Graph database watermarking using pseudo nodes

Tsvetomir Hristov[1]

Supervisor(s): Dr. Zekeriya Erkin[1]**, Devriş Işler**[2]

[1]EEMCS, Delft University of Technology, The Netherlands
[2]IMDEA Networks Institute, Universidad Carlos III de Madrid, Spain

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 29, 2023

Name of the student: Tsvetomir Hristov
Final project course: CSE3000 Research Project
Thesis committee: Dr. Zekeriya Erkin, Devriş Işler, Dr. Petr Kellnhofer

Abstract

Although digital watermarking has been a well-researched topic for the past decades and has seen numerous implementations for relational databases, it still lacks research for non-relational schema-less databases. [1] In this paper, we explore proposed techniques for non-relational database watermarking and introduce an improved technique for NoSQL database watermarking. The improved technique produces pseudo documents, embeds a watermark in them, and inserts them into the database. Then it modifies the relations between the pseudo and real documents to indicate which documents are covered by the watermark. Lastly, the pseudo documents act as a cover for the watermark embedding, proving the authenticity of the documents connected. We tested the technique against numerous modification and deletion attacks. The results show that the introduced technique does not improve the robustness of the watermark. Although the technique has the potential to produce a robust watermark, it still needs to be improved before being used in a commercial setting.

1 Introduction

Today, an enormous amount of information is shared over public channels, such as the internet. However, the source of this information can rarely be verified, allowing for a high amount of unauthorized or fake data to be marketed as authentic and reliable. This creates an environment of mistrust where the receiver cannot easily verify or check data. [2] Numerous techniques have been developed to tackle this problem, such as encryption; however, they require either a lot of resources or pre-negotiated information. [3] Digital watermarking techniques tackle such issues. A watermarked document, or a part of it, can prove its integrity and, ultimately, be used as evidence for the authenticity and ownership of the data.

Watermarking [4] is a well-established and often-used technique for protecting and validating intellectual property. Hidden inside the information, it provides an easy-to-use yet very effective technique for validating the source of the database, and its contents, while also proving to be hard to find, forge or modify without a secret key. Watermarking can accurately indicate and verify the ownership, authenticity, and integrity of data, whether the information is transferred over an open channel and needs to be validated; or leaked from a closed, confidential channel. Although watermarking for copyright protection and ownership verification has seen extensive research, it still cannot keep up with the ever-growing database management field. This is made apparent by the lack of standardized, well-researched watermarking algorithms for graphical database management systems (DBMS).

Databases can further be separated into two main categories: relational or sequence, databases, and non-relational. Until now, relational databases have been mainly used in production environments, which is why these systems have also been researched more. However, in recent years, non-relational databases have been steadily growing in popularity due to their lack of schemas, high distributively, and adaptivity [5][6].

Unfortunately, this rise in popularity has only recently been matched by increased research on the topic. The lack of research could allow for algorithms that are not well-tested and proven to work to be used in real-world environments. Thus, it is essential for an algorithm to be well-researched and tested by numerous researchers before it is used in the real world. This paper introduces an improved method for watermarking by identifying and improving upon flaws inside a recently proposed watermarking technique.

The aforementioned technique and how it works are presented in section *Technical background*. The sections *Methodology* and *Method* discuss the analysis of the previous method and how they were used to form a new method for database watermarking. The *Analysis* section then talks about the analysis performed on the new method, alongside their results, compared to the previous method. Finally, the paper concludes with a *Conclusion* section and *Further research*.

2 Technical background

For the purpose of this research, a popular graph database called Neo4j [7] is used. This chapter juxtaposes how a graphical database works and its SQL counterpart. It also introduces terminology which is utilized throughout this paper.

First and foremost, SQL and NoSQL databases vary by terminology: an SQL database consists of tables, which contain rows, also called records, and columns. On the other side, NoSQL databases consist of collections containing documents that store information into key/value pairs, as explained in figure 1.

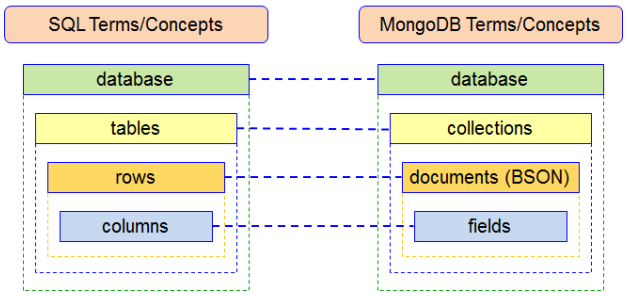

Figure 1: Sketch showing the difference in terminology between SQL and NoSQL database, published by Studio 3T, a GUI tool for Mongo [8].

