Data Tamper Detection from NoSQL Database in Forensic Environment

Rupali Chopade* and Vinod Pachghare

Department of Computer Engineering and IT, College of Engineering Pune, Savitribai Phule Pune University, India
E-mail: rmc18.comp@coep.ac.in; vkp.comp@coep.ac.in
**Corresponding Author*

Received 29 September 2020; Accepted 16 December 2020;
Publication 29 March 2021

Abstract

The growth of service sector is increasing the usage of digital applications worldwide. These digital applications are making use of database to store the sensitive and secret information. As the database has distributed over the internet, cybercrime attackers may tamper the database to attack on such sensitive and confidential information. In such scenario, maintaining the integrity of database is a big challenge. Database tampering will change the database state by any data manipulation operation like insert, update or delete. Tamper detection techniques are useful for the detection of such data tampering which play an important role in database forensic investigation process. Use of NoSQL database has been attracted by big data requirements. Previous research work has limited to tamper detection in relational database and very less work has been found in NoSQL database. So there is a need to propose a mechanism to detect the tampering of NoSQL database systems. Whereas this article proposes an idea of tamper detection in NoSQL database such as MongoDB and Cassandra, which are widely used document-oriented and column-based NoSQL database respectively. This research work has proposed tamper detection technique which works in forensic

Journal of Cyber Security and Mobility, Vol. 10_2, 421–450.
doi: 10.13052/jcsm2245-1439.1025
© 2021 River Publishers

environment to give more relevant outcome on data tampering with hundred percent accurate result and distinguish between suspicious and genuine tampering.

Keywords: Database, database forensics, MongoDB, cassandra, NoSQL, tamper detection.

1 Introduction

Now a days because of automation in every sector, organizations are storing their data in electronic format. This electronic data has stored in the form of a database. Database is an easy way of managing and accessing the data [1]. Security of database is always a big concern for every organization. Database security bounds for Confidentiality, Integrity & Availability (CIA) and these trio plays an important role in information security. Database crimes are mainly affecting on CIA. Integrity is one of the major pillars of the database. Database integrity is useful to identify any change in the database. Any alteration in the database is called as database tampering. Database tampering, addresses the problem of determining who, when and what data has been tampered [2]. This article addresses for when and what data has tampered, which may be useful for database forensic studies. Database forensic is one of the sub-areas of digital forensics. Database forensics involves a detailed study of database and their associated artifacts [3]. In database forensics, there is a need to identify and trace the evidence against such crimes and divulge the affected database. This article presents a technique to detect the tamper in MongoDB and Cassandra, which are one of the popular NoSQL databases. Researchers have already worked for tamper detection in relational databases like Oracle, MySQL, and SQL Server, etc. Different tools are available for forensics purpose but each tool has its own limitations [4]. These tools are designed for specific database, considering its internal architecture and most of these tools are available for relational databases. NoSQL databases are getting widely adopted due to huge data storage requirements and unstructured data formats. NoSQL database falls in four categories namely document-based, key-value based, column-based, and graph-based. MongoDB is a document-oriented, while Cassandra is a column-based NoSQL database. The reason behind selecting these two databases, is their top ranking in the NoSQL database [5]. As every NoSQL database has its own storage engine and internal data storage format, hence this work has limited to these two databases. Before designing and proposing this tamper detection technique for these databases, a survey was performed on NoSQL databases by focusing on their working and internal data storage formats. The observations presented in Table 1 gives the limitations on implementing the generalized tamper detection technique for NoSQL database. As several number of NoSQL database are available, hence the most preferred five databases (other than MongoDB and Cassandra) are selected from NoSQL category and their details are presented through Table 1.

The hashing is a most commonly used technique for tamper detection [10]. Hashing is useful to identify the tampering in database [11, 12], but it won't help to identify, what has tampered? and that's the major limitation. In this view, we have developed a new tamper detection technique, which

Table 1 NoSQL database details

Database	NoSQL Category	#DB Ranking (Among NoSQL) [5]	Specification	Data Format and Working
Redis [6]	Key-value	2	In-memory database	Snapshot of RDB file memory data is stored on disk. Log of write operations has stored in AOF file. RDB data set consists of string, list, set and hash. While AOF data set contains string, ziplist, linkedlist, intset, skiplist and hash table.
Amazon DynamoDB [7]	Document Store, Key-Value store	4	Amazon web services database	Data has managed using hashing and B-trees and it has stored in the form of tables with dynamic attributes set.
Neo4j [8]	Graph	5	ACID compliant transactional database	Data is modelled in the form of graph with two elements, nodes (vertices) and relationships (edges).
HBase [9]	Wide Column	6	Hadoop database	This database is based on Hadoop distributed file system (HDFS). Data is written to HBase file called as HFile and is organized in sorted map order.

will identify the actual tampering in these databases and this will be useful in database forensics. The tamper detection result consist of Timestamp of tampering, actual operation involved in data tampering, namespace/keyspace on which tampering has happened and actual data involved in it.

1.1 Objectives

The main objectives of this article are:

- Develop a technique to identify, when and what data has tampered in MongoDB and Cassandra database in forensic environment
- Classify tampered operations as genuine or suspicious for e-commerce dataset
- Performance evaluation of data tampering in standalone, replication and sharding mode of MongoDB database & in standalone mode and high availability with local storage mode of Cassandra database

1.2 Paper Layout

The rest of this article is organized as follows. Section 2 reviews the internal working of MongoDB and Cassandra database along with attacks and security issues in these databases. The related work has mentioned in Section 3 and threat model for tamper detection is given in Section 4. A proposed technique for tamper detection in forensic environment along with its process flow and algorithms are explained in Section 5. The implementation and performance evaluation has shown in Section 6. Finally, we conclude this article in Section 7.

2 Database Internals

2.1 MongoDB

MongoDB is becoming popular [13] due to its great performance, flexibility to quickly accommodate the changes, versatility, scalability, and ease of use [14]. MongoDB is an unstructured database in which data is stored in the form of documents, which are part of the collection. The supported storage engines by MongoDB are MMAPv1, WiredTiger and In-Memory. The storage engine is a component that manages storage of data in memory and on the disk. MongoDB Enterprise supports In-Memory storage engine. MMAPv1 is not available in the current versions of MongoDB. The default storage engine is WiredTiger; hence the further details specified are allied with this storage engine. The current storage engine can be checked by issuing the command db.serverStatus().storageEngine. The data is stored in terms of page units [15]. Actual data is stored in cell units and which is presented in B-tree structure format. Data is present in the Binary Script Object Notation (BSON) format. Data is stored in compressed form using snappy as the default algorithm. From the forensic perspective, the important files of WiredTiger storage engine are _mdb_catalog.wt, Collection.wt, WiredTiger.turtle and WiredTiger.wt. These files are generally available under the data folder of MongoDB. The wiredTiger.turtle is a text file that contains the root page offset address of WiredTiger.wt file in the form of configuration string [15]. The collection *-#-####*∼.wt is the file in which actual data is stored and whenever the collection is created, this file gets generated for every new collection. The # is the number associated with the file name and the number behind the first hyphen gets incremented by one, as the file gets created. The number behind the second hyphen is the file name represented with 19 digits randomly generated numbers. This can be checked from WiredTiger.wt and _mdb_catalog.wt files. The file _mdb_catalog.wt consists of namespace and filename pair [16]. Through this storage engine, multiple clients can update different documents of the collection at the same time. Document-level concurrency control is used by this storage engine for a write operation.

2.2 Cassandra

Apache Cassandra is a column-based NoSQL database. It is highly scalable and distributed database [17], which handles very huge amount of data. Cassandra consist of peer to peer architecture, supporting no single point of failure. Whenever write operation is executed the data is written to memtable which is available in memory and simultaneously data is also written to commit log [18], which is located on disk. This duplicate write policy make sure that there is no data loss. When the storage area of memtable is full, the entire data is flushed to ssTable, which is located on disk. During read operation, the request is given to memtable to extract the requested data, if data is not available in memtable then read request is forwarded to ssTable.

2.3 Security Issues

Whenever any application is proposed, security of that application is a major concern. Though care is taken while designing such applications, but still attackers will try to break the security. The security features available in

Table 2 Security features in MongoDB and cassandra [19, 20]

MongoDB		Cassandra	
Security Feature	Stature	Security Feature	Stature
Data at rest	Not encrypted	Data at rest	Not encrypted
Authentication	Available with unshared configuration	Authentication	Available solution is not feasible
Authorization	Available with unshared configuration	Authorization	It is done at the granularity level. Available solution is not feasible
Auditing	Not available	Auditing	Not available
Injection Attacks	Possible	Injection Attacks	Possible in Cassandra Query Language
Database communication	Encryption not available	Client communication	Encryption not available
AAA (Authentication, authorization, auditing) for RESTful connections	Permissions and users are maintained externally	Intercluster network communication	Encryption is available
Fast Password Hashing	The MD5 hashing algorithm used	–	–
Salt Reuse	Mongo is used as salt for password hash calculation	–	–

MongoDB and Cassandra are explained in Table 2 along with the stature, which may hamper the security.

