The Knowledge Engineering Review, Vol. 35, e20, 1 of 12. © The Author(s) 2020. Published by Cambridge University Press.
doi:10.1017/S0269888920000302

A blockchain-based database management system

JEYAKUMAR SAMANTHA THARANI[1], MUKUNTHAN THARMAKULASINGAM[2], and VALLIPURAM MUTHUKKUMARASAMY[3]

[1] *Department of Computer Science, University of Jaffna, Sri Lanka*
e-mail: *samantha@univ.jfn.ac.lk*
[2] *Department of Electrical and Electronic Engineering, University of Jaffna, Sri Lanka*
e-mail: *mukunthan@eng.jfn.ac.lk*
[3] *School of Information & Communication Technology, Griffith University Gold Coast Campus*
e-mail: *v.muthu@griffith.edu.au*

Abstract

The software and hardware applications are clearly on the way of becoming an integral tool of business, communication and popular culture in many parts of the world. People are interacting with the environment via the Internet to perform physical activities remotely. These applications are hosted in the public or private servers under the control of the server admin. The users' online usage data can be stored in public or private cloud platforms, used for processing and monitoring users' online behaviour and emotional factors and shared with third parties to facilitate making their business decisions. When users allow their data to be collected via software applications and mobile devices, users need to have some level of trust and control over their data. But, software applications or mobile devices connected to the cloud server using client–server architecture does not ensure the reliability, security and integrity among their data. To get over these kinds of limitations, we propose a database management system using blockchain technology that can be used by any software applications. The blockchain database connected to the cloud server can be used to increase the trustfulness of the application. `Blockchain` has the capability to provide decentralization, immutability and owner-controlled digital assets among software applications. Since users can save their data in a shared transaction repository with tamper-resistant records, it enables related parties to access and control users' data without the need for a central control system.

1 Introduction

Software and hardware applications are clearly of the way on becoming an integral tool of business, communication and popular culture in many parts of the world. People are interacting with the environment via the Internet to perform physical activities remotely. For example, hardware applications and wearable computing devices are frequently used in the areas of health, education, reservation, sports, entertainment, management and controlling of resources (abbasi2017addressing). Wearable computing devices are being utilized to monitor blood pressure, heart rate and predict different diseases by using `Computer Vision` and `Artificial Intelligence` in the healthcare field. Yet, there are multiple challenges in enabling this kind of technologies related to data integrity, data heterogeneity, knowledge management and data analysis tools. Data stored in the cloud can be vulnerable to tampering. The adoption of the technology has also been hampered by the necessity to ensure the ability to stop data temper and corruption in the data flowing of software applications potentially due to accidents on industrial scales. To solve these well-known issues, a technology that can be used to trace the transaction of data from the source to destination is needed. Blockchain technology is a more suitable integration platform for decentralization, immutability and owner-controlled asset. Industries are currently using blockchain technology in supply chain management, which helps to track the objects as they traverse the export/import supply chain while enforcing shipping and expediting incremental payments (staples2017risks). Our proposed database management system is used to track the database operations such as CREATE, UPDATE, READ, etc. for the applications hosted in the cloud platforms. When an application requests a database operation, this database management system will record in the distributed ledger by using that an application can ensure their data integrity. The data in the database cannot be altered or modified by an unauthorized person.

2 Related work

According to the previous literature studies, blockchain technologies were proposed to ensure decentralizing privacy among users personal data collected via social media's mobile applications such as Facebook, WhatsApp, Viber, Messenger (Zyskind *et al.*, 2015). In their proposed algorithm, they have included three entities such as mobile phone users, service providers and nodes. The proposed blockchain allowed two types of transactions, namely, T_{access} and T_{data}. When a user signed up for the first time, a new shared identity key was generated and sent along with the associated permissions to the blockchain via T_{access} transaction. By using that shared key, the collected data were encrypted and sent to the blockchain via T_{data} transaction, which subsequently routed it to an `off-blockchain` key-value store, while retaining only a pointer to the data on the public ledger (*SHA-256 hash* of the data). Both the service and the mobile phone user could query the data using a T_{data} transaction with the pointer (key) associated with it. The blockchain verified that the digital signature belonged to either the mobile phone user or the service. For the service, it would give permissions to access and check the status of the data. The user could change the permission granted to the service at any time by issuing a T_{access} transaction with a new set of permissions, including revoking access to previously stored data. In this way, they had made users aware of their data and how it was used by the services. In addition to that, this proposed blockchain considered the users as the owners for their personal data. They can decide which services can access their data by granting or rejecting the permissions via T_{access} transaction.

