Symbiosis

www.symbiosisonline.org
www.symbiosisonlinepublishing.com

Research article Journal of Computer Science Applications and Information Technology Open Access

Dynamic Watermark Injection in NoSQL Databases

Vidhi Khanduja*

Department of Computer Engineering, Netaji Subhas Institute of technology, Delhi, India.

Received: October 18, 2016; Accepted: October 22, 2016; Published: January 02, 2017

***Corresponding author:** *Vidhi Khanduja, Department of Computer Engineering, Netaji Subhas Institute of technology, Delhi, India, E-mail: Vidhikhanduja9@gmail.com*

Abstract

With the advent of gathering real time information, the need of Not Only SQL (NoSQL) databases has skyrocketed. No prior work is done on watermarked protection of such Schema-less databases. Traditional watermarking techniques cannot be applied as NoSQL databases are dynamically increasing and often have irregularities in schema. In this paper, a new perspective of embedding watermark into such databases is proposed. Watermark that acts as a signature is securely prepared for each tuple individually. The proposal deals with the issues by leveraging the flexible schema features provided by NoSQL database. A new attribute is dynamically injected into the tuple before inserting or updating it in database. Experimental results and analyses demonstrate that with even the slightest modification of up to 5% alteration in tuples, the watermark recreated from the suspected database changes by as much as 20%, thus facilitating immediate detection with localization up to tuple level in the database.

Keywords: Fragile Watermarking; Tamper Detection; NoSQL databases; Big Data; Dynamic Watermark Injection;

Introduction

With the advent of high internet accessibility and electronic devices like smart phones & tablets, more and more data is collected and stored in databases. Cloud & Web services by Amazon, Microsoft and alike enabled developers to scale up by making use of distributed computing and clusters to gather much more real time information like sensor data from Internet of Things (IoT) devices, online game data, telemetry of user events on applications and websites, etc. Since the traditional database systems weren't designed or optimized for such enormous size of data with complex data types and structures, the need of Not Only SQL (NoSQL) databases increased [1]. These databases may or may not have features of traditional relational databases in favor of better horizontal scaling facilities, which is a problem for RDMS or having schema-less document based objects, which allow capturing complex structures such as data coming from several different sensors and also allowing faster access in some cases [2].

Not only the quantity of data has increased, but the value of the same has also skyrocketed. The data is used by big firms to perform scientific analysis using machine learning and artificial intelligence concepts to improve their products on basis of that data. Further, dynamic data analysis of such huge data requires Hadoop like map-reduce stacks [3]. For example, Netflix, an online video viewing subscription based platform relies on customer's viewing preferences to recommend better titles which are more relevant [4]. Face book tracks user interactions with news feed items, friends and pages, photos, text posts etc. to provide better features on its social networking platform.

Data in modern web services and applications thus plays really important role. This creates necessity of Technologically Protective Measures (TPM) to protect the company data related assets [5]. One technique popular in this field is of Watermarking. The core idea is to embed a watermark into tuples of database. Watermark is usually based on some secret string chosen by owner and hashing functions. However, static watermarking works on snapshots of databases and often uses techniques that are slow and can't be scaled for big data easily. Big data is dynamic in the sense that the rate of change in it is too high, and the size is too huge, which makes static watermarking on snapshots nearly impossible. This poses a challenge to traditional watermarking techniques in order to implement TPM on Big Data.

The dynamically increasing NoSQL databases with irregularities in schema require parallelizable and atomic watermarking techniques that deal with following problems:

(a) The technique shouldn't rely on schema specific details as NoSQL databases tend to change very often and with document store systems like MongoDB, CouchDB, RethinkDB etc, schema is flexible and different documents can have different structure.

(b) Technique shouldn't work on a snapshot of database but should work dynamically as and when data comes. This would mean the implementation should have minimum computational overhead to be worth.

(c) Technique should exploit new features of NoSQL databases like support for new data types and sharded/distributed databases that can't support inter tuple dependency for watermark embedding.

Considering these issues in mind, a technique is proposed to protect integrity of NoSQL databases. In this work, the emphasis is laid on the challenges faced while designing the technique and identifying the characteristics of watermarking techniques for NoSQL databases.