The major attacks happened earlier on MongoDB database are the main reasons in developing this tamper detection technique (as mentioned here). In the year 2012, South Korea’s National Intelligence Service (NIS) employee has been found dead in his car with a suicide note. According to police investigation [21], in his note, he confessed that, he deleted the data related to counter-terrorism and North Korean surveillance. NIS purchased Italian Remote control system software (RCS). As per the news, they had purchased it for research purpose. RCS is hacking software, capable of hacking computers and mobile phones. MongoDB was used as back end database.

Three groups hijacked around 26000 servers and attacked on MongoDB Database with a ransom attack. This has started in December 2016 and continued till January 2017. During this attack [22], hackers scanned internet port for MongoDB database and wiped database contents replacing it with a ransom demand of depositing Bitcoin amount. Victim restored their database with a backup copy but on the same day hacking group attacked on restored database. Starting from MongoDB, ransom attacks also spread to other server technologies like Cassandra, Hadoop, CouchDB, and MySQL [23].

3 Related Work

This article will enlighten on tamper detection in MongoDB and Cassandra database. Previous research work related to database tampering has discussed in detail and identified the potential field for this research work. As most of the researchers have mentioned about the requirement of tamper detection techniques for huge or high performance databases in their future scope, we have concentrated on the same in this research work.

Wagner et al. have presented an approach to verify the integrity of a database log [24]. An audit log is considered to be an important file, as it records the information about compromised and accessed data. All these records should be maintained as per the Sarbanes-Oxley Act (SOX) [25] and the Health Insurance Portability and Accountability Act (HIPAA) [26]. As far as database log alteration is concerned an attacker can modify write-ahead logs (WAL) and audit logs. The WAL stores recent modifications performed on tables whereas audit logs stores user actions executed on the table. As the file format of WAL is not easily readable to the user, so it is difficult to modify. This requires special tools to parse such file. The audit log can be edited in Oracle, MySQL, and PostgreSQL using sys.aud$, general_log_file, and pg_log directory respectively.

DBDetective tool has been developed to detect the tampering from database log file. Implementation and experimentation of DBDetective have been performed for deleted, inserted, and updated records. Snapshots of RAM and disk are captured and processed for above purpose using existing tool called DICE. In case of a deleted records, the record is not physically deleted from the database but it will be marked as deleted unless the new record is overwritten. In case of an insert operation, new record will be appended either at the end of all records or at the location of deleted record, depending on the size. This information is useful to find the change in an insert record. Update operation is executed with deletion of record followed by insert operation.

This experimentation is done on three well-known databases namely Oracle, MySQL, and PostgreSQL.

Tripathi and Meshram have identified the locations of digital evidence in Oracle database considering the possibilities of tampering [27]. The evidence locations are namely redo logs, locating dropped objects, live response, system change number, views and audit trail [3].

Rajguru and Sharma have proposed a model [28] for detecting the tamper in Oracle database in the form of modified table, deleted table and un-authorized access table. In the modified table module, evidence can be found by extracting information from system defined tables. Whereas, implementation of the deleted table and un-authorized access table module were left for the future scope.

Database triggers are auto-generated events in the execution of any data manipulation operation. Many times user write the trigger for created tables and views in the database. Database like oracle has the trigger functionality for schema events too. Kataria and Kanwal [29] have used concept of trigger for tamper detection. Trigger was written for a specific table and when user performs any alteration, then that information got recorded in the backup table.

Snodgrass et al. have proposed a mechanism to prevent an attacker from corrupting the audit log [30]. This mechanism is based on concept of cryptographic hash function and this context of implementation includes Berkeley DB storage engine, specifying TRANSACTIONTIME and maintaining an audit log with additional efforts. Timestamp along with hash value will be recorded for the database tuple. The notarization service will issue an ID for the recorded hash value and for any tampering in the database this ID will get altered. There is an opportunity to extend the prototype for tamper detection in the high-performance database [31].

Azemovic and Music have developed a database tamper detection model [32]. This model is useful to identify what, when and who has tampered the data. This model works in three stages, to observe the operations performed on the database, to filter out operations for identifying the modifications in database and to use hash function as an advanced security layer [33]. The hash values will be recorded separately for rows and columns of a table and detects the data tamper with mismatch of hash values. There is a scope to develop the tamper detection model, which works for commercial database system. Table 3 gives the limitations of existing tamper detection techniques for MongoDB and Cassandra database.

Table 3 Limitations of existing tamper detection techniques for MongoDB and cassandra

Reference	Tamper Detection Technique	Database	Applicability for MongoDB/ Cassandra	Remark
[24]	Based on RAM and disk snapshots	Oracle, MySQL, PostgreSQL	No	MongoDB collection files stores data in compressed format. A separate wt file will be created for every new collection (table). Only string contents will be retrieved from collection files after analyzing the disk snapshot. In Cassandra, ssTable info can can be extracted in the form of tombstone, but it will store only primary key id of updated and deleted record.
[27]	Evidence locations specified are redo logs, system change number, locating dropped objects, live response etc.	Oracle	No	These locations are not available in MongoDB and Cassandra database, however idea of log file for tamper detection is appreciated.
[29]	Based on Trigger	Oracle	No	Trigger concept is not available with MongoDB command line mode. Though Cassandra supports trigger, creating trigger for every table is not feasible and it will also generate redundancy in terms of table to store backup data.
[2, 30, 31, 33]	Based on Hashing concept	MySQL, Berkeley DB	Partially	Hashing is a general technique used to verify the integrity of the data and it won't detect any modifications performed in the database.
[32]	Based on the hash calculation for row and column values.	Relational Database	No	MongoDB stores data in an unstructured format. Though Cassandra stores data in tabular form, but hash will only notify the change in database state.

4 Threat Model

In this research work, we have considered the possibility of database tampering by authorized and unauthorized categories of users and understood the likely ways of attacks based on available literature. This work has been progressed with verified database, properly functioning hardware and correctly installed operating system. As authorized users have direct access to the database, they can easily tamper the data. Whereas, unauthorized users have no direct access, but they can access the data by cracking the security. Data tampering may happen by establishing connections with the database using programming languages like Python/Java etc. The data tampering is also possible by penetrating malware into the machine to update the record value. Ultimately database integrity may get tampered by either authorized users or unauthorized users through cracking the security. The proposed tamper detection technique will identify the modifications performed in the database. By using ecommerce case study the authenticity or doubtfulness of operations is shown [34, 35]. As the operations performed on database may be genuine or suspicious but they are application dependent and hence distinguish between them is shown for ecommerce application.

5 Tamper Detection in Forensic Environment

Tamper detection technique has been developed for MongoDB and Cassandra database within forensic environment. The major steps involved in the execution of this technique have been in detail as shown in Figure 1.

Process Flow

Step 1: Preparation at database side

Tamper detection from database will be performed with the help of log file. These log files are oplog.rs and audit.log for MongoDB and Cassandra database respectively. The Oplog.rs file will be available as collection under local database of MongoDB, whereas audit.log file will be available as text file in the log folder of Cassandra. Before accessing the log file in MongoDB, mongod configuration file must be updated with dbpath and replica set name. While accessing the audit.log file of Cassandra, system must be configured with Ecaudit plugin [36].

