [12] survey secret sharing schemes for database outsourcing and categorize existing approaches into several groups, each of them appropriate in different situations concerning security and performance requirements. Some researchers use the multi-cloud scenario for data security. In these approaches, data is partitioned and distributed among different clouds [13, 14]. Multi-cloud data storage improves availability and security following the idea of "don't put all your eggs in one basket". It can also provide parallel computing capability to reduce query response time. **Security in NoSQL data outsourcing:** Previous approaches focus on securely outsourcing relational data. They cannot be used for NoSQL data due to differences in data models and supported queries. Moreover, they have efficiency and scalability limitations when applied to big data. Poddar *et al.* [15] propose "Arx", a practical and functionally rich database system for the document-based data model. This approach used SQL to execute queries. Suntaxi *et al.* [16] addresses the NoSQL database security problem for graph-structured data. They propose an approach based on bucketization. Shih *et al.* [8] introduce the "Crypt-NoSQL" system, inspired by CryptDB [1], for column-oriented NoSQL databases. It supports Cassandra query language (CQL) over encrypted data. Raza *et al.* [17] propose an approach for document-oriented data models. They recommend splitting the database into two parts. One part contains data attributes used to search the data. Another part contains the search results. They also claim that the database decomposition into two sub-databases improves data confidentiality. All of the mentioned works assume a data model except the key-value model. Some other works transform other data models into the key-value model to preserve data confidentiality. Yuan *et al.* [18] use the key-value model to provide data security. They first store data in a structured table. The table is then converted to key-value pairs, where each key-value pair consists of a key and an encrypted value like "$\langle P(ka, R||C), Enc(kv, V)\rangle$" (R is the row number and C is the column name).

This approach uses searchable encryption to execute queries. The basic idea in their work is to convert data into tables. While it seems that the method is suitable for column-oriented databases, it has limitations to support other data models and their data types. Yuan *et al.* [19] proposed a new approach, which is similar to [18]. They generate some tokens consisting of a column name, row number, and query condition. The tokens are used to execute queries on encrypted data. However, in key-value databases, queries cannot be executed without keys. Also, in almost all queries, the condition is executed on the keys. Therefore, their approach is not suitable for securing key-value databases. Zhang *et al.* [20] proposed a new encrypted key-value storage structure based on the concept of horizontal-vertical division. They first convert data into a group of columns and then compress and encrypt them. Fine-grained data retrieved in this approach is limited to columns. Converting key-value data types to tabular data is challenging and leads to the inefficiency of key-value-specific queries. For example, for the List data type, it is possible to assign either a column to each value in the List or a column to the whole List; both of them result in the inefficiency of key-value queries on the List data type. Some researchers have focused on the secure outsourcing of key-value data. Zaki *et al.* [21] introduced a new approach for providing authentication and confidentiality in the Redis database management system. This approach only supports SET and GET queries. Pontes *et al.* [22] proposed a new approach to preserve the confidentiality of key-value data based on secret sharing and secure multi-party computation. This approach transforms key-value data into tabular data and does not directly support key-value-specific queries and data types. This brief review of current approaches indicates that there is no effective approach for the secure outsourcing of key-value data stores.

3 The Proposed Method

In this section, we first describe our proposed system architecture. Then, we discuss query processing focusing on Redis data types and queries. Our method is simply extendable to other known key-value database management systems, such as Memcached and Etcd. The intuition behind our method is based on the following facts:

(1) Most key-value database queries are executed on keys. Values in rare queries are searchable.
(2) Keys are not often confidential and can be plainly outsourced.

To address data security issues, we use the intrinsic segregation between keys and values and store them in separate index and data servers.

3.1 System Architecture

The architecture of the proposed method is illustrated in Figure 1 We have two domains: the trusted domain including users and the proxy, and the untrusted domain including the index and data servers.

(1) The **user** is an entity that maintains encryption keys and poses queries on outsourced data.
(2) The **proxy** mediates user queries. It transforms them into index/data server apprehensible queries and performs encryption/decryption of values. The proxy in this method is deployed on the user's system.
(3) The **index server** maintains the data structure, stores the keys, and addresses encrypted values in the data servers.
(4) The **data servers** store confidential values in key-value pairs, where a key is an incremental number and the value is the encrypted form of the original value.