In another study, Blockchain-based database aimed at providing a replicated database (Gaetani *et al.*, 2017). In this proposed method, they devised the blockchain into two layers. The first layer ensured adequate performance, while the second layer ensured strong integrity quarantines. The first layer employed a lightweight distributed consensus protocol that assumed low latency and high throughput and provided weak data integrity guarantees due to the lack of `Proof of Work(PoW)`. Thus, second layer was designed as a PoW-based blockchain that stored evidence of the database operations logged by the first layer. This was a guarantee for strong data integrity, but with poor performance. The operations were first logged via appropriate evidence of the first-layer blockchain, then they were executed on the distributed DB replicas. The first-layer blockchain was related to permission, and it was proposed to have one miner node on each number of clouds in the future. The miner nodes were relying on the public/private key pair to sign messages and achieve consensus. One miner would be selected as a leader in each round based on the time they had taken to achieve consensus. The leader was the in charge for mining new operations, signing them with its private key and broadcasting them to other miners. Once all miners had signed the operation, they became a part of the blockchain. After that, all miners added those operations into their local ledger and applied them to their local replica. The interaction with the second-layer PoW-based blockchain was released via blockchain anchoring technique. This technique was a timed operation that permits linking a specific first-layer blockchain with the second-layer blockchain. At a certain interval of time, a witness transaction contained the hash of the first-layer blockchain up to the current operation. This hash value was sent to the second-layer blockchain and consequently stored as an immutable irreversible transaction. Hashes acted as forensic evidence for proving and validating the integrity of the data stored in the first-layer blockchain. The proposed blockchain had three entities, namely, Data Owner Application (DOA), Data Consumer Applications (DCAs) and Cloud Storage Service (CSS). DOA and DCAs had an agreement via smart contract. This contract was used by DCAs to verify the data integrity

https://doi.org/10.1017/S0269888920000302 Published online by Cambridge University Press

of data owned by DOA stored in CSS. In addition to that, DOA and CSS had another contract which controlled the CSS to shared data only with the signed DCAs.

Bigchain is a database (McConaghy *et al*., 2016) known as the blockchain database since it has the database properties as well as some blockchain properties. It uses Tendermint (Kwon, 2014) as a protocol for all networking and consensus activities. Each node in the Bigchain database has its own MongoDB database. If local MongoDB is attacked by a malicious hacker in any one of the nodes, other MongoDBs in other nodes will not be affected. Bigchain database gets decentralized property due to this feature. To achieve immutability, it does not provide APIs to change or erase stored data and cryptographic signs all transactions after a transaction is stored. Changing its contents will change the signature, which can be detected. The only owner of an asset is allowed to transfer that asset, not even a node operator can transfer an asset. When a node creates an asset, it needs to cryptographically sign using its private key. When a transaction reached the Tendermint node, it first checks the transaction. If it is a success, then it will be included in the member broadcast to other peers, and eventually included in a block. There are several use cases including supply chain, IP rights management, digital twins and IoT, data governance and immutable audit trails.

ChainSQL (Muzammal *et al*., 2018) is a blockchain database application platform. It is developed by integrating blockchain with the traditional database. Chain SQL has three main concepts such as blockchain networks, a database and a set of users. The database is configured on top of the blockchain nodes which is synchronized with the blockchain and facilitates quick database operations. The transaction received from the user is authenticated by the subset of nodes in the blockchain network. Authenticated transaction sent to the blockchain network for consensus and is subsequently written into the corresponding database. If consensus is failed, the database operation is rolled back. In addition to that, it has a data recovery facility. Since one of the nodes in the blockchain network is configured with the database to keep transactions in the blockchain or to execute database operations to recreate a new table. A node on the blockchain network can be either a full-record node which can store all the transactions on the blockchain network or a partial-record node.

This research aims to take benefits from all these state-of-art works and contributions to derive a well-defined solution for the software or hardware application user's shared data which are available in the cloud platforms. We are mainly focusing on data management and data sharing for software applications and mobile devices which are interconnected with cloud platforms. In the next section, we will discuss our proposed system architecture and its functionalities.

3 Proposed system

In this section, we introduce our proposed method for an effective blockchain-based database management system. It can store evidence of database operations that cannot be repudiated. Figure 1 graphically depicts the proposed database management system distributed in the cloud server. This proposed database management system considers three entities, namely, blockchain-based cloud server, software application and an off-chain. Once a software application joins in the block chain-based cloud server, it has a database and a distributed ledger. Database details of the application such as the number of tables, name of the table, name of column and so on can be stored and the authorized data access operations can be mentioned in the `smart contract` of the application. When an application sends a request to store/view/update the value of the data in the cloud server, it will be stored in the distributed ledger and will be checked with the smart contract of the application. If the data access request is a success, it will be notified to the application and a hash pointer will be sent to the off-chain blockchain in the second layer.