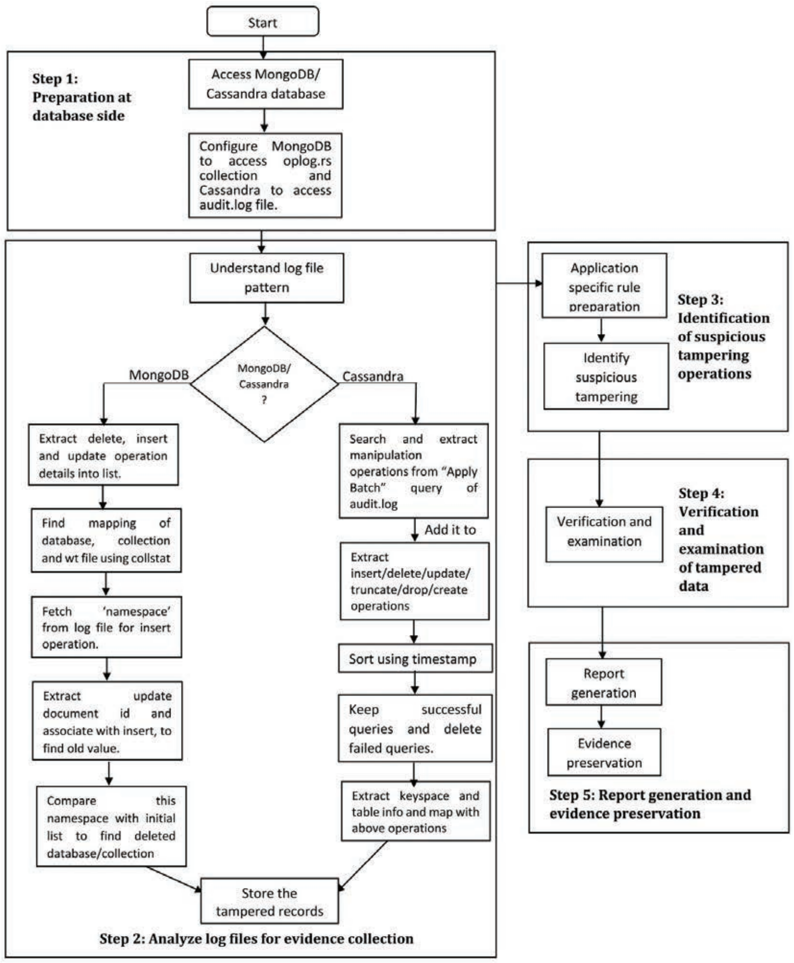

Start
Step 1: Preparation at database side
Access MongoDB/ Cassandra database
Configure MongoDB to access oplog.rs collection and Cassandra to access audit.log file.
Understand log file pattern
MongoDB/ Cassandra ?
MongoDB
Cassandra
Extract delete, insert and update operation details into list.
Find mapping of database, collection and wt file using collstat
Fetch 'namespace' from log file for insert operation.
Extract update document id and associate with insert, to find old value.
Compare this namespace with initial list to find deleted database/collection
Search and extract manipulation operations from "Apply Batch" query of audit.log
Add it to
Extract insert/delete/update/ truncate/drop/create operations
Sort using timestamp
Keep successful queries and delete failed queries.
Extract keyspace and table info and map with above operations
Store the tampered records
Step 2: Analyze log files for evidence collection
Application specific rule preparation
Identify suspicious tampering
Step 3: Identification of suspicious tampering operations
Verification and examination
Step 4: Verification and examination of tampered data
Report generation
Evidence preservation
Step 5: Report generation and evidence preservation

Figure 1 The overall process flow of the proposed technique.

Step 2: Analyse log files for evidence collection

To analyse the log files, it is important to understand the log file pattern. The algorithms used to analyse these log files and tamper detection are explained here. This technique will be useful with following operations.

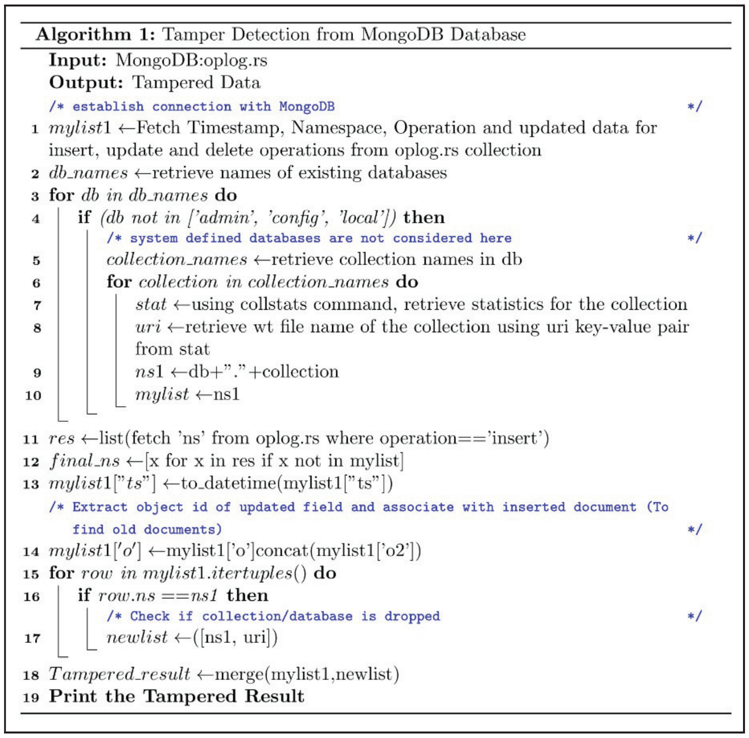
- Tampering with basic write (insert/update/delete) operations.
- Data tampering with collection/table deletion.
- Tampering with database/keyspace deletion.

Algorithms

MongoDB

The pseudocode in Algorithm 1 gives the methodology for detecting the tamper with insert, update and delete operations. The operation details have been retrieved from oplog [37]. The mapping list will be prepared with the help of collection statistics for database, collection and corresponding wt file. This is useful to detect data tampering related to collection creation and deletion. Every collection entry will be compared with the list to detect or identify the creation and deletion of collection. The missing and added entries will be detected from mapping list.

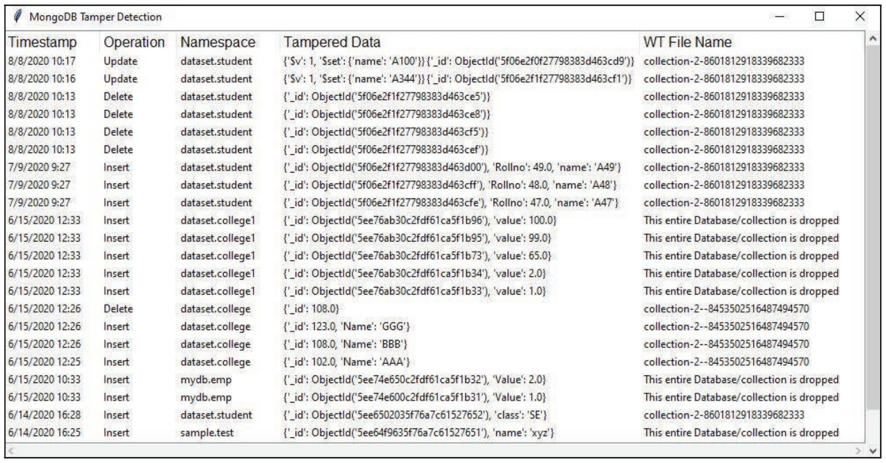
Algorithm 1: Tamper detection from MongoDB database



```
Algorithm 1: Tamper Detection from MongoDB Database
  Input: MongoDB:oplog.rs
  Output: Tampered Data
  /* establish connection with MongoDB                                  */
  mylist1 ←Fetch Timestamp, Namespace, Operation and updated data for
  insert, update and delete operations from oplog.rs collection
  db_names ←retrieve names of existing databases
  for db in db_names do
      if (db not in ['admin', 'config', 'local']) then
          /* system defined databases are not considered here          */
          collection_names ←retrieve collection names in db
          for collection in collection_names do
              stat ←using collstats command, retrieve statistics for the collection
              uri ←retrieve wt file name of the collection using uri key-value pair
              from stat
              ns1 ←db+"."+collection
              mylist ←ns1
  res ←list(fetch 'ns' from oplog.rs where operation=='insert')
  final_ns ←[x for x in res if x not in mylist]
  mylist1["ts"] ←to_datetime(mylist1["ts"])
  /* Extract object id of updated field and associate with inserted document (To
     find old documents)                                                  */
  mylist1['o'] ←mylist1['o']concat(mylist1['o2'])
  for row in mylist1.itertuples() do
      if row.ns ==ns1 then
          /* Check if collection/database is dropped                      */
          newlist ←([ns1, uri])
  Tampered_result ←merge(mylist1,newlist)
  Print the Tampered Result
```

Experiment #1: Sample dataset

The sample dataset contains dataset, mydb & sample database created in MongoDB and collections were created with student, college, college1, emp


MongoDB Tamper Detection

Timestamp	Operation	Namespace	Tampered Data	WT File Name
8/8/2020 10:17	Update	dataset.student	{'$v': 1, '$set': {'name': 'A100'}} {'_id': ObjectId('5f06e2f0f27798383d463cd9')}	collection-2-8601812918339682333
8/8/2020 10:16	Update	dataset.student	{'$v': 1, '$set': {'name': 'A344'}} {'_id': ObjectId('5f06e2f1f27798383d463cf1')}	collection-2-8601812918339682333
8/8/2020 10:13	Delete	dataset.student	{'_id': ObjectId('5f06e2f1f27798383d463ce5')}	collection-2-8601812918339682333
8/8/2020 10:13	Delete	dataset.student	{'_id': ObjectId('5f06e2f1f27798383d463ce8')}	collection-2-8601812918339682333
8/8/2020 10:13	Delete	dataset.student	{'_id': ObjectId('5f06e2f1f27798383d463cf5')}	collection-2-8601812918339682333
8/8/2020 10:13	Delete	dataset.student	{'_id': ObjectId('5f06e2f1f27798383d463cef')}	collection-2-8601812918339682333
7/9/2020 9:27	Insert	dataset.student	{'_id': ObjectId('5f06e2f1f27798383d463d00'), 'Rollno': 49.0, 'name': 'A49'}	collection-2-8601812918339682333
7/9/2020 9:27	Insert	dataset.student	{'_id': ObjectId('5f06e2f1f27798383d463cff'), 'Rollno': 48.0, 'name': 'A48'}	collection-2-8601812918339682333
7/9/2020 9:27	Insert	dataset.student	{'_id': ObjectId('5f06e2f1f27798383d463cfe'), 'Rollno': 47.0, 'name': 'A47'}	collection-2-8601812918339682333
6/15/2020 12:33	Insert	dataset.college1	{'_id': ObjectId('5ee76ab30c2fdf61ca5f1b96'), 'value': 100.0}	This entire Database/collection is dropped
6/15/2020 12:33	Insert	dataset.college1	{'_id': ObjectId('5ee76ab30c2fdf61ca5f1b95'), 'value': 99.0}	This entire Database/collection is dropped
6/15/2020 12:33	Insert	dataset.college1	{'_id': ObjectId('5ee76ab30c2fdf61ca5f1b73'), 'value': 65.0}	This entire Database/collection is dropped
6/15/2020 12:33	Insert	dataset.college1	{'_id': ObjectId('5ee76ab30c2fdf61ca5f1b34'), 'value': 2.0}	This entire Database/collection is dropped
6/15/2020 12:33	Insert	dataset.college1	{'_id': ObjectId('5ee76ab30c2fdf61ca5f1b33'), 'value': 1.0}	This entire Database/collection is dropped
6/15/2020 12:26	Delete	dataset.college	{'_id': 108.0}	collection-2--8453502516487494570
6/15/2020 12:26	Insert	dataset.college	{'_id': 123.0, 'Name': 'GGG'}	collection-2--8453502516487494570
6/15/2020 12:26	Insert	dataset.college	{'_id': 108.0, 'Name': 'BBB'}	collection-2--8453502516487494570
6/15/2020 12:25	Insert	dataset.college	{'_id': 102.0, 'Name': 'AAA'}	collection-2--8453502516487494570
6/15/2020 10:33	Insert	mydb.emp	{'_id': ObjectId('5ee74e650c2fdf61ca5f1b32'), 'Value': 2.0}	This entire Database/collection is dropped
6/15/2020 10:33	Insert	mydb.emp	{'_id': ObjectId('5ee74e600c2fdf61ca5f1b31'), 'Value': 1.0}	This entire Database/collection is dropped
6/14/2020 16:28	Insert	dataset.student	{'_id': ObjectId('5ee6502035f76a7c61527652'), 'class': 'SE'}	collection-2-8601812918339682333
6/14/2020 16:25	Insert	sample.test	{'_id': ObjectId('5ee64f9635f76a7c61527651'), 'name': 'xyz'}	This entire Database/collection is dropped