In our method, the key-value data is securely outsourced without changing the data model. That is, data is maintained as key-value data in the data servers as well as in the index server. Choosing the number of data servers is a security-cost trade-off. As the number of data servers increases, the potential degree of providing security requirements such as availability, and pattern confidentiality grows with the expense of cloud storage and communication costs.

3.2 Key-Value Data Distribution

Now, we discuss the distribution of a sample key-value pair to the index/data servers. The index server, stores string pairs of key-value. The key is the original data key, and the value is the address of confidential value stored in the data servers. The data servers store key-value pairs in string data type. A key is an incremental number, and its value is the encrypted form of the original value. We require the last key stored on a data server to increment it in the subsequent SET query. For this purpose, the last key of each data server is stored in the table of proxy. Figure 2 is an example of data distribution in our method. This example distributes four pairs of key-value that include user information like username, password, name, and age. In this example, we use two data servers. However, the number of data servers is optional and is chosen according to the trade-off between performance and security. Figure 2 shows how key-value pairs of *String*, *List*, and *Hash* data types are outsourced. These data types are Redis' key-value data types. *String* is the most basic type of value and *List* is a list of strings. *Hash* is a collection of key-value pairs to represent objects. In addition to storing confidential data addresses, the index server

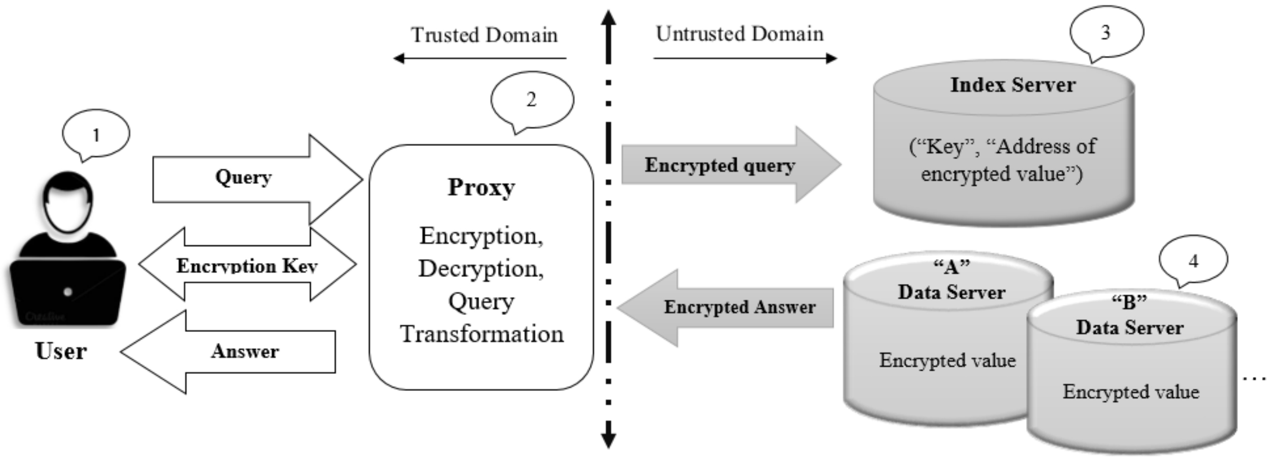
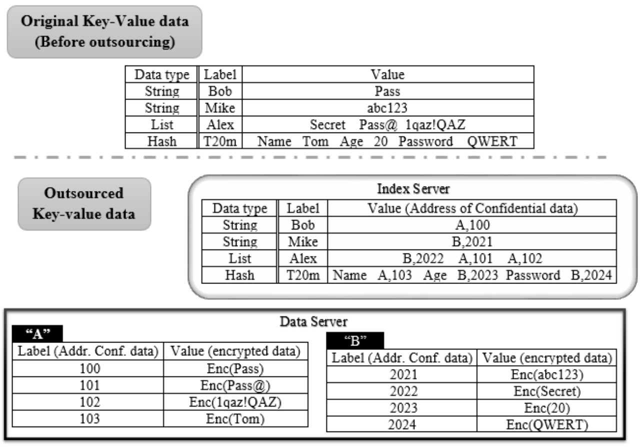


Figure 1. System architecture



Original Key-Value data (Before outsourcing)

Data type	Label	Value
String	Bob	Pass
String	Mike	abc123
List	Alex	Secret Pass@ 1qaz!QAZ
Hash	T20m	Name Tom Age 20 Password QWERT