The use of this database management is not only providing data integrity but also fully distributed control of data in the databases. As an external client cannot tamper the records without the owner's permission and only authorized client has permission to access the data, we can ensure the data integrity and owner-controlled digital asset. The client can identify the interested parties and the usage of their data through history. This will help to identify the future markets demands and new features of their applications.

https://doi.org/10.1017/S0269888920000302 Published online by Cambridge University Press

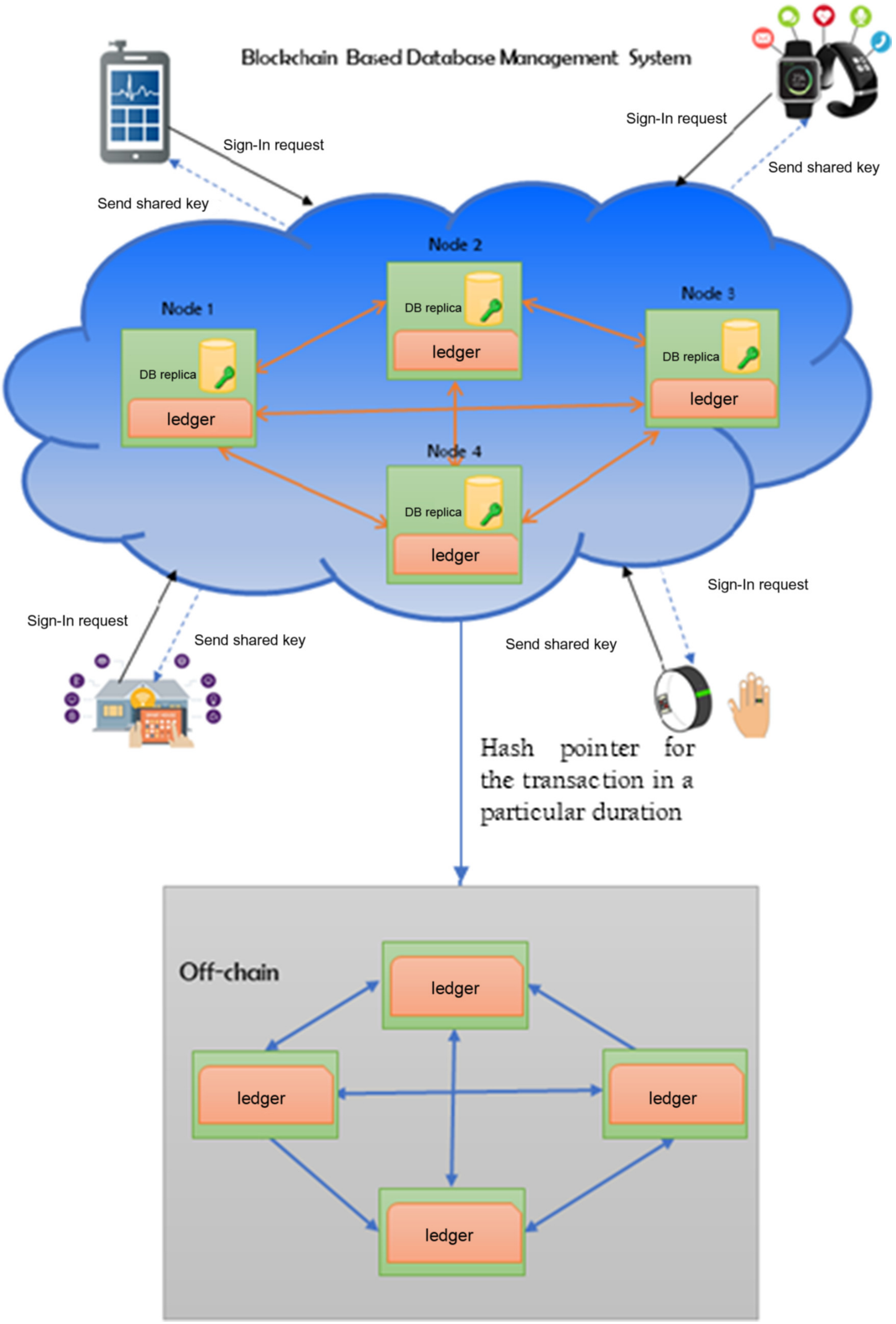


Figure 1 Blockchain-based database management system in cloud server

4 Design of the proposed system

4.1 Defining the scenario

The proposed blockchain-based cloud server consists of two layers, namely, on-chain and off-chain. On-chain is a blockchain-based cloud server. It contains nodes for each application. Each node of the application has a database replica and a distributed ledger. Nodes are interconnected with each other in

https://doi.org/10.1017/S0269888920000302 Published online by Cambridge University Press