Figure 2 MongoDB: tampered data.

and test respectively. JSON documents were inserted in each collection and random write operations (update and delete) including collection creation and deletion were performed. Figure 2 shows the results for tampered data. Whenever any collection is created in MongoDB, it will generate a data file named collection-XXXXX.wt. In the following result namespace name and its corresponding data file name is also printed. If any database/collection is deleted, that is marked in WT File Name column as "This entire/Database/collection is dropped".

Cassandra

The pseudocode shown in Algorithm 2 is used to detect the tamper from Cassandra. Cassandra's log file is read to extract the data manipulation queries like insert, update, delete, truncate, create and drop. The log file logs failed queries also. So these queries are identified and deleted from log file. Data manipulation queries are also given in batches, so these queries are also identified. Identification and association of keyspace & table name is also a challenging task because queries can be used in different ways as shown below.

Query 1: Create table sample.employee. . . . . . .
Query 2: use sample;
Create table employee. . . . .

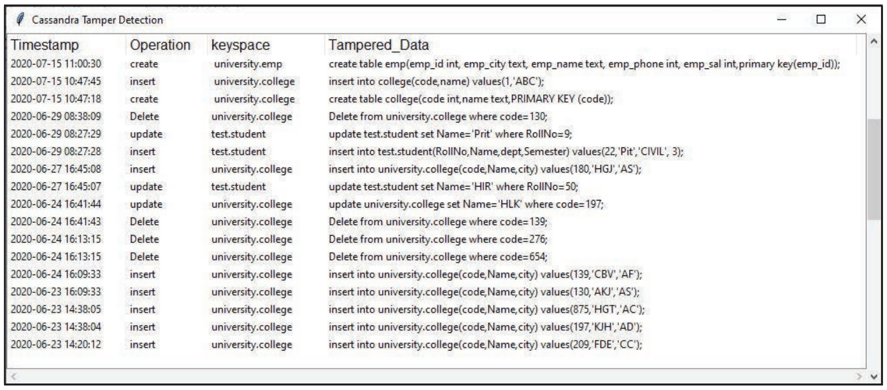
In query 1, table name is used along with keyspace name and in query 2, initially keyspace is used and later table name is used.

Algorithm 2: Tamper detection from Cassandra database

```
Algorithm 2 : Tamper Detection from Cassandra Database
Input: Cassandra:Audit.log
Output: Tampered Data
$infile$ ←open("audit.log")
for $line1$ $in$ $infile$ do
    if ("BEGIN BATCH" in line1) then
    $v1$←line1[0 : 50] /* Extract up to timestamp details */
    $nextline$ ←next(infile)
    while ("APPLY BATCH" not in nextline) do
        $mydf$ ← ($v1 + nextline$)
    $nextline$ ←next(infile)
$data$←insert('Timestamp')
$data$←insert('Status')
$data$←insert('Operation')(Create‖$Insert$‖$Update$‖$Delete$‖$Drop$‖$Truncate$)
$data$←insert('Tampered_Data')(Create.*‖$Insert.*$ ‖$Update.*$ ‖$Delete.*$ ‖$Drop.*$ ‖$Truncate.*$)
$s$ ←data('Status').eq('status=FAILED')
/* Find and delete failed queries */
$grp$ ←s.groupby(data('Timestamp'))
$m1$ ←grp.transform('any')
$m2$ ←grp.cumsum().ne(0)
$data$ ←data(($\tilde{m}1$ ‖$m2$)&$\tilde{m}$)
$flag$ ← 0
for $t$ $in$ ('Tampered_Data') do
    if re.search(r'[.], re.split(" ",t)))==True then
        $flag$ ← 0
    if 'USE' in $t$ then
        $d$←(t.split("USE",1)[1])
        $new\_data$←new_data.append([d], ignore_index=True)
        $flag$← 1
    if (Create‖Insert‖Update‖Delete‖Truncate‖Drop)in($t$) then
        $res$←Extract keyspace
        if $flag$==0 then
            new_data←new_data.append(res)
        else
            new_data←new_data.append(d+"."+res)
$data$←data.join(new_data('keyspace'))
Print the Tampered Result
```

Experiment #1: Sample dataset

The sample keyspace "university" with table, emp & college and another keyspace "test" with table, student were created by inserting random records. The data manipulation operations including update and delete were performed randomly. The result of tampered records is as given in Figure 3.



Timestamp	Operation	keyspace	Tampered_Data
2020-07-15 11:00:30	create	university.emp	create table emp(emp_id int, emp_city text, emp_name text, emp_phone int, emp_sal int,primary key(emp_id));
2020-07-15 10:47:45	insert	university.college	insert into college(code,name) values(1,'ABC');
2020-07-15 10:47:18	create	university.college	create table college(code int,name text,PRIMARY KEY (code));
2020-06-29 08:38:09	Delete	university.college	Delete from university.college where code=130;
2020-06-29 08:27:29	update	test.student	update test.student set Name='Prit' where RollNo=9;
2020-06-29 08:27:28	insert	test.student	insert into test.student(RollNo,Name,dept,Semester) values(22,'Pit','CIVIL', 3);
2020-06-27 16:45:08	insert	university.college	insert into university.college(code,Name,city) values(180,'HGJ','AS');
2020-06-27 16:45:07	update	test.student	update test.student set Name='HIR' where RollNo=50;
2020-06-24 16:41:44	update	university.college	update university.college set Name='HLK' where code=197;
2020-06-24 16:41:43	Delete	university.college	Delete from university.college where code=139;
2020-06-24 16:13:15	Delete	university.college	Delete from university.college where code=276;
2020-06-24 16:13:15	Delete	university.college	Delete from university.college where code=654;
2020-06-24 16:09:33	insert	university.college	insert into university.college(code,Name,city) values(139,'CBV','AF');
2020-06-23 16:09:33	insert	university.college	insert into university.college(code,Name,city) values(130,'AKJ','AS');
2020-06-23 14:38:05	insert	university.college	insert into university.college(code,Name,city) values(875,'HGT','AC');
2020-06-23 14:38:04	insert	university.college	insert into university.college(code,Name,city) values(197,'KJH','AD');
2020-06-23 14:20:12	insert	university.college	insert into university.college(code,Name,city) values(209,'FDE','CC');

Figure 3 Cassandra: tampered data.

Step 3: Identification of suspicious tampering operations

- ***Application specific rule preparation:***

The results of tampered data from MongoDB database have been distinguished into genuine and suspicious tampering with the help of rule based classification [38] for ecommerce data set. Case study of ecommerce dataset (India) with following collections has been used. The dataset consists of around 10000 json documents in each collection.

Customer_data (CustomerName, contactFirstName, contactLastName, Phone, Address, City, State, postalcode, creditLimit)
Products (productName, productLine, productScale, productVendor, productDescription, quantity, buyPrice, MSRP)
Orders (Order_id, orderDate, requiredDate, shippedDate, status, comments, CustomerNumber)
Order_details (OrderNumber, Network, Card Details, shippingCity, productCode, quantityOrdered, price Each, orderLine, paymentMethod)

Suspicious tampering does not mean a fraudulent activity but it can be used as an alert for an investigator to verify that order. Depending on type of tampering operation, rules are specified as shown in Table 4.

- ***Identify suspicious tampering***

The result of this classification is shown as snapshots, by highlighting suspicious tampering in Figure 4 for insert operation and in Figure 5 for update operation.