Outsourced Key-value data

Index Server

Data type	Label	Value (Address of Confidential data)
String	Bob	A,100
String	Mike	B,2021
List	Alex	B,2022 A,101 A,102
Hash	T20m	Name A,103 Age B,2023 Password B,2024

Data Server

"A"

Label (Addr. Conf. data)	Value (encrypted data)
100	Enc(Pass)
101	Enc(Pass@)
102	Enc(1qaz!QAZ)
103	Enc(Tom)

"B"

Label (Addr. Conf. data)	Value (encrypted data)
2021	Enc(abc123)
2022	Enc(Secret)
2023	Enc(20)
2024	Enc(QWERT)

Figure 2. An example of key-value data outsourcing

also keeps data structures. This figure also illustrates that our method preserves the data model in the sense that all key-value pairs are outsourced in the key-value format.

3.3 Query Processing

A query searches keys in key-value databases. In this method, keys and encrypted values are stored in two separate clouds. The proxy transforms users' queries into new queries posed to the index server or the data servers. In this section, we describe query processing in the Redis syntax. We exemplify it by "Bob-Pass" as a key-value pair where "Pass" is a confidential value, encrypted and stored on the data server A as well as other data servers. The process performed in a data server is similarly performed in other data servers, so in the query processing scenarios, we consider only a data server, i.e., data server A.

3.3.1 *SET* Queries

The user wants to store a key-value pair, e.g., "Bob-Pass", so he sends `SET \Bob" \pass"` to the proxy. Since "pass" is a confidential value, the proxy encrypts it and selects a data server, e.g., data server A, to store the encrypted value. To this purpose, the proxy looks up the data server "A" in the table to find its "lastkey". Assume "lastkey" is "100". Then, the proxy poses two queries, `SET \100" \Enc(pass)"` to the data server and `SET \Bob" \A,100"` to the index server. Finally, the proxy increases the value of "lastkey" by one unit. Figure 3 depicts the SET query execution scenario.

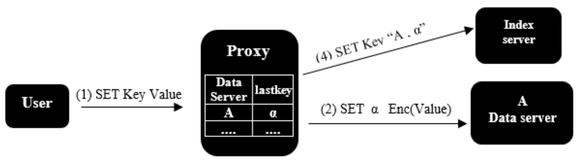


Figure 3. The sequence of "SET" query

3.3.2 *GET* Queries

The user wants to retrieve a key-value pair when the key is "Bob". He submits `GET \Bob"` to the proxy. The proxy asks the index server to send the address of the encrypted value. The index server sends `\A,100"`. Then, the proxy sends a GET query `GET \100"` to A. The data server A sends back `\Enc(pass)"` to the proxy. Finally, the proxy decrypts Enc(pass) and shows the result to the user. Figure 4 depicts the GET query execution sequence.

3.3.3 *DEL* Queries

The user wants to delete a key-value pair, where the key is "Bob". He poses `DEL \Bob"` to the proxy. The proxy executes `GET \Bob"` on the index server

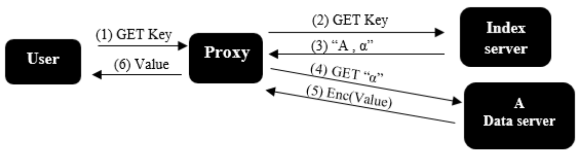


Figure 4. The sequence of "GET" query

to retrieve the address of the encrypted value corresponding to "Bob". It receives `\A, 100"` as the response. Then, the proxy sends a DEL query with the key "100" to data server A. Subsequently, data server A deletes key 100 and its corresponding value. The index server deletes pair "(Bob,100)", as well. Figure 5 depicts the DEL query execution sequence.

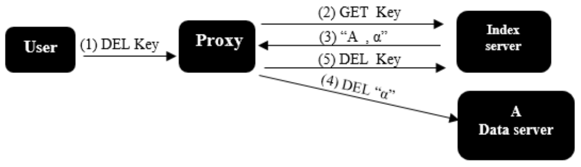


Figure 5. The sequence of "DEL" query

3.3.4 ***EXISTS*** Query

EXISTS queries are used to search on the keys and do not require receiving data from the data server. The user poses the query to the proxy and receives the proxy response from the index server. For example, if a user wants to know whether a key "Bob" exists, he sends `EXISTS \Bob"` to the proxy. Subsequently, the proxy asks the index server to check the key's existence by posing the same query `EXISTS \Bob"`. If it exists, the index server sends "1", otherwise "0", back to the proxy. Figure 6 depicts the EXISTS query execution sequence.