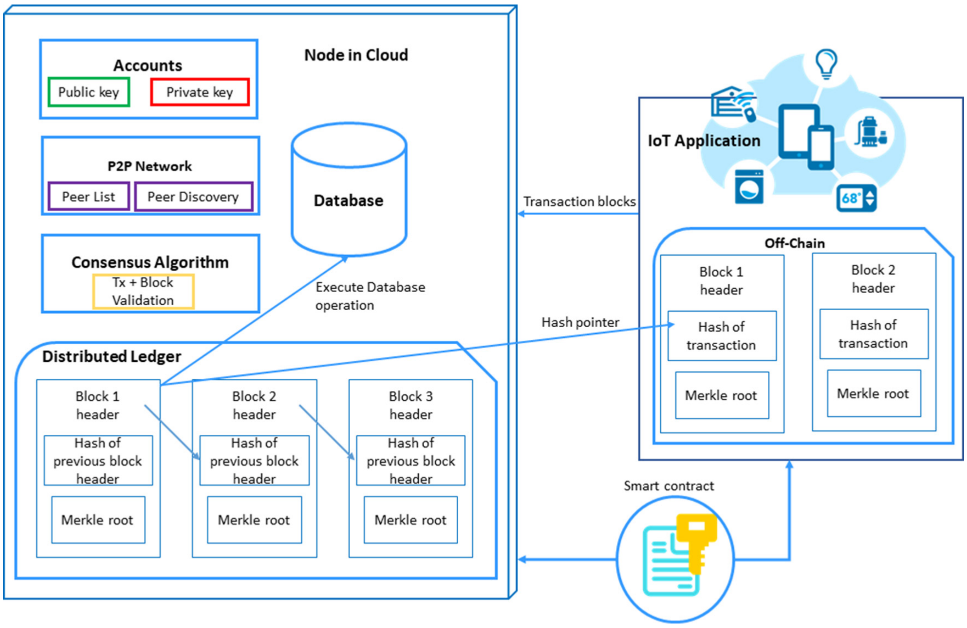


Figure 2 Internal structure of the node in the cloud server and software application

a peer-to-peer manner. Hence, all database operations such as INSERT, UPDATE, READ are visible to each other.

At the beginning, each application needs to register in the cloud server by using Algorithm 1 protocol. Once they are registered, a shared key (public, private) is generated for them, and the shared key will be stored in the distributed ledger. The shared key is used by the nodes to identify other nodes during the transaction.

Once the initial process is done, an application can sent a database operation request as an INSERT query for storing data in the database or a READ query to retrieve data from the database or an UPDATE query for the modification. All these operations and their execution status (SUCCESS, FAILED) will be stored in the distributed ledger with the timestamp and signed with *miner's* public key.

At a time, one node in the blockchain can be selected as a miner and that node will only have the permission to receive the request Algorithm 2 from the application. Once a request is received by the miner node, it will sign the request with its private key and broadcast to other nodes. After that, all nodes will check the validity of the request. Once it is authorized, the operation will be recorded in the distributed ledger with the timestamp and the miner's public key. If the request is rejected, that request will be recorded as FAILED in the distributed ledger with the sender's public key, and the application which made the request will be notified with the status. A hash pointer will be created for the accepted and rejected requests and send to the second layer of the blockchain.

The second layer of the proposed blockchain is an off-chain. Off-chain does not necessarily mean 'not on the blockchain', it means that it is not on a publicly accessible service. Based on the response available in the hash pointer, an application performs the requested action in the appropriate database.

Within the certain interval of time, a hash pointer is created and send to the second layer off-chain. Time can be configured as either per week or month or year by the owner of the software application. Hash pointer is stored in the off-chain ledger. By using that, a node can control or monitor the data accessing or sharing within a certain interval of time.