The rest of the paper is organized as follows. Firstly, prior work in the domain of watermarking relational databases is discussed. Secondly, the proposed database watermarking technique for NoSQL databases for tamper detection is presented followed by the experimental results and Integrity analysis. Lastly, the paper is concluded.

Prior Work

Agrawal and Kiernan [6] introduced the concept of robust watermarking of relational databases. Their technique embeds single bit watermark into secretly selected positions. Several other researchers have contributed in the domain of robust watermarking [7-10]. Watermarking schemes often require manipulation with a snapshot of entire database. Either some values of attributes are modified [7-9], or the order of tuples in the database is modified to embed a watermark.

Fragile watermarking techniques are also classified as distortion [11-12] or distortion-free techniques Guo et al [13-16]. Proposed the technique to embed and verify watermark group by group independently according to some secure parameters [11]. In this scheme two sets of watermarks are embedded into LSB's of the attributes of a tuple within a group to localize modifications made to the database. Recently, Khan et al. proposed another fragile technique [13]. The core idea of their technique is to generate watermark based on local characteristics of database relation like frequency distribution of digit, length and range. The watermark thus produced was not embedded into the databases; however, was secured with trusted third party for future reference. Techniques proposed in [14-16] embeds fragile watermark by changing the position of tuples within a database. Such techniques use a watermark string and create partitions using primary key and the binary string of watermark. The tuples in the partitions are then rearranged by comparison of their monotonicity, and relies on order of the tuples in the database to verify status of the database [17].

Literature shows several researches on XML documents as well [18-19]. In [18], authors extended the work of Agrwawal et al [6] on XML data by defining locators in XML. Another approach suggested by them, compresses the data before watermarking. This claims to achieve better data security. Clearly, these approach can't be applied to databases where;

(i) Data comes every minute or so, making snapshot nearly impossible to take.

(ii) Data is saved in document stores which don't maintain order of documents.

No prior work is done on watermarked protection of such Schema-less databases. To fulfill this gap, new perspective of embedding watermark into such databases is proposed. Our proposal deals with the above mentioned issues by leveraging the flexible schema features provided by NoSQL database. Since tuples need not have uniform schema in document-based databases, one can inject attribute(s) right before database operation to tuples in order to embed watermark. This gives us the required dynamic nature that can work on a live, constantly changing database.

The Proposed Watermarking Technique

In this paper, a tamper detection technique is proposed. The watermark creation is a function of owner decided parameters and tuple signature from sensitive attributes which one wish to track tampering. Figure 1 shows the flow diagram of the proposed watermarking technique.

For every incoming tuple, firstly, the watermark is generated. This process is discussed in detail further in the paper. Generated watermark is saved in a new attribute, i.e. a new attribute is dynamically injected into the tuple before inserting it to database or updating it in database. Figure 2 depicts this process. Injecting of watermark is followed by completion of tuple addition operation on database. Finally, Verification can be done by re-calculating the watermark and comparing it with existing value in the "injected attribute". Re-calculation of the watermark and comparison with the extracted value would be required to check whether the tuple was tampered or not.

Since the embedding process for one tuple is completely independent from another one, this approach can actually be applied in practice to databases where order isn't maintained for the documents. Further, isolation of each document allows us to perform verification using concurrent paradigms, thus resulting in near-real-time detection of tampering.

Implementation

Object Relational Mapping (ORM) layers in software stacks allow users to add programmatic hooks to operations like save, delete etc. This is where one should implement this watermarking scheme. Essentially, one can transform a tuple by adding a new attribute with some value. Pseudocode 1, **Add_Attribute** shows the process of inserting a tuple into DB by adding extra attribute containing watermark.

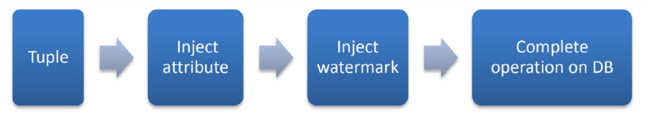

Figure 1: Flow diagram of the proposed watermarking technique

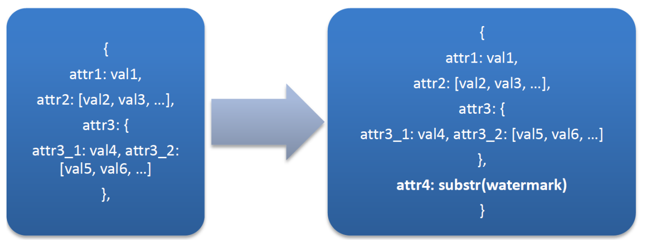
{
attr1: val1,
attr2: [val2, val3, ...],
attr3: {
attr3_1: val4, attr3_2: [val5, val6, ...]
},
{
attr1: val1,
attr2: [val2, val3, ...],
attr3: {
attr3_1: val4, attr3_2: [val5, val6, ...]
},
attr4: substr(watermark)
}