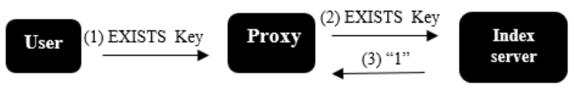


Figure 6. The sequence of "EXISTS" query

3.3.5 Other Queries

Other key-value queries can be divided into two groups, namely, composite queries and specific queries. ***Composite queries*** are queries with a combination of GET, SET, DEL, and EXISTS queries. Examples include but are not limited to storing multiple key-value pairs and storing if a key does not exist. To execute these queries, the above queries are repeated several times in a specific order. For example, the `\SETNX"` query means storing if the key does not exist. For this purpose, EXISTS query is executed first. Then, if the key is not found, a SET query is executed. ***Specific queries*** are queries for specific data types or situations. Some of these queries depend on the data type's structures; for example, a query to insert a value into the specific index of a List data type. Since SecureKV maintains the data structure in the index server, such queries are simply executed similarly to the above queries. Another example is inserting a value if there is not exist the same field in the Hash data type. This query is simply executed as well, since the index server stores fields in plain format. Some queries depend on the features of the chosen encryption method. For example, if deterministic encryption is chosen, equality queries can be executed on encrypted values, e.g., `\LINSERT"` query in a List data type insert a value after/before a specific value. If the owner chooses homomorphic encryption to encrypt values, `\INCREMENT"` queries can be executed on encrypted integer values.

4 Method Evaluation

In this section, we discuss the security and performance aspects of our system in two subsections.

4.1 Performance Evaluation

To assess the performance of SecureKV, we implemented the proxy in C# on Windows 10 x64 with a 2.3 GHz Core i5 processor and 4GB RAM. The cloud service provider is ArvanCloud[1]. We use two cloud servers with Ubuntu 20,0,4 operating system, 1 core CPU, 2GB RAM, and 50GB disk storage. Redis 5.0.7 is installed on both cloud servers. The performance tests are executed on a dataset of about three million key-value pairs. The key is a numeric string and the value is a random eight-character string. We perform experiments to assess the computation cost of SET, GET, and DEL queries. We measure the processing times of query execution in two situations of plain and encrypted values:

(1) Plain values: Plain key-value pairs stored on a cloud Redis server.
(2) Encrypted values: Keys are stored on a cloud Redis server as the index server. AES encrypted values are stored on a separate cloud Redis server as the data server.

We use StackExchange.Redis[2] library for remote connection to the Redis cloud server and ArtisanCode.SimpleAesEncryption library for AES encryption. We assess computation cost by measuring the response times of query processing in the proxy, index server, and data servers. To show query processing times more sensibly, the queries are submitted in batch files including 250, 500, 1000, and 2000 queries. Also, communication, as well as storage costs in our

[1] https://www.arvancloud.com
[2] https://github.com/StackExchange/StackExchange.Redis

approach, are theoretically evaluated based on the number of data servers.

4.1.1 Computation Cost

To measure the computation cost of query execution, we use a collection of 250 to 1000 queries to evaluate the processing times of SET, GET, and DEL queries. ***SET***: Figure 7a illustrates the response times of the collection of SET queries for two situations of Plain and Encrypted data. For the encrypted situation, we have measured both the index and the data servers times. The figure shows that the query processing time increases linearly with the increasing number of queries in the query collection. Moreover, it indicates that query processing time is similar in both the data and index servers and is nearly equal to the server time in the plain situation. We expect the same times since an equal number of SET queries are executed in the three states. Figure 7b shows the proxy response time for the collection of SET queries. It shows that the client-side part of query processing for SET is negligible compared to that of the server side, which is highly interesting in cloud-based outsourcing scenarios. Figure 8a displays the total response time of the SET query for both plain and encrypted data situations. Since in the encrypted situation, queries can be executed in parallel on the index server and the data server, the response time is similar to that of the plain situation. ***GET***: The response time of separate GET queries is similar to the collection of SET queries. Thus, Figure 8b just illustrates the total response time of GET queries for the two situations. In an encrypted data situation, it is not possible to execute queries in parallel. The data address must be retrieved first and the desired value must be retrieved based on the address. Therefore, the total response time is the response time of the index server plus the data server response time. Total response time increases linearly with the increasing number of queries in the query collection. ***DEL***: For a DEL query in the plain situation, one query is executed on the server. However, in the encrypted situation, the index server first executes a GET query to retrieve the address of the data. Then, the DEL query is executed. Figure 9a shows the response time of the index server which is the sum of GET and DEL queries. In the encrypted situation, two DEL queries in the index server and the data server are executed in parallel. Figure 9b illustrates the total response time of DEL queries in two situations of plain and encrypted data. The figure shows that the DEL query processing time in the encrypted situation is more than plain. Also, the time increases linearly with the increasing number of queries in the query collection.