4.2 Defining the internal structure of the node and the software application

Figure 2 describes the internal structure of the node in the cloud server and the internal structure of the software application. Each software application owns a node. Node has two main components such as database and distributed ledger. The distributed ledger holds all history of the database operations

https://doi.org/10.1017/S0269888920000302 Published online by Cambridge University Press

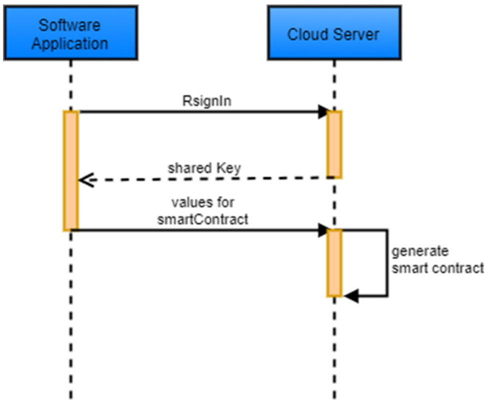


Figure 3 Sequence diagram for Rsign-in request

and its status. Based on the status of the database operation, a node can perform the operations such as SELECT, CREATE and UPDATE on its database. `Smart contracts` define which database operations are allowed in which columns of each table. In addition to that, a node has an account to hold private and public keys, and nodes link with other nodes in a `peer-to-peer` (P2P) communication with its own layer of protocol messages for node communication and peer discovery. It uses a consensus algorithm to validate the transactions from other pair nodes.

A software application has an off-chain distributed ledger which is a key store for the database operation with a time stamp. Each database operations has a hash pointer and stored in this off-chain distributed ledger. In addition to that, there is a smart contract between the node and software application. It is a digital agreement for the data access control for one node and its pair nodes.

4.3 *Defining the operations*

Before describing our protocol, we are first defining some notations that will be later used in the paper:

pk_{iapp} = Private key for the software application
sk_{iapp} = Shared key or Public key for the software application
DL = Distributed Ledger
hp = Hash pointer
Tr = Transaction
Ltbls = List of Tables
tblNm = Table Name
clmNm = Column Name
opts = operations
Lclms = List of Columns
clms = columns

1. $R_{\text{sign-in}}$

Software application sends a Algorithm 1 request to cloud server as shown in Figure 3. In the sign-in, an application needs to provide the shared key(public key), App Id (auto generated), Database Name, Number of tables and its clients' address(public key).

2. R_{access}

When a software application performs a Algorithm 2 as shown in the Figure 4, it is needed to provide receiver's address, app Id, tables name, column name and the operations to be performed in the given column. These are sent along with its public key and timestamp.

https://doi.org/10.1017/S0269888920000302 Published online by Cambridge University Press

Algorithm 1 $R_{\text{sign}-\text{in}}$

```
Input:  address, appId, databaseName, NoOfTables, clientAddresses
Output:  Registered transaction Id
if checkAddress(address) then
    Registered as new application;
    Store new app in the application list;
else
    notified as "Already registered app";
```