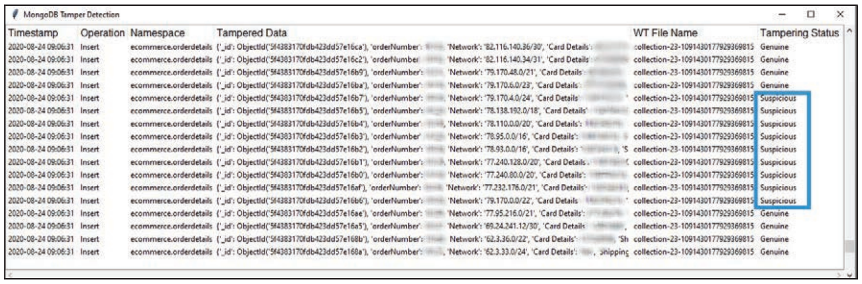
Table 4 Rules for tampering status classification

Tampering Operation	Rule	Description	Type of Attack

Insert	IF CustoNo = A AND orderNo = N AND Network = X AND CardNo = C THEN **Tampering Status** = **Genuine**
	
	IF CustoNo = A AND orderNo = N AND Network = X, Y, Z AND CardNo = C THEN **Tampering Status** = **Suspicious**
		If a customer performs online purchase from different network addresses (IPs) using same card details and if geographical locations of IPs are found far away (geographical locations can be verified using network details) then that purchase might have been done by misuse of credit card details.	Identity Theft [39]

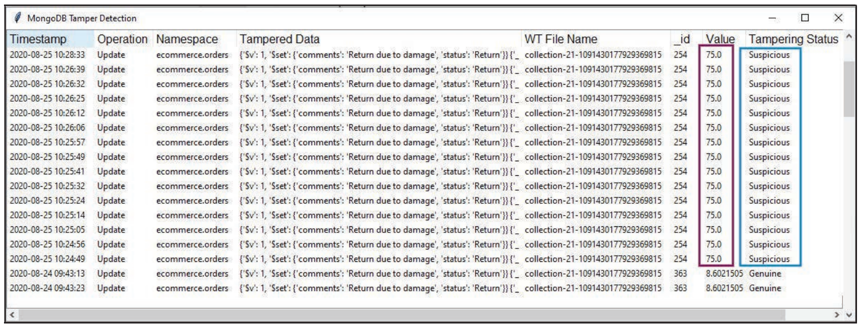
Update	IF CustoNo = A AND orderNo = N AND status = "Placed" or status = "shipped" THEN **Tampering Status** = **Genuine** IF CustoNo = A AND orderNo = N AND status = "Cancelled" AND comments="Return due to damage" AND count>=70% THEN **Tampering Status** = **Suspicious**	If customer purchases online products and returns used products by claiming issue of damage etc. if this happens multiple time (>= 70%) from the same customer. Here online service provider may consider product return count and accordingly that customer may be considered in the list of suspicious.	Friendly Fraud [39]
Delete	As delete operation is restricted hence not considered for this case study. Tampering with delete operations may be useful for data recovery purpose.		

The coverage and accuracy are the parameters associated with the specified rule. When the given transactional operations are satisfied by this rule then that rule covers those tuples. Coverage is the percentage of number of tuples covered by rule and the total number of tuples present in the transaction. Accuracy is the percentage of number of tuples correctly classified by rule and the number of tuples covered by rule. The coverage for given rule R,



```
MongoDB Tamper Detection
Timestamp           Operation Namespace               Tampered Data                                                                                                                       WT File Name                           Tampering Status
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16ca'), 'orderNumber': [redacted] 'Network': '82.116.140.36/30', 'Card Details': [redacted]     collection-23-1091430177929369815      Genuine
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16c2'), 'orderNumber' [redacted] 'Network': '82.116.140.34/31', 'Card Details [redacted]       collection-23-1091430177929369815      Genuine
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16b9'), 'orderNumber': [redacted] 'Network': '79.170.48.0/21', 'Card Details [redacted]        collection-23-1091430177929369815      Genuine
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16ba'), 'orderNumber': [redacted] 'Network': '79.170.6.0/23', 'Card Details': [redacted]        collection-23-1091430177929369815      Genuine
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16b7'), 'orderNumber': [redacted], 'Network': '79.170.4.0/24', 'Card Details' [redacted] '      collection-23-1091430177929369815      Suspicious
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16b5'), 'orderNumber': [redacted] 'Network': '78.138.192.0/18', 'Card Details' [redacted]       collection-23-1091430177929369815      Suspicious
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16b4'), 'orderNumber' [redacted], 'Network': '78.110.0.0/20', 'Card Details': [redacted]        collection-23-1091430177929369815      Suspicious
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16b3'), 'orderNumber' [redacted] 'Network': '78.95.0.0/16', 'Card Details': [redacted]          collection-23-1091430177929369815      Suspicious
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16b2'), 'orderNumber': [redacted] 'Network': '78.93.0.0/16', 'Card Details': [redacted], 'S     collection-23-1091430177929369815      Suspicious
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16b1'), 'orderNumber': [redacted], 'Network': '77.240.128.0/20', 'Card Details [redacted]      collection-23-1091430177929369815      Suspicious
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16b0'), 'orderNumber': [redacted] 'Network': '77.240.80.0/20', 'Card Details': [redacted]       collection-23-1091430177929369815      Suspicious
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16af'), 'orderNumber': [redacted] 'Network': '77.232.176.0/21', 'Card Details': [redacted],     collection-23-1091430177929369815      Suspicious
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16b6'), 'orderNumber': [redacted] 'Network': '79.170.0.0/22', 'Card Details [redacted] '        collection-23-1091430177929369815      Suspicious
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16ae'), 'orderNumber': [redacted] 'Network': '77.95.216.0/21', 'Card Details': [redacted]       collection-23-1091430177929369815      Genuine
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e16a5'), 'orderNumber': [redacted] 'Network': '69.24.241.12/30', 'Card Details [redacted],      collection-23-1091430177929369815      Genuine
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e168b'), 'orderNumber': [redacted] Network': '62.3.36.0/22', 'Card Details': [redacted], 'Sh     collection-23-1091430177929369815      Genuine
2020-08-24 09:06:31 Insert    ecommerce.orderdetails  {'_id': ObjectId('5f4383170fdb423dd57e168a'), 'orderNumber': [redacted], 'Network': '62.3.33.0/24', 'Card Details': [redacted], Shipping collection-23-1091430177929369815   Genuine
```

Figure 4 Insert: classification of tampered data.



```
MongoDB Tamper Detection
Timestamp           Operation Namespace         Tampered Data                                                                      WT File Name                        _id  Value      Tampering Status
2020-08-25 10:28:33 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:26:39 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:26:32 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:26:25 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:26:12 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:26:06 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:25:57 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:25:49 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:25:41 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:25:32 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:25:24 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:25:14 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:25:05 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:24:56 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-25 10:24:49 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   254  75.0       Suspicious
2020-08-24 09:43:13 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   363  8.6021505  Genuine
2020-08-24 09:43:23 Update    ecommerce.orders  {'$v': 1, '$set': {'comments': 'Return due to damage', 'status': 'Return'}} {'_   collection-21-1091430177929369815   363  8.6021505  Genuine
```

Figure 5 Update: classification of tampered data.

is given by Equation (1) and accuracy is given by (2).

$$coverage(R) = \frac{ncovers}{|D|} \quad (1)\ [38]$$

$$accuracy(R) = \frac{ncorrect}{ncovers} \quad (2)\ [38]$$

Where,
ncovers $=$ The number of tuples covered by rule
$|D| =$ Total number of tuples present in transaction
ncorrect $=$ The number of tuples correctly classified by rule

The coverage and accuracy for insert and update tampering are given in Table 5. For insert operation there are 9 tuples which satisfies the rule and for update operation, the rule is covered by 21 tuples among 10000 transactional tuples.

Table 5 Coverage and accuracy for insert and update operation

Insert	$coverage(R) = \frac{9}{10000} = 0.09\%$
	$accuracy(R) = \frac{9}{9} = 100\%$
Update	$coverage(R) = \frac{21}{10000} = 0.21\%$
	$accuracy(R) = \frac{21}{21} = 100\%$

Step 4: Verification and examination of tampered data

The tampering operations classified as suspicious in previous step are verified and examined to fetch the customer details. In examination step investigator may come up with the following three possibilities by verifying tampered details.

1. The suspicious transaction seems to be fraudulent
2. The suspicious transaction looks like a valid transaction
3. The suspicious transaction may be valid or fraudulent. Due to lack of supporting information final decision is delayed and it needs further investigation.

Step 5: Report generation and evidence preservation

- ***Report generation:***

Based on identification of fraudulent tampering, the report can be prepared with supporting evidence details.

- ***Evidence Preservation:***

The forensic investigation process starts by creating image of source. The evidences are analyzed from image copy. For evidence preservation following methods are useful [40].

i. Drive imaging – The entire drive image is created for analysis purpose. Write-blocker may be useful to create image.
ii. Hashing – Integrity of image can be verified by calculating hash value (MD5, SH-1).
iii. Chain of Custody – This document maintains the evidence collected in chronological order.