4.1.2 Communication Cost

The number of queries sent from the proxy affects the communication cost. It varies in SET, GET, and DEL queries, so the communication cost is calculated separately for each query type. Table 1 shows the communication costs of SET, GET, and DEL queries. In this Table, we assume *"M"* as the volume of data exchanged between the client and cloud server in the plain situation and *"d"* as the number of data servers. ***SET***: For plain data, one query is executed for each SET query. Thus, the communication cost is *1M*. In the encrypted data situation, two queries are executed per SET query, one query on the index server and one query on the data server. Therefore, the communication cost is *2M*; *1M* for the SET query on the index server, *1M* for the SET query on the data server. That is, the communication cost is *1M* for the index server and *1M* per data server. Therefore, the communication cost of a SET query is *1M+1dM* where *1dM* is the communication overhead. ***GET***: For a GET query, the plain situation has *2M* communication cost. The communication cost in the encrypted situation is *4M*; *2M* on the index server and *2M* on the data server. Generally, the communication cost for executing a GET query is *2M+2dM* where *2dM* is the overhead. ***DEL***: The communication cost of a DEL query in the plain and encrypted data situations are *1M* and *4M*, respectively. The communication cost for executing a DEL query is *3M+1dM*. *3M* consists of *2M* to retrieve the address of data, and *1M* to delete key-value pairs on the index server. There is also *1M* communication cost per data server. Therefore, the communication overhead is *2M+1dM*.

4.1.3 Storage Cost

Suppose *"S"* as the storage cost of storing plain key-value pairs on the cloud. In our method, *1S* is the cost of the index server and, *1S* is added to the storage cost for each data server. Therefore, the storage cost is *"1S+1dS"* and storage overhead is *1dS*. The important point of this method is that the client-side storage cost is negligible.

4.2 Comparative Analysis

This section compares our method with some similar related works in terms of some performance aspects. Table 2 shows the results of our comparison. Our method is schemaless in the sense that it neither needs to change data types nor perform pre-configurations for outsourcing data. Our method, unlike other ones, is schemaless, preserves the data model after outsourcing, and uses a multi-cloud storage scenario. Also, it supports specific key-value queries while other approaches neglect to focus on key-value query processing.

Table 1. The cost of communication & storage

Cost type	Plain	Encrypted data	Overhead
$SETqueryCommunication$	$1M$	$1M + 1(1)M = 2M$	$1dM$
$GETqueryCommunication$	$2M$	$2M + 2(1)M = 4M$	$2dM$
$DELqueryCommunication$	$1M$	$3M + 1(1)M = 4M$	$2M + 1dM$
$StorageCost$	$1S$	$1S + 1(1)S = 2S$	$1dS$

Table 2. Comparative analysis

	Orginal data model	Outsourced data model	Supporting Key-value specific queries	Schemaless	Multi/Single Cloud
Secure	Key-value	Key-value	✓	✓	Multi Cloud
[19]	Tabular	Key-value	×	×	Single Cloud
[18]	No constraint	Key-value	×	×	Single Cloud
[20]	Tabular	Column-based	×	×	Single Cloud

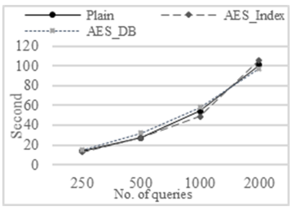
4.3 Security Aspects in the Proposed Method

In this section, we analyze the security aspects of our method. To analyze the security aspects, we assume users and the proxy are trusted entities, and both the index server and data servers are honest-but-curious. Segregating keys in the index server from values in the data servers ensures the confidentiality of the associations among keys and values. Data encryption ensures the confidentiality of stored data in the data servers and data in the move between the proxy and the servers.