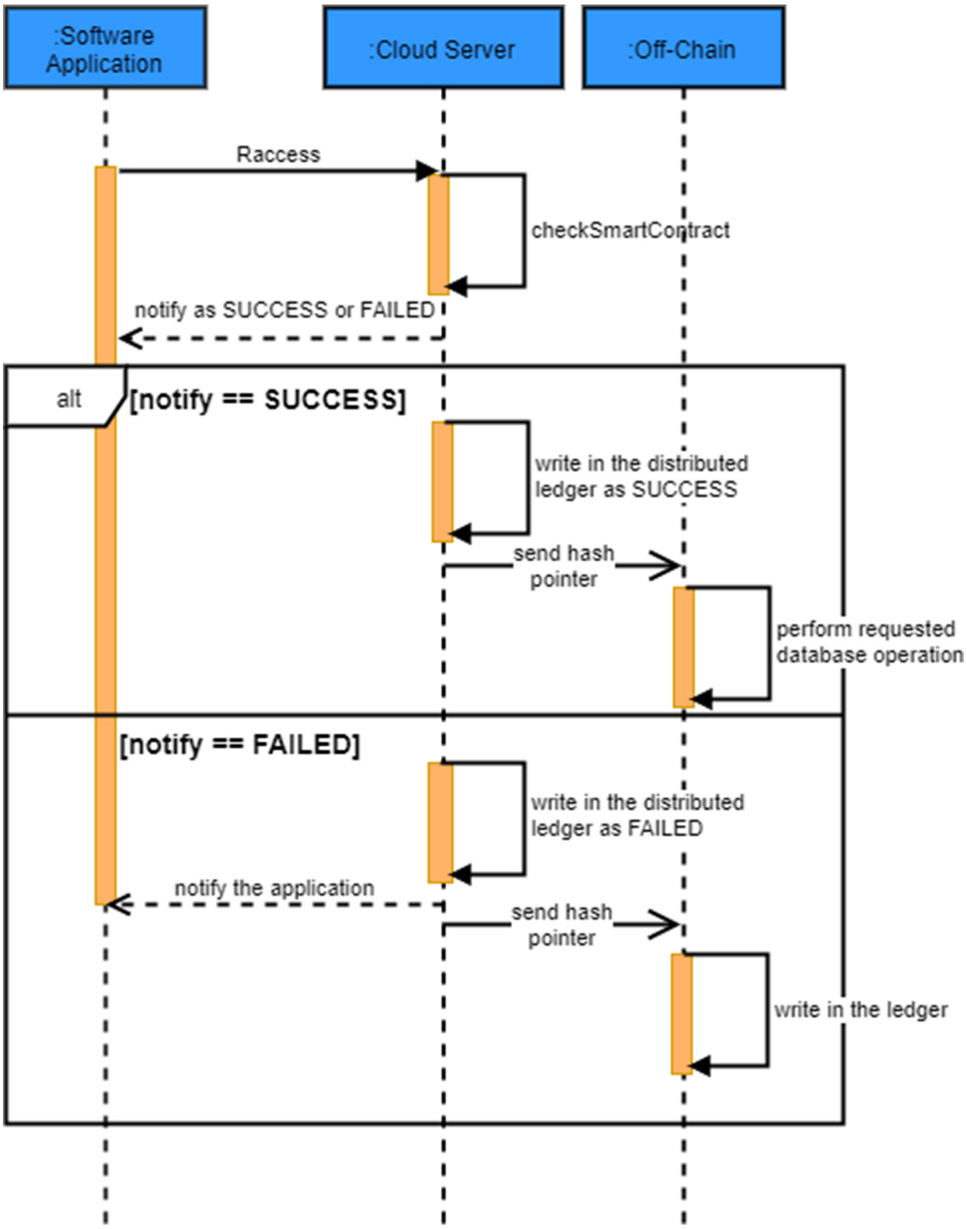


Figure 4 Sequence diagram for Raccess request

https://doi.org/10.1017/S0269888920000302 Published online by Cambridge University Press

Algorithm 2 R_{access}

```
Input: receiver app address, appId, tableName, columnName, columnOperations
Output: appId, tableName, columnName, columnOperations
if validateTransaction(address, appId, tblNm, clmNm, opts) then
    return (appId, tblNm, clmNm, opts)
else
    notified as "Unauthorized data access request";
```

Algorithm 3 Validate Request

```
Input: receiver app address, appId, tableName, columnName, columnOperations
Output: TRUE or FALSE
if checkAddress(address) then
    Get the list of tables of the app using appId;
    for each table t in tbls do
        if t.Name is equal to tableName then
            Get the list of columns of the table using tableName;
            for each column c in clms do
                if c.Name is equal to columnName then
                    Get the authorized column's operations;
                    for each opt o in opts do
                        if o equal to columnOperation then Return TRUE; ;
                        else Return FALSE; ;
```

3. Check the validity of data access request

   The Algorithm 3 can be checked using the smart contract.

4. Update the data operations of the application

   The Algorithm 4 is used to update the existing data operations based on the future modification.

4.4 *Defining the smart contract*

A smart contract is created for software application and cloud server. In that smart contact, it needs to be defined about the allowed database operations, allowed tables and columns for those operations.

4.5 *Defining off-chain implementation*

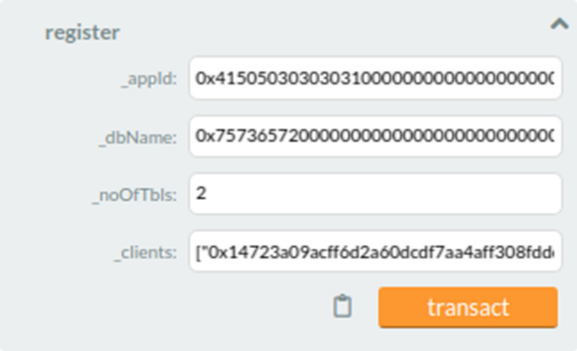
Off-chain is used to store the logs of the database operations with a certain time interval gap. When a modification is made in the database of the software application, it will be hashed by the miner and notified to the application by sending the hash pointer. It will be stored in the off-chain ledger. When an application receives the hash pointer, it can query by using that pointer, view the status of its data and identify their targeted audience.