Figure 2: Addition of new attribute within a selected tuple in an Attribute injection process

Pseudocode 1: Add_Attribute

```
function insertTuple(tuple) {
  secret='secretkey';
  tuple[getInjectedAttributeName(tuple)] =
getWatermark(tuple, secret);
  insert(tuple);
        }
```

In order to use this proposal, owner would have to define following functions.

getInjectedAttributeName (tuple)

The function returns the name of the attribute that is to be injected into the tuple. Work can be done in deciding an attribute name in order to avoid obvious patterns that can help an attacker to suspect the attribute to have non application related usage. More the names match to application's domain, easier it is to deceive the attackers.

Implementation of this can be done as elucidated in Pseudo code 2, assuming data-type of watermark would be integral. Names are an array that holds list of various possible attribute names and Count. Names tell the number of elements in names. Use of secret key in selection condition of new attribute adds a level of security to the process [20].

Pseudocode 2: getInjectedAttributeName (.)

```
function getInjectedAttributeName(tuple)
        {
        const names = ['tweet_media_id', 'user_relation_id', 'tweet_location_id'];
 return names[(tuple.id+secret) % count.names];
}
```

One key point that is to be kept in mind while implementing the function is that the logic for choosing a name should be reversible with the help of tamper-free tuple data so as to detect modifications.

getWatermark (tuple, secret)

This function is arguably the most important part of entire proposal. Owner must implement this carefully in such a way that an attacker can't reverse the process of creating the watermark by observing patterns. The function returns watermark of decided data-type keeping dynamically injected attribute name in mind. A basic strategy that can be employed is as follows in Pseudo code 3. Watermark is prepared by concatenating secret key with the value of attributes in that tuple and then taking hash of it. Attributes in database are of varied data type. Each attribute of a tuple is processed and converted to real number and then concatenated to prepare signature.

To maintain security of the algorithm, secret key is used. Substring of hash is taken as the value of newly added attribute. Many cryptographic hash algorithms exist in literature, e.g. MD5, RIPE-MD, SHA-2, SHA-3, SNEFRU, etc. We implement SHA-2 algorithm as a cryptographic hash function that yields 256-bit hash value owing to its improved resilience against attacks [21-22]. Hash possesses a strong avalanche effect. Even with a single-bit change in input, large number of bits changes in hashed output. Hence, it is difficult to guess input given the output of the secure hash function.

The signature is prepared using all attributes of the tuple. However same can be modified by concatenating only crucial attributes. In such case, temperedness in participating attributes is detected. This may be used for large databases with large number of attributes; where information of few attributes is crucial and requires protection.

Pseudocode 3: getWatermark(..)

```
function getWatermark(tuple, secret) {return parseInt( hash(secret
     + tuple.id + tuple.attr1 + tuple.attr2 + .....+ tuple.attrN_a + secret)
             .substring(0, 16), 16 );
   }
```

Tamper Detection

For fragile watermarking, signature using crucial or all attributes of the tuple has to be generated. The key idea is to make the process fast, reversible with randomness. A fairly straight-forward implementation is mentioned below in pseudocode 4. The function returns a Boolean. It simply recalculates the watermark for a tuple existing in database, and compares it with the one concealed in the injected attribute. If they match, then the database was preserved, i.e. no tampering occurred. However, if they don't match or the attribute itself is missing from the tuple, it would mean that the database was tampered in a particular tuple. Thus, localization up to tuple level is attained.

The proposed technique employs two security levels to complete the entire watermarking process. Firstly, watermark is prepared securely using a secret key making difficult for an attacker to crack the watermark. Secondly, a new attribute where a watermark is embedded is chosen using a secret parameter.