6 Implementation and Performance Evaluation

6.1 Implementation

A system with specifications: windows 10 64-bit OS, 3 GB RAM with Intel Core 2.40 GHz i3 processor has been used for implementation. Algorithms are designed with python 3.7, MongoDB version 4.0 and Cassandra 3.11.6. The operations are performed on sample collections/tables to verify the result. To analyse the performance of proposed technique, large datasets are used. Data tampering results for large datasets can be exported to either excel or CSV file.

6.2 Performance Evaluation

Experiment #2: Large dataset

The data sets for experimentation were downloaded from GitHub for MongoDB [41]. The actual data size for experimentation was large, but MongoDB WiredTiger storage engine stores data in compressed form. Hence experiments were carried for compressed database size; varying from 610KB to 423MB. For Cassandra, the dataset was created manually by inserting, deleting and updating records randomly. Table 6 shows the number of tampered documents correctly identified and execution time for the same. Figure 6 shows the graphical analysis of tamper detection technique in standalone setup for MongoDB and Cassandra database.

For MongoDB more than one million tampered records were detected in one minute [42] and Cassandra shows more time requirement when dataset size is increased [43]. The reasons for the same are explained through Table 7.

The MongoDB log file consists of fields timestamp(ts), unique id for each operation(h), any operation executed(t), version number(v), operation type(op), namespace(ns), wallclock (wall), actual operation executed(o) and for update related operation(o2). The snapshot of MongoDB log file is as shown in Figure 7. The sample snapshot consists of insert operation executed on ecommerce.orders collection.

The Cassandra log file consists of timestamp, client connection port (client), username used for Cassandra connection, status of executed query and actual query executed. The snapshot of Cassandra log file is as shown in Figure 8. The log file can be configured through Cassandra's audit.yaml file to change the parameters depending on requirement.

Table 6 Execution Time Analysis of MongoDB and Cassandra Tamper detection

Number of Tampered Documents Identified	Operation Type	Number of Operations	Execution Time (Seconds)	
			MongoDB (Algorithm 1)	Cassandra (Algorithm 2)
500	Insert	200	0.19	0.53
	Update	200		
	Delete	100		
2500	Insert	1000	0.45	1.86
	Update	800		
	Delete	700		
12500	Insert	5000	0.64	10.23
	Update	4000		
	Delete	3500		
62500	Insert	22000	3.06	91.51
	Update	20000		
	Delete	20500		
312500	Insert	115000	28.32	1508.9
	Update	100000		
	Delete	97500		
1562500	Insert	600000	76.34	–
	Update	500000		
	Delete	462500		
3000000	Insert	1000000	184.24	–
	Update	1000000		
	Delete	1000000		

Experiment #3: Tamper detection in different deployment mode

The performance of tamper detection from MongoDB and Cassandra in standalone mode is already shown through experiment #1 and #2. In this section performance analysis is discussed for other deployment modes of these two databases.

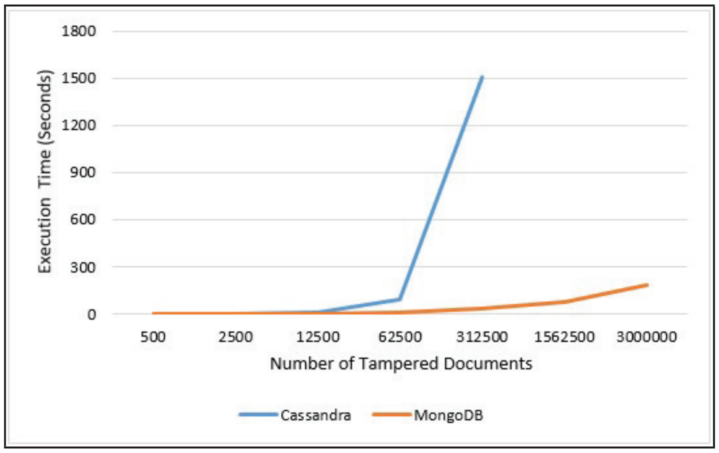


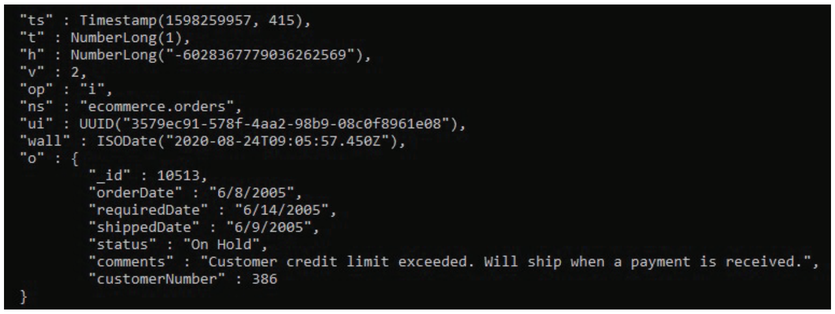
Figure 6 Performance: MongoDB and Cassandra tamper detection.

Table 7 Reasons for variation in execution time analysis of MongoDB and cassandra

Parameter	MongoDB	Cassandra
Log File	It is available in the form of database collection.	It is available in the form of text file.
Query Execution and Logging	Only successful queries are logged.	All executed queries are logged to log file. Separate code is written to remove failed queries.
Bulk Query Execution	For bulk query execution only actual write operation queries will be logged to log file.	It will log all queries including BEGIN BATCH and APPLY BATCH. Separate code is written to remove unwanted queries and other details.
Namespace	Log file contains all queries with combination of database and collection name.	Log file may not contain all queries with combination of keyspace and table name. Separate code is written to form this combination.

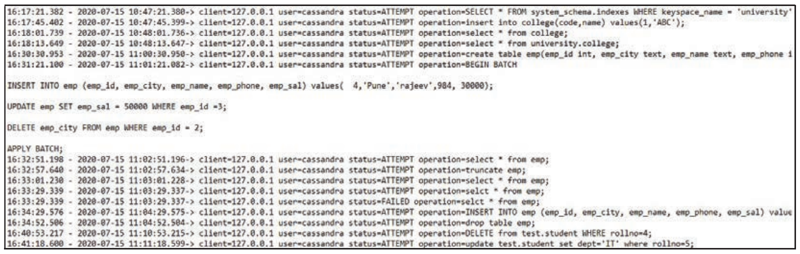
MongoDB

MongoDB supports replication and sharding as deployment modes along with standalone. The performance of proposed tamper detection technique in replication and sharding mode is as mentioned below.



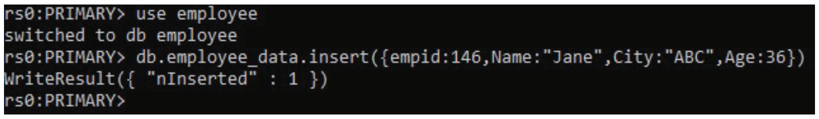
```
"ts" : Timestamp(1598259957, 415),
"t" : NumberLong(1),
"h" : NumberLong("-6028367779036262569"),
"v" : 2,
"op" : "i",
"ns" : "ecommerce.orders",
"ui" : UUID("3579ec91-578f-4aa2-98b9-08c0f8961e08"),
"wall" : ISODate("2020-08-24T09:05:57.450Z"),
"o" : {
        "_id" : 10513,
        "orderDate" : "6/8/2005",
        "requiredDate" : "6/14/2005",
        "shippedDate" : "6/9/2005",
        "status" : "On Hold",
        "comments" : "Customer credit limit exceeded. Will ship when a payment is received.",
        "customerNumber" : 386
}
```

Figure 7 MongoDB: snapshot of log file.



```
16:17:21.382 - 2020-07-15 10:47:21.380-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=SELECT * FROM system_schema.indexes WHERE keyspace_name = 'university'
16:17:45.402 - 2020-07-15 10:47:45.399-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=insert into college(code,name) values(1,'ABC');
16:18:01.739 - 2020-07-15 10:48:01.736-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=select * from college;
16:18:13.649 - 2020-07-15 10:48:13.647-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=select * from university.college;
16:30:30.953 - 2020-07-15 11:00:30.950-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=create table emp(emp_id int, emp_city text, emp_name text, emp_phone i
16:31:21.100 - 2020-07-15 11:01:21.082-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=BEGIN BATCH

INSERT INTO emp (emp_id, emp_city, emp_name, emp_phone, emp_sal) values(  4,'Pune','rajeev',984, 30000);

UPDATE emp SET emp_sal = 50000 WHERE emp_id =3;

DELETE emp_city FROM emp WHERE emp_id = 2;

APPLY BATCH;
16:32:51.198 - 2020-07-15 11:02:51.196-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=select * from emp;
16:32:57.640 - 2020-07-15 11:02:57.634-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=truncate emp;
16:33:01.230 - 2020-07-15 11:03:01.228-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=select * from emp;
16:33:29.339 - 2020-07-15 11:03:29.337-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=selct * from emp;
16:33:29.339 - 2020-07-15 11:03:29.337-> client=127.0.0.1 user=cassandra status=FAILED operation=selct * from emp;
16:34:29.576 - 2020-07-15 11:04:29.575-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=INSERT INTO emp (emp_id, emp_city, emp_name, emp_phone, emp_sal) value
16:34:52.506 - 2020-07-15 11:04:52.504-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=drop table emp;
16:40:53.217 - 2020-07-15 11:10:53.215-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=DELETE from test.student WHERE rollno=4;
16:41:18.600 - 2020-07-15 11:11:18.599-> client=127.0.0.1 user=cassandra status=ATTEMPT operation=update test.student set dept='IT' where rollno=5;
```

Figure 8 Cassandra: snapshot of log file.



```
rs0:PRIMARY> use employee
switched to db employee
rs0:PRIMARY> db.employee_data.insert({empid:146,Name:"Jane",City:"ABC",Age:36})
WriteResult({ "nInserted" : 1 })
rs0:PRIMARY>
```

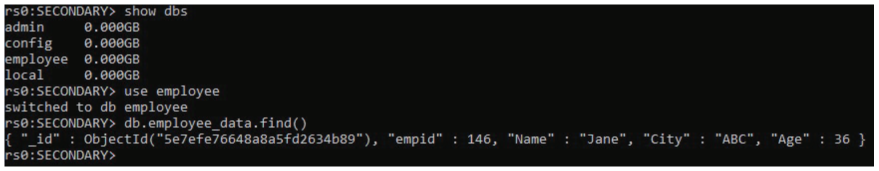
Figure 9 Primary replica set.