In the next section, we will explain how our proposed system overcomes the existing challenges in database management and data sharing among users and software applications.

https://doi.org/10.1017/S0269888920000302 Published online by Cambridge University Press

Algorithm 4 Update Operation

```
Input:  receiver app address, appId, tableName, columnName, oldOperation,
        newOperation
Output:  Modified Application Operation;
if checkAddress(address) then
    Get the list of tables of the app using appId;
    for each table t in tbls do
        if $t.Name$ is equal to $tableName$ then
            Get the list of columns of the table using tableName;
            for each column c in clms do
                if $c.Name$ is equal to $columnName$ then
                    Get the authorized column’s operations;
                    for each opt o in opts do
                        if o equal to oldOperation then
                            $o \leftarrow new\ Operation$ ;
    Return modified application operation;
```



register

_appId: 0x41505030303031000000000000000000000

_dbName: 0x75736572000000000000000000000000000

_noOfTbls: 2

_clients: ["0x14723a09acff6d2a60dcdf7aa4aff308fdd

Figure 5 Register the software application

5 Implementation details

A blockchain-based cloud server is implemented in `Ethereum` platform which is an open-source distributed public blockchain network (Wood, 2014). Like other blockchains, Ethereum has a native cryptocurrency called `Ether` (ETH). ETH is digital money which has many of the same features as the well-known `Bitcoin` (Nakamoto, 2008). It is purely digital and can be sent to anyone anywhere in the world instantly. It allows decentralized apps to be built on it with the help of smart contract functionality.

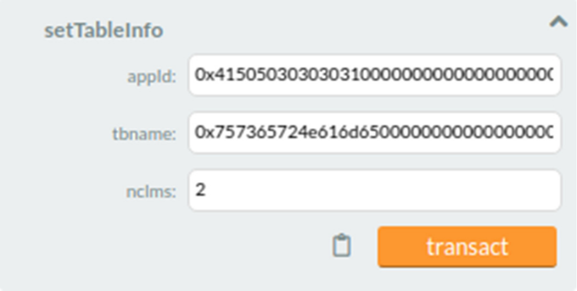
For simplicity, a smart contract called DbContract is designed and implemented with three major functions. One function `register` is used to register software application such as application Id, database name, number of tables and the clients’ addresses. Figure 5 shows the initial configuration of the software application in *Remix IDE*.

Once the application is registered, it is needed to configure its tables and columns details. Other function `setTableInfo` is used to configure the property of table for an application, and the other `setColumnInfo` function is used to configure the columns details of the table. Figure 6 illustrates the configuration of the table of the software application. In that, we need to give appId, table name and the number of columns in that table. Figure 7 illustrates the configuration of the columns of table related to the software application. In that, we need to give appId, table name, column name and the allowed data operations can be performed that column. Table 1 shows the Transaction cost and Execution cost for the operations register, `setTableInfo` and `setColumnInfo`.

https://doi.org/10.1017/S0269888920000302 Published online by Cambridge University Press

Table 1 Resource usage for the operations in smart contract

Operation	Transaction cost	Execution cost
register	168955 gas	143331 gas
setTableInfo	164558 gas	141878 gas
setColumnInfo	257513 gas	232337 gas

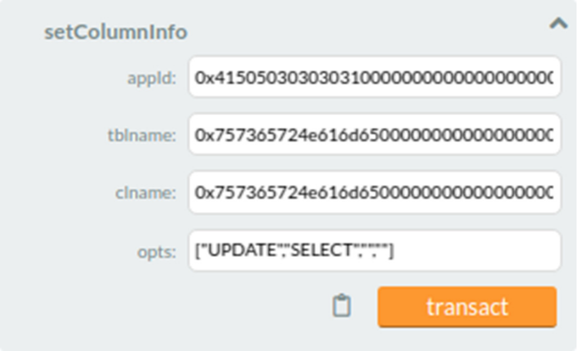

setTableInfo

appId: 0x415050303030310000000000000000000000

tbname: 0x757365724e616d65000000000000000000

nclms: 2

Figure 6 Configure table properties


setColumnInfo

appId: 0x415050303030310000000000000000000000

tblname: 0x757365724e616d65000000000000000000

clname: 0x757365724e616d65000000000000000000

opts: ["UPDATE","SELECT","",""]

Figure 7 Configure column properties

6 Analysis and discussion

This proposed blockchain-based database management system contains two layers to ensure the key features such as secure communication, data integrity, privacy, query processing and data availability. Every participating node has a database replica and a copy of the distributed ledger. Software applications can perform INSERT, UPDATE, READ database operations based on the access permissions mentioned in the smart contract.

6.1 Secure communication

One of the main security challenges for software application or mobile device is to ensure secure communication across the network with cloud platforms and third-party applications. This blockchain-based database management system ensures the authentication via public-key cryptography encryption and the smart contract. If an application has an authorized key, it can INSERT, UPDATE, READ data from the databases. In this way, access can be controlled over data of the application.