Pseudocode 4: Tamper Detection

```
function verifyWatermark (tuple, secret)
 {
           return tuple[getInjectedAttributeName(tuple)] ===
     getWatermark(tuple, secret);
   }
```

Experimental Results and Analysis

Experiments are performed on the dataset that contains the preprocessed and filtered session data for DePaul CTI web server [23]. The data is based on a random sample of users visiting this site for a two-week period during April 2002. The database contains 20509 tuples with varying attributes of maximum limit 10.

Proposed technique is tried on [23] and the performance loss one would achieve due to overhead of watermarking is tested. Results recorded in table 1 show that the change is nominal.

Integrity Analysis

Next, the experiment to check the robustness of proposed technique is performed. The attribute values are altered and the change in regenerated watermark with the extracted one is recorded. The graph in figure 3 shows the changes in extracted watermark by altering attributes that does not contain watermark. Since, I have applied hash of the concatenated attribute values of a tuple; even a single bit change in input will result in change of large number of bits which can be verified from the graph. With 5% change in tuple, the change in watermark extracted is 20% which further increases on increasing the alterations in tuple.

Let us consider the following cases that may arise. We take W_s as original signature watermark that was embedded, W_g as regenerated watermark from suspected tuple and W_x as watermark extracted from suspected tuple.

(a) There were no integrity attacks. If neither the attributes nor the watermark were changed, then $W_g == W_x$. Thus, the two watermarks will match stating no perturbations in a tuple.

(b) The content of the any of the tuple attribute was changed but not the embedded watermark. In this case, the re-generated watermark W_g will not be the same as the original watermark W_s. Thus, the re-generated watermark will not match the extracted watermark, i.e. $W_x \neq W_g$ and the tampering event will surely be detected.

(c) The positions where watermark is embedded was tampered while other attribute values were not changed. In this case, the extracted watermark W_x will not be the same as the original watermark W_s. Thus, the re-generated watermark will not match the extracted watermark, i.e. $W_x \neq W_g$ and the tampering event will surely be detected.

Table 1: Comparison of tuple insertion time with and without watermarking

Database Size (No. Of Tuples)	Insertion Time (Without Watermarking)	Insertion Time (With Watermarking)
10,000	0.523s	0.581s
50,000	0.601s	0.640s
100,000	0.612s	0.649s

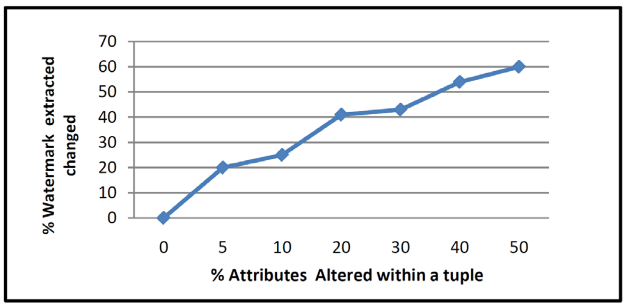

Figure 3: Change in extracted watermark by altering the different attribute values

(d) Both the watermark bit positions and attribute values was changed. This case has a very remote chance of the two new watermarks produced as a result of the changes to attribute values and the inserted watermark respectively, turns out to be exactly the same. Hence $W_x \neq W_g$ and tampering is detected.

From the above, it is clear that the watermark is highly fragile. Any changes made to the dataset that affects the redacts and/or rules or the embedded watermark or both can be immediately detected.

Conclusion and Future scope

The proposal exploits the option to have schema-less database in NoSQL to inject an attribute to tuples and save the watermark in it. The name of the attribute and its value can be dynamically modified to increase randomness and thereby security. No dependency of watermark for a tuple on other tuples enables the scheme to be used in distributed/sharded systems. Further, absence of sharded data leaves space for concurrent operations for quick runtimes of the watermark embedding phase. Experimental results and analysis proves the fragility of our watermark against perturbations.

The same framework can be extended to work for embedding watermark such that it can help in proving ownership of a particular database. The idea is to make use of several owner secrets that can't be reproduced probabilistically by a data-thief. So by only tweaking the get Watermark function, the same framework can be extended to serve as a technique for ownership verification.

Acknowledgements

The author wishes to thank Mr Divjot Singh for his collaborative efforts towards the work present in this paper. He was in department of Computer Engineering at Netaji Subhas Institute of Technology, Delhi, India.