A typical SQL database stores all its information in tables, where each table has a well-defined schema, and each entry must abide by this scheme. A NoSQL database does not enforce a scheme, which makes watermarking harder, as there is no guarantee that a document follows a specific data

model. Furthermore, NoSQL databases vary by how they store relations between documents. A typical SQL database uses a table to store relations between records, whereas a graph database uses edges, and a document database uses nesting/embedding. Each edge between two nodes, or documents, is directed and can store additional data about the relation.

3 Methodology

In this research, we aim to improve the current techniques for graph database watermarking by introducing a novel technique designed explicitly for graph databases. The technique works by introducing imperceptible fake documents that carry a watermark. Then, we implement the algorithm and test our system based on popular metrics used in the field. Lastly, the results are compared against results from previous algorithms.

A literature study is performed to find relevant watermarking techniques on NoSQL databases. The search uses two search engines and numerous keywords: combined with AND, OR, and NOT operators. The search parameters are shown in Table 1. The resulting papers are manually scanned and the algorithms proposed are extracted. The steps of the algorithms are then analyzed to determine the techniques used for the watermarking. Although this "classification" step does not help the analysis of the algorithms, it helps understand how an algorithm works and why it works that way.

The literature study identifies an algorithm to be analyzed and improved. As part of the analysis, a thorough inspection of the method is performed, finding critical assumptions and possible security vulnerabilities. These analyses are then used to hypothesize attack techniques to which the algorithm might be susceptible and improve the algorithm, so those attacks do not work anymore.

Lastly, improvements to the algorithm are implemented and tested to determine how the applied changes influenced the security of the watermark.

4 Literature study

The literature study identified several watermarking techniques which could be used for graphical bases. [9] [10]

We present an algorithm based on a promising algorithm for watermarking NoSQL databases introduced in [10]. The algorithm watermarks a MongoDB database using connected graphs.

4.1 Steps of the previous algorithm

The algorithm introduced in [10] watermarks a MongoDB database using connected graphs. The database is presented as a graph, where references between documents as edges and documents as nodes. Firstly, pseudo documents are created using a genetic algorithm. Then, a watermark is embedded into the pseudo nodes and they are added to the database. Lastly, the documents are connected into a k-connected graph.

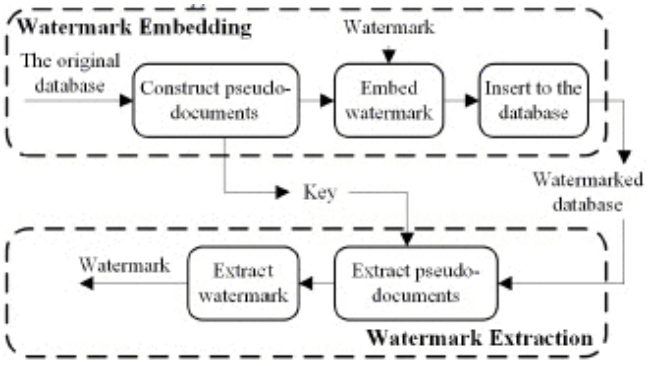

Figure 2: Chart of the previous algorithm in [10]

4.2 Analysis on a previous watermarking method

Firstly, the algorithm [10] was carefully examined for important assumptions and possible attacks other than the ones stated in the paper. The analysis presented below is vital for breaking the watermarking technique and serves as the main point for improving the algorithm.

Attack on the primary key

By default, every document in the database should have an `_id` field, which uniquely identifies each document [11]. As the field is randomly generated upon insertion, changing its value of it would not change the data stored inside the database as long as the relations between objects are updated. Any watermark, depending on this field, would then be undetectable. Similar attacks have been observed on relational databases with the primary key instead of the `_id` field; however, techniques for evading this attack are yet to be researched.

Attack on the genetic algorithm

All genetic algorithms are designed to tackle complex problems in optimization and search algorithms [12]. This means that, fundamentally, these algorithms are trying to optimize the fitness function that they are given. Given this information, an attacker should be able to distinguish pseudo documents from real documents, provided how close they are to a local or global minima/maxima.

Limitation to the genetic change

To stop the genetic algorithm from changing a character too much, changing an English letter to a non-English one, for example, the algorithm's creators have chosen to only change a certain number of characters. That, however, is not a good practice, as it guarantees that a set of letters remain unchanged, setting a maximum possible Hamming distance [13] per text. This distance can be used to easily find similar records, which can be inspected to recognize the watermarked documents further.