6.2 Efficiency

In the real world, most of the mobile devices have less memory capacity since they could not act as a node in blockchain. Off-chain transactions bring huge value as they are increasing the security and also

https://doi.org/10.1017/S0269888920000302 Published online by Cambridge University Press

not bound by the transactional speed limitations that on-chain transactions have. In a typical on-chain transaction, each transaction needs to be confirmed by all nodes. Thus, this makes it very slow, whereas an off-chain transaction does not need to wait for all the nodes to confirm the transaction before it is marked as complete or successful. As proposed blockchain-based database management system is going to be deal with the real-time software applications, the performance of the application should be acceptable by the user. By incorporating the off-chain layer, it is possible to perform users' request as much as fast users need. Another important factor of the off-chain system is that it can easily store any type of real sensitive data. As on-chain data are nonnegotiable in terms of modification, this sort of problem does not arise with the off-chain data management system.

6.3 Ensure data privacy and integrity

Once data are transmitted across the network, it needs to be stored and processed securely. Threats to data integrity and privacy are high as tampering with data may maliciously affect crucial business decisions. This threat is very high in cloud environments, where data owners cannot control fundamental data aspects such as physical storage of data and the control of its accesses. But, this blockchain-based database management system can ensure data integrity in cloud computing environments. By using this cloud server, software applications can monitor or track their database status (insert, delete, update). If the data in a database has tampered, the blockchain-based distributed method can easily identify and ignore. This blockchain-based database management system targets at providing a replicated database which integrity is testified via adequate evidence stored on a two-layered (on-chain with off-chain) blockchain system. Off-chain assures the performance with lightweight consensus process and on-chain assures integrity with proper consensus mechanism. Other than that, a hash mechanism is used to link all transactions within a timestamp with the off-chain distributed ledger. Hence, it's highly prevented to perform the same operation multiple time in a node. In this way, this system is ensuring data privacy and integrity.

6.4 Secure web, mobile and cloud applications

Web, mobile and cloud apps and services are used to manage, access and process data. In this cloud server, app services need to have a smart contract with a particular application before getting permission to access their data. When they access the data, a track of the record is kept in distributed ledger with the timestamp. Therefore, it facilitates secured Web, Mobile and Cloud Applications.

6.5 Detect vulnerabilities and incidents

The complexity of the system, in terms of the number of devices connected, and the variety of devices, apps, services and communication protocols involved can make it difficult to identify the time of an incident occurred in large-scale IoT systems. By using the records in the distributed ledger, time of those incidents can be easily found.

6.6 Ensure high availability

Software application developers must consider the availability of user's data in the web and mobile apps. By using the hash pointer within a particular timestamp, the user can check which applications or devices are accessing their data and they can make sure whether the data are outdated or not.

7 Conclusion

The objective of this proposed database management system was to consider the possibility of using blockchain technology in data access protection IoT applications, mainly for wearable devices with the protection of personal medical information collected via medical sensors and environmental sensors in the smart home and smart agriculture applications when they share their data via the cloud.

https://doi.org/10.1017/S0269888920000302 Published online by Cambridge University Press

We have proposed architecture of the solution to protect, control and monitor the data of the IoT applications. By using the smart contract, IoT applications can control the database operations among its database tables. By using the hash pointer existing in the distributed ledger of off-chain, it can monitor the status of its data within a time interval. It will help the application to identify the useful data via that and it can find the target clients, enhancing the existing feature and add new features.

References

Gaetani, E., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A. & Sassone, V. 2017. Blockchain-based database to ensure data integrity in cloud computing environments.

Kwon, J. 2014. Tendermint: Consensus without mining. Draft v. 0.6, fall.

McConaghy, T., Marques, R., Müller, A., De Jonghe, D., McConaghy, T., McMullen, G., Henderson, R., Bellemare, S. & Granzotto, A. 2016. BigchainDB: A scalable blockchain database. White Paper, BigChainDB.

Muzammal, M., Qu, Q. & Nasrulin, B. 2018. *A Blockchain Database Application Platform.* arXiv preprint arXiv:1808.05199.

Nakamoto, S. 2008. *Bitcoin: A Peer-to-Peer Electronic Cash System*, http://bitcoin.org/bitcoin.pdf.

Wood, G. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151, 1–32.

Zyskind, G., Nathan, O. *et al.* 2015. Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops*, 180–184. IEEE.

https://doi.org/10.1017/S0269888920000302 Published online by Cambridge University Press