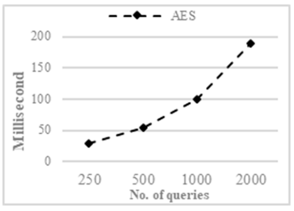
The confidentiality of data in our method relies on the strength of the used encryption algorithms. For example, if deterministic encryption is used, the data servers leak data distribution. The number of data servers and data distribution among them directly impacts access pattern confidentiality. Let us clarify it with an example. Consider Figure 4 for a GET query. When a user poses a query`"GET k1"` to the proxy, the proxy asks the index server to send the address of the encrypted value corresponding to `k1`. The index server sends the address of the desired value in a data server, e.g., `\dataSerever1,100"`. Subsequently, the proxy sends `"GET 100"` to dataSerevr1. Depending on the number of data servers and the way values are distributed among them, the index server may return different addresses on different data servers for identical queries posed by the proxy. That is, the same query `"GET k1"` may result in returning different addresses on different data servers from the proxy. That is why our method can provide access pattern confidentiality proportional to the number of data servers.

5 Conclusion

In this paper, we proposed a method called SecureKV, a key-value database framework that deals with the challenges of secure search on outsourced key-value databases. Separate cloud servers in a multi-cloud scenario have been used to provide data confidentiality through encrypting values detached from their corresponding keys. The SecureKV prototype evaluation indicates that increasing the query response time in the encrypted data situation is similar to the plain situation. Moreover, the evaluation results indicate that the client-side computation cost is negligible versus the server-side, which is highly interesting in cloud-based scenarios. However, communication and storage costs increase proportionally to the number of data servers. On the other hand, redundant data servers provide access pattern confidentiality. We plan to extend our work to address data integrity concerns. Also, we would like to investigate more on security, cost, and efficiency trade-offs in our method.

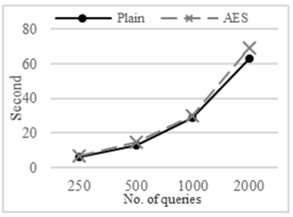


(a) Server time for SET query

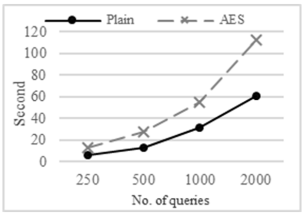


(b) Proxy time for SET query

Figure 7. Server time and Proxy time for SET query



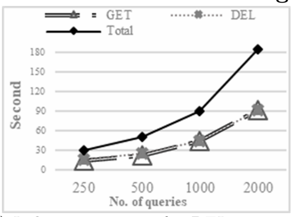
(a) Total time for SET query



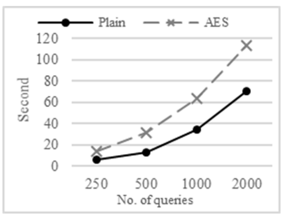
(b) Total time for GET query

Figure 8. Total time for SET query and GET query

Figure 9.



(a) Index server time for DEL query on encrypted data



(b) Total time for DEL query

Figure 10. Index server time and Total time for DEL query

References

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

[2] Pierangela Samarati, S De Capitani di Vimer-

cati, S Murugesan, and I Bojanova. Cloud security: Issues and concerns. *Encyclopedia on cloud computing*, pages 1–14, 2016.

[3] Mohammad Ali Hadavi, Rasool Jalili, Ernesto Damiani, and Stelvio Cimato. Security and searchability in secret sharing-based data outsourcing. *International Journal of Information Security*, 14(6):513–529, 2015.

[4] Jason W Woodworth and Mohsen Amini Salehi. S3bd: Secure semantic search over encrypted big data in the cloud. *Concurrency and Computation: Practice and Experience*, 31(11):e5050, 2019.

[5] Lanxiang Chen, Nan Zhang, Hung-Min Sun, Chin-Chen Chang, Shui Yu, and Kim-Kwang Raymond Choo. Secure search for encrypted personal health records from big data nosql databases in cloud. *Computing*, 102(6):1521–1545, 2020.

[6] Mamdouh Alenezi, Muhammad Usama, Khaled Almustafa, Waheed Iqbal, Muhammad Ali Raza, and Tanveer Khan. An efficient, secure, and queryable encryption for nosql-based databases hosted on untrusted cloud environments. *International Journal of Information Security and Privacy (IJISP)*, 13(2):14–31, 2019.