Replication:

Availability of data has confirmed through replication and data is synchronized across several servers through replica set. The single server data loss has protected by other nodes in replication. The primary and secondary replica was set for experimentation purpose as shown in Figures 9 and 10. Employee database containing employee_data collection was created in primary replica set and same collection was accessed through secondary replica for setup verification.

Sharding:

In sharding, data will be distributed among several servers. Horizontal data scaling will be performed through sharding. This mode gives better



```
rs0:SECONDARY> show dbs
admin     0.000GB
config    0.000GB
employee  0.000GB
local     0.000GB
rs0:SECONDARY> use employee
switched to db employee
rs0:SECONDARY> db.employee_data.find()
{ "_id" : ObjectId("5e7efe76648a8a5fd2634b89"), "empid" : 146, "Name" : "Jane", "City" : "ABC", "Age" : 36 }
rs0:SECONDARY>
```

Figure 10 Secondary replica set.

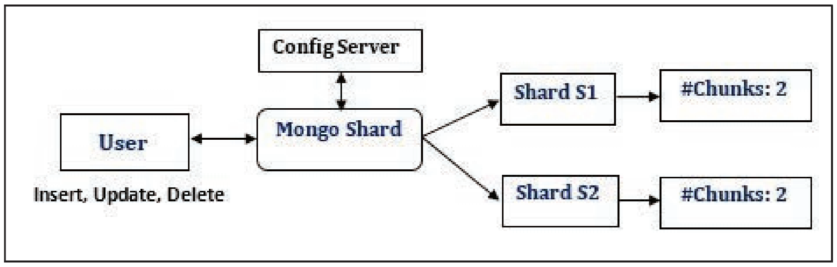


Figure 11 Sharding setup.

performance and scalability for index and range based queries. As shown in Figure 11 shards and chunks were created for experimentation purpose. Mongo shard termed as mongos is useful to execute write operations. Shard S1 and S2 are the MongoDB instances, which holds the actual data and will partition the data in the form of chunk. With the help of shard key, minimum and maximum range is specified for chunk. Config server stores the details about MongoDB instances. As documents were varied for tamper detection, the minimum and maximum values for documents are not mentioned in the chunks.

In replication and sharding modes, performance evaluation of proposed tamper detection technique has been executed with the dataset used in experiment #2. Table 8 shows the execution time required for tamper detection in both these modes and Figure 12 shows the graphical analysis for the same. The tampered documents are combinations of insert, update and delete operations as shown in Table 6. Up to 3 lac documents, it takes almost same time for execution in both the modes. However, after 3 lac documents sharding mode requires more execution time for detection of tampered data. This is because, sharding gives high performance for fetching the records based on range or index queries and here all tampered records are retrieved, which does not include any index.

Table 8 Execution time analysis of tamper detection in replication and Sharding mode of MongoDB

Number of Tampered Documents Identified	Execution Time (Seconds) MongoDB (Replica Mode)	MongoDB (Sharding Mode)
500	0.15	0.04
2500	0.25	0.56
12500	0.75	0.38
62500	2.18	1.99
312500	21.58	10.76
1562500	95.87	172.69

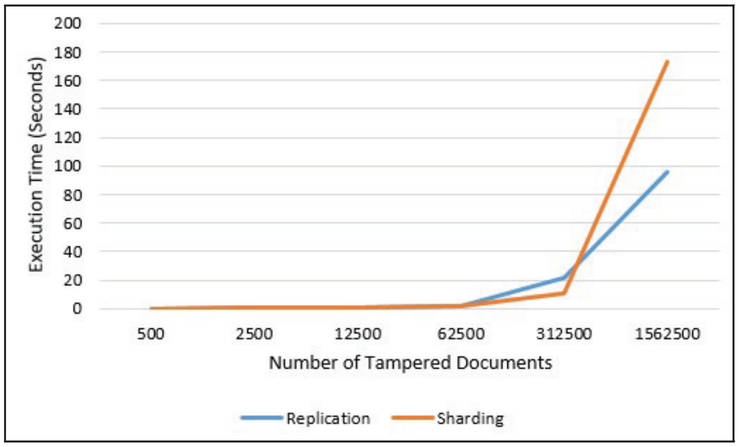


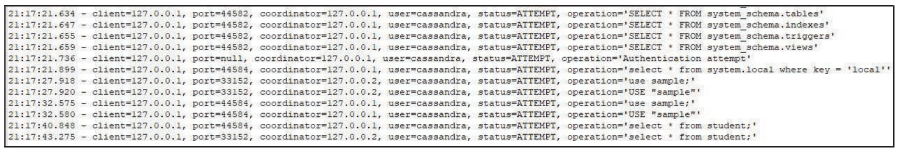
Figure 12 Performance: tamper detection in replication and sharding mode.

Cassandra

Cassandra database can be deployed in three modes namely standalone, high availability with local storage and high availability with remote storage [44].

High availability with local storage

In high availability with local storage, Cassandra node and API gateway instance runs on the same host. The multi node cluster has setup on single machine and generation of log file is verified. Here two Cassandra instance nodes were created to form the cluster. The setup is using configuration files of Cassandra. The generated log file snapshot is shown in Figure 13.



```
21:17:21.634 - client=127.0.0.1, port=44582, coordinator=127.0.0.1, user=cassandra, status=ATTEMPT, operation='SELECT * FROM system_schema.tables'
21:17:21.647 - client=127.0.0.1, port=44582, coordinator=127.0.0.1, user=cassandra, status=ATTEMPT, operation='SELECT * FROM system_schema.indexes'
21:17:21.655 - client=127.0.0.1, port=44582, coordinator=127.0.0.1, user=cassandra, status=ATTEMPT, operation='SELECT * FROM system_schema.triggers'
21:17:21.659 - client=127.0.0.1, port=44582, coordinator=127.0.0.1, user=cassandra, status=ATTEMPT, operation='SELECT * FROM system_schema.views'
21:17:21.736 - client=127.0.0.1, port=null, coordinator=127.0.0.1, user=cassandra, status=ATTEMPT, operation='Authentication attempt'
21:17:21.899 - client=127.0.0.1, port=44584, coordinator=127.0.0.1, user=cassandra, status=ATTEMPT, operation='select * from system.local where key = 'local''
21:17:27.918 - client=127.0.0.1, port=33152, coordinator=127.0.0.2, user=cassandra, status=ATTEMPT, operation='use sample;'
21:17:27.920 - client=127.0.0.1, port=33152, coordinator=127.0.0.2, user=cassandra, status=ATTEMPT, operation='USE "sample"'
21:17:32.575 - client=127.0.0.1, port=44584, coordinator=127.0.0.1, user=cassandra, status=ATTEMPT, operation='use sample;'
21:17:32.580 - client=127.0.0.1, port=44584, coordinator=127.0.0.1, user=cassandra, status=ATTEMPT, operation='USE "sample"'
21:17:40.848 - client=127.0.0.1, port=44584, coordinator=127.0.0.1, user=cassandra, status=ATTEMPT, operation='select * from student;'
21:17:43.275 - client=127.0.0.1, port=33152, coordinator=127.0.0.2, user=cassandra, status=ATTEMPT, operation='select * from student;'
```

Figure 13 Cassandra: Snapshot of Log File with Multimode Cluster.

In this snapshot additional parameters are port and coordinator as compared to standalone mode log file.

From the snapshot, the different coordinators can be observed, which shows that who have initiated the operations. The performance of tamper detection depends on the number of operations generated in log file (as Cassandra's log file is generated in text file format) and it is already analysed in standalone mode (Table 6).

High availability with remote storage

In this mode, Cassandra node and API gateway instance runs on the different host and it requires open ports in firewall. Here log file will be generated in the same way as like high availability with local storage.