References

1. Parker Z, Scott P, Vrbsky SV. Comparing nosql mongodb to an sql db. 51st ACM Southeast Conference. 2013.
2. Kaur, K. and Rani, R. Modeling and querying data in NoSQL databases. IEEE International Conference on Big Data. 2013:1-7. DOI: 10.1109/BigData.2013.6691765.
3. Dittrich J, Quiané-Ruiz JA. Efficient big data processing in Hadoop MapReduce. Proceedings of the VLDB Endowment. (2012);5(12):2014-2015.
4. Netflix. Available from: https://www.**netflix**.com/
5. Maggon H. Legal Protection of Databases: An Indian Perspective. Journal of Intell Prop Rights. 2006;11:140-144.
6. Agrawal R, Haas PJ, Kiernan J. Watermarking relational data: framework, algorithms and analysis. VLDB J. 2003;12(2):157-169. DOI: 10.1007/s00778-003-0097-x.
7. Farfoura ME, Horng SJ, Lai JL, Run RS, Chen RJ, Khan MK. A blind reversible method for watermarking relational databases based on a time-stamping protocol. Expert Systems with Appl. 2012;39(3):3185-3196.

8. Khanduja V, Verma OP, Chakraverty S. Watermarking Relational databases using Bacterial Foraging Algorithm. Multimed Tools & Appl. 2013;74(3):813-839. DOI: 10.1007/s11042-013-1700-9.

9. Ifthikar S., Kamran M. and Anwar Z. RRW-A robust and reversible watermarking technique for relational Data. IEEE Transactions on Knowledge and Data Engineering. 2015;27(4):1131-1145. DOI: 10.1109/TKDE.2014.2349911.

10. Khanduja V, Chakraverty S, Verma OP. Watermarking Categorical Data: Algorithm and Robustness Analysis. Defense Science Journal. 2015;65(3):226-232.

11. Guo H, Li Y, Lui A, Jajodia S. A fragile watermarking scheme for detecting malicious modifications of database relations. Information Sciences. 2006;176(10):1350–1378.

12. Khataeimaragheh H, Rashidi H. A Novel Watermarking Scheme for Detecting and Recovering Distortions in Database Tables. International Journal of Database Management Systems. 2010;2(3):1-11.

13. Khan A, Husain SA. A fragile zero watermarking scheme to detect and characterize malicious modifications in database relations. The Scientific World Journal. 2013:1-16.

14. Li Y, Guo H, Jajodia S. Tamper detection and localization for categorical data using fragile watermarks. ACM workshop on Digital Rights Management. 2004:73-82.

15. Camara L, Li J, Li R, Xie W. Distortion-Free Watermarking Approach for Relational Database Integrity Checking. Mathematical Problems in Engineering. 2010;2014:1-10.

16. Kamel I. A schema for protecting the integrity of databases. Computers and Security. 2009;28(7):698-709.

17. Khanduja V, Chakraverty S, Verma OP. Ownership and Tamper detection of Relational Data: Framework, Techniques and Security Analysis. Published as the chapter in the book titled: Embodying Intelligence in Multimedia Data Hiding at Science gate Publishing. 2016:21-36. DOI: 10.15579/gcsr.vol5.ch2.

18. Wilfred Ng, LauHL. Effective Approaches for Watermarking XML Data. LNCS. 2005;3453:68-80.

19. Chen L, He W, Shu H, You FC. Research on the Method of Text Information Hiding Based on XML. Applied Mechanics and Materials. 2013;385-386:1665-1668.

20. Khanduja V, Chakraverty S, Verma OP. Enabling information recovery with ownership using robust multiple watermarks. Journal of Information Security and Applications. 2016;29:80-92. DOI: 10.1016/j.jisa.2016.03.005.

21. Ristic I. sha1-deprecation -what- you-need -to-know. 2016. Available from: https://blog.qualys.com/ssllabs/2014/09/09/sha1-deprecation-what-you-need-to-know

22. Wikipedia. SHA-2. 2016. Available from: https://en.wikipedia.org/wiki/SHA-2.

23. Depaul University: College of Computing and Digital Media. Online dataset. 2016. Available from: http://facweb.cs.depaul.edu/mobasher/classes/ect584/resource.html.