[7] Karamjit Kaur and Rinkle Rani. Modeling and querying data in nosql databases. In *2013 IEEE international conference on big data*, pages 1–7. IEEE, 2013.

[8] Ming-Hung Shih and J Morris Chang. Design and analysis of high performance crypt-nosql. In *2017 IEEE Conference on Dependable and Secure Computing*, pages 52–59. IEEE, 2017.

[9] Abdulatif Alabdulatif, Ibrahim Khalil, and Xun Yi. Towards secure big data analytic for cloud-enabled applications with fully homomorphic encryption. *Journal of Parallel and Distributed Computing*, 137:192–204, 2020.

[10] Viswanath Gudditti and P Venkata Krishna. Light weight encryption model for map reduce layer to preserve security in the big data and cloud. *Materials Today: Proceedings*, 2021.

[11] Divyakant Agrawal, Amr El Abbadi, Fatih Emekci, and Ahmed Metwally. Database management as a service: Challenges and opportunities. In *2009 IEEE 25th International Conference on Data Engineering*, pages 1709–1716. IEEE, 2009.

[12] Varunya Attasena, Jérôme Darmont, and Nouria Harbi. Secret sharing for cloud data security: a survey. *The VLDB Journal*, 26(5):657–681, 2017.

[13] G Viswanath and P Venkata Krishna. Hybrid encryption framework for securing big data storage in multi-cloud environment. *Evolutionary Intelligence*, 14(2):691–698, 2021.

[14] Gunasekaran Manogaran, Chandu Thota, and M Vijay Kumar. Metaclouddatastorage architecture for big data security in cloud computing. *Procedia Computer Science*, 87:128–133, 2016.

[15] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. Arx: an encrypted database using semantically secure encryption. *Cryptology ePrint Archive*, 2016.

[16] Gabriela Suntaxi, Aboubakr Achraf El Ghazi, and Klemens Böhm. Secrecy and performance models for query processing on outsourced graph data. *Distributed and Parallel Databases*, 39(1):35–77, 2021.

[17] Muhammad Ali Raza, Muhammad Usama, Waheed Iqbal, and Faisal Bukhari. Secure nosql over cloud using data decomposition and queryable encryption. In *International Conference on Intelligent Technologies and Applications*, pages 409–421. Springer, 2019.

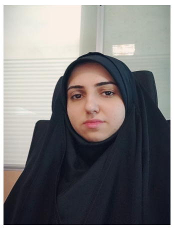
[18] Xingliang Yuan, Xinyu Wang, Cong Wang, Chen Qian, and Jianxiong Lin. Building an encrypted, distributed, and searchable key-value store. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 547–558, 2016.

[19] Xingliang Yuan, Yu Guo, Xinyu Wang, Cong Wang, Baochun Li, and Xiaohua Jia. Enckv: An encrypted key-value store with rich queries. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 423–435, 2017.

[20] Meng Zhang, Saiyu Qi, Meixia Miao, and Fuyou Zhang. Enabling compressed encryption for cloud based big data stores. In *International Conference on Cryptology and Network Security*, pages 270–287. Springer, 2019.

[21] Asadulla Khan Zaki and M Indiramma. A novel redis security extension for nosql database using authentication and encryption. In *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–6. IEEE, 2015.

[22] Rogério Pontes, Francisco Maia, Ricardo Vilaça, and Nuno Machado. d'artagnan: A trusted nosql database on untrusted clouds. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, pages 61–6109. IEEE, 2019.



Maryam Saeedi Sadr received her M.Sc. and B.Sc. degrees in Computer Engineering from Malek Ashtar University of Technology, Tehran, Iran, in 2022, and from Shariati Technical and Vocational College, Tehran, Iran, in 2019, respectively.

Mohammad-Ali Hadavi received his Ph.D. degree in Computer Engineering from the Sharif University of Technology, Tehran, Iran, in 2015. He received his M.Sc. and B.Sc. degrees in Software Engineering from Amirkabir University of Technology, Tehran, Iran, in 2004, and from Ferdowsi University of Mashhad, Iran, in 2002, respectively. Now, he is an assistant professor at the Malek Ashtar University of Technology. Focusing on information security, he has published more than 50 papers in national and international journals and conference proceedings. His research interests include software security, security measurement, database security, cloud security, and security aspects of data outsourcing.