7 Conclusion and Future Work

Now a days considering the necessity of digitization in every business sector, security of database is at prime importance. Because of the increased number of cybercrimes, it is highly essential to secure and maintain the integrity of important data stored in database. Tamper detection techniques are more useful for maintaining the integrity of database. The establishments; where huge data handling is involved, it attracts for implementation of tamper detection techniques with high performance database management systems such as MongoDB and Cassandra. Most of the earlier research work in tamper detection has inclined towards relational database. Hence in this research work, the aspect of tamper detection in MongoDB and Cassandra database has been concentrated. A unique tamper detection technique has been developed to detect the data tampering more effectively, in standalone, replication and sharding deployment modes of MongoDB as well as standalone, high availability with local storage mode of Cassandra. As this technique detects the data tampering with its classification into suspicious and genuine, it will be very useful for database forensics. Detection of data tamper in NoSQL database like Redis, CouchDB and HBase etc. using their respective log file, remains in the future scope.

Declaration of Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors wish to acknowledge the Information Security Education and Awareness Project, Department of Electronics and Information Technology, Ministry of Communications and Information Technology, Government of India which has made it possible to undertake this research.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

[1] Guo J. 2011. Fragile watermarking scheme for tamper detection of relational database. In: 2011 Int. Conf. Comput. Manag. pp 1–4.

[2] Pavlou KE, Snodgrass RT. 2008. Forensic analysis of database tampering. ACM Trans Database Syst 33(4):30.

[3] Chopade R, Pachghare VK. 2019. Ten years of critical review on database forensics research. Digit. Investig.

[4] Cankaya EC, Kupka B. 2016. A survey of digital forensics tools for database extraction. In: Futur. Technol. Conf. pp 1014–1019.

[5] DB-Engines Ranking – popularity ranking of database management systems. Available from: https://db-engines.com/en/ranking.

[6] Xu M, Xu X, Xu J, Ren Y, Zhang H, Zheng N. 2014. A forensic analysis method for Redis database based on RDB and AOF file. J Comput 9(11):2538–2544.

[7] What Is Amazon DynamoDB? – Amazon DynamoDB. Available from: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html.

[8] Neo4j – Overview – Tutorialspoint. Available from: https://www.tutorialspoint.com/neo4j/neo4j_overview.htm.

[9] Apache HBase – Apache HBaseTM Home. Available from: https://hbase.apache.org/.

[10] Kataria C, Kanwal G. 2015. Database tamper detection. Int. J. 5.

[11] Kumbhare R, Nimbalkar S, Chopade R, Pachghare VK. 2020. Tamper Detection in MongoDB and CouchDB Database. In: Proceeding Int. Conf. Comput. Sci. Appl. pp 109–117.

[12] Golhar A, Janvir S, Chopade R, Pachghare VK. 2020. Tamper Detection in Cassandra and Redis Database—A Comparative. In: Proceeding Int. Conf. Comput. Sci. Appl. ICCSA 2019. p 99.

[13] The MongoDB 4.2 Manual – MongoDB Manual. Available from: https://docs.mongodb.com/manual/.

[14] Mango DB. Top 5 considerations when evaluating NoSQL Databases. White Pap.

[15] Yoon J, Lee S. 2018. A method and tool to recover data deleted from a MongoDB. Digit Investig 24:106–120.

[16] Yoon J, Jeong D, Kang C, Lee S. 2016. Forensic investigation framework for the document store NoSQL DBMS: MongoDB as a case study. Digit Investig 17:53–65.

[17] What is Cassandra? | Datastax. Available from: https://www.datastax.com/cassandra.

[18] How is data written? | Apache Cassandra 3.0. Available from: https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/dml/dmlHowDataWritten.html.

[19] Okman L, Gal-Oz N, Gonen Y, Gudes E, Abramov J. 2011. Security issues in nosql databases. In: 2011 IEEE 10th Int. Conf. Trust. Secur. Priv. Comput. Commun. pp 541–547.

[20] Aggarwal P, Rani R. 2014. Security issues and user authentication in MongoDB.

[21] South Korean intelligence employee found dead – CNN. Available from: https://edition.cnn.com/2015/07/20/asia/south-korea-nis-suicide/index.html.

[22] Other 26,000 MongoDB servers hit in a new wave of ransom attacks Security Affairs. Available from: https://securityaffairs.co/wordpress/62717/cyber-crime/mongodb-ransom-attacks.html.

[23] Almost 4,000 databases wiped in ‘Meow’ attacks | WeLiveSecurity. Available from: https://www.welivesecurity.com/2020/07/27/almost-4000-databases-wiped-meow-attacks/.

[24] Wagner J, Rasin A, Glavic B, Heart K, Furst J, Bressan L, Grier J. 2017. Carving database storage to detect and trace security breaches. Digit Investig 22:S127–S136.

[25] The Sarbanes-Oxley Act 2002. Available from: http://www.soxlaw.com/.

[26] Summary of the HIPAA Privacy Rule | HHS.gov. Available from: https://www.hhs.gov/hipaa/for-professionals/privacy/laws-regulations/index.html.

[27] Tripathi S, Meshram BB. 2012. Digital evidence for database tamper detection. J Inf Secur 3(02):113.

[28] Rajguru S, Sharma D. 2014. Database Tamper Detection and Analysis. Int. J. Comput. Appl. 105.

[29] Kataria C, Kanwal G. 2015. To detect who and when tamper data in database. Int J Eng Res Technol 4(06):181–2278.

[30] Snodgrass RT, Yao SS, Collberg C. 2004. Tamper detection in audit logs. In: Proc. Thirtieth Int. Conf. Very large data bases-Volume 30. pp 504–515.

[31] Khanuja HK, Adane DS. 2011. Database security threats and challenges in database forensic: A survey. Proc. 2011 Int. Conf. Adv. Inf. Technol. (AIT 2011), available http//www.ipcsit.com/vol20/33-ICAIT2011-A4 072.pdf.

[32] Azemović J, Mušić D. 2009. Efficient model for detection data and data scheme tempering with purpose of valid forensic analysis. 2009 Int. Conf. Comput. Eng. Appl. (ICCEA 2009).

[33] Kambire MK, Gaikwad PH, Gadilkar SY, Funde YA. 2015. An improved framework for tamper detection in databases. Int J Comput Sci Inf Technol 6:57–60.

[34] Camino RD, State R, Montero L, Valtchev P. 2017. Finding Suspicious Activities in Financial Transactions and Distributed Ledgers. In: 2017 IEEE Int. Conf. Data Min. Work. pp 787–796.

[35] Khanuja HK, Adane D. 2018. Detection of Suspicious Transactions with Database Forensics and Theory of Evidence. In: Int. Symp. Secur. Comput. Commun. pp 419–430.

[36] Ericsson/ecaudit: Ericsson Audit plug-in for Apache Cassandra. Available from: https://github.com/Ericsson/ecaudit.

[37] MongoDB: Understanding Oplog: Available from: http://dbversity.com/mongodb-understanding-oplog/.

[38] Han J, Pei J, Kamber M. 2011. Data mining: concepts and techniques. Elsevier.

[39] The seven types of e-commerce fraud explained – Information Age. Available from: https://www.information-age.com/seven-types-e-commerce-fraud-explained-123461276/.

[40] 3 Methods to Preserve Digital Evidence for Computer Forensics | CI Security. Available from: https://ci.security/resources/news/article/3-methods-to-preserve-digital-evidence-for-computer-forensics.

[41] GitHub – ozlerhakan/mongodb-json-files: A curated list of JSON/BSON datasets from the web in order to practice/use in MongoDB. Available from: https://github.com/ozlerhakan/mongodb-json-files.

[42] Scaling for the Future of Finance With Coinbase | MongoDB. Available from: https://www.mongodb.com/customers/coinbase.

[43] Shanthi E, others. A MongoDB based Performance Optimization Framework for Big Data in Cloud Environments.

[44] Cassandra deployment architectures. Available from: https://docs.axway.com/bundle/APIGateway_753_InstallationGuide_allOS_en_HTML5/page/Content/InstallGuideTopics/cassandra_architecture.htm.

Biographies

Rupali Chopade is a full time Research Scholar under AICTE-QIP scheme, at Department of Computer Engineering and IT, College of Engineering Pune, India. She is working as Assistant Professor at Department of Information Technology, Marathwada Mitra Mandal's College of Engineering Pune, India. She has 17 years of teaching experience. Her research interest includes database forensics and database security. She has received "Distinguished HOD "Award by Computer Society of India (CSI) in 2017.

Vinod Pachghare is Associate Professor in the Department of Computer Engineering and Information Technology, College of Engineering, Pune (An autonomous institute of Government of Maharashtra), India. He has 29 years of teaching experience and has published the books on Cloud Computing and Computer Graphics. Dr. Pachghare has over 37 research publications in various international journals and conferences. His area of research is network security. Also he is a member of Board of studies in Computer Engineering / Information Technology of a number of Autonomous Institutes. He is an Investigator for the Information Security Education and Awareness [ISEA] Project, Ministry of Information Technology, Govt. of India. He was a Principal Investigator for a research project "Wireless IDS", sponsored by AICTE, New Delhi. He delivered lectures on recent and state of the art topics in Computer Engineering and Information Technology as an invited speaker. He has received "Best Faculty Award" 2018 by CSI, Mumbai Chapter.