Graph creation

The method groups pseudo documents into groups, which are then connected to form a k-connected graph. This is a novel approach for database watermarking; however, it gives another attack method: graph analysis. Analyzing the graph

should show multiple groups perfectly connected with k-connections each other. However, this is rarely observed in real life. This makes the pseudo documents stick out, which makes them even easier to remove, as every document connected to a pseudo document, is a pseudo document.

5 Algorithm

We present a novel algorithm inspired by the graph method introduced in [10] but designed to tackle the problems pointed out in the previous sections of this paper.

5.1 The watermarking technique

The new technique borrows existing watermarking techniques to implement a robust watermarking scheme for graphical databases. The embedding method consists of several well-defined steps visualized in figure 3.

The documents from the database are first divided into groups, each varying in size. Then, a pseudo document is added for each group, which carries a watermark. Then, each pseudo document is linked with all documents in the group. Lastly, the documents are inserted into the database.

Dividing into groups

The first step of the algorithm is to divide the nodes into groups. For this purpose, an algorithm is used to find a random list of integers whose sum adds up to a number. For example, if the algorithm is run with the range (3,5) and a target sum of 15, a viable solution is [3,4,3,5]. The algorithm is then given the min_group_size and max_group_size parameters as the range and the number of nodes, which need to be watermarked, as the total sum. The output is then used to partition the nodes into groups. It is essential to mention that the minimum and maximum size parameters should have a significant enough difference between the two. This should prevent an attacker from analyzing a small portion of the database and potentially finding a watermarked group. For more clarity, a pseudo code of this step is provided in algorithm 1 and a figure visualizing this effect figure 8.

Adding a pseudo document

Numerous techniques have been extensively studied [14] for the creation of pseudo documents; however, as this step only helps with making the watermark imperceptible, a simple algorithm is considered for the creation of such documents. Pseudo documents are created by "borrowing" fields from other real documents. To determine which fields are "borrowed" a schema analysis is manually performed beforehand, which is explained in more detail in the next section. For each group, the following tasks are performed:

1. Determine which fields a document have
2. Randomly choose several "donor" documents from whom some field values are taken
3. Create the pseudo document from the "donor" document's data

The algorithm is further explained with pseudocode in the Appendix 3.

A new pseudo document is added to each group of real documents, hosting information about the group. After adding, the pseudo document is linked with each document inside the group.

Linking documents

The linking of the pseudo document and the other documents inside the group is a process that can be made more complex with the complexity of the database schema itself. Every node can have one or multiple labels attached, and every edge must have a type. This is why it requires manual analysis to determine what types of documents should carry the watermark and with which types of edges these documents should be connected. Ideally, the manual analysis would produce a set of documents, which could be used for watermarking, and a list of node types, which should be used to connect the pseudo node and the other nodes. To prevent anomalies from being introduced to the dataset, the algorithm can also accept weights, specifying how probable it is for two nodes to be connected with a certain type of edge. An algorithm without weights is provided in Appendix 4.

Embedding the watermark

To embed a watermark into the document, a well-established method of embedding the watermark using the least significant bit is used. Bits from different variable types, such as dates, floats, etc., can be used to store information, which can then be used to verify the watermark. The information stored inside the watermark can also include part of the node's data or the connections it has with other nodes.

As the embedding of information is highly dependent on the capacity of the cover, the amount of information may vary based on the available space. For now, the amount of connections is considered the most critical, which is why it is embedded first. After that, its integrity and its creator(s).

Adding watermarked documents to the database

As the private key is used to hint about the location of the pseudo documents, several ways to add a watermark inside the database can be considered based on the way the database stores the documents. For example, the primary key for a MongoDB database is a random string; however, the primary key for a Neo4j database is an integer.

This research focuses on embedding documents into a Neo4j database; however, similar techniques can be used for other NoSQL databases. To add all watermarked documents inside the database, the amount of all documents, including the pseudo ones, should be known. This can be computed for all groups G_i using the following formula:

$$N = \sum_{G_i \in G} |G_i| \tag{1}$$

where G is the set of all groups and $|G_i|$ means the size of group G_i.

The private key K is put in a Pseudorandom number generator (PRNG) to generate the primary keys of documents. The generator's output is treated as a 32-bit unsigned integer, as Neo4j's id field uses the same type [15]. For each integer i from the generator, the primary key PK of the next document is calculated using the modulo function:

$$PK = i \mod N \tag{2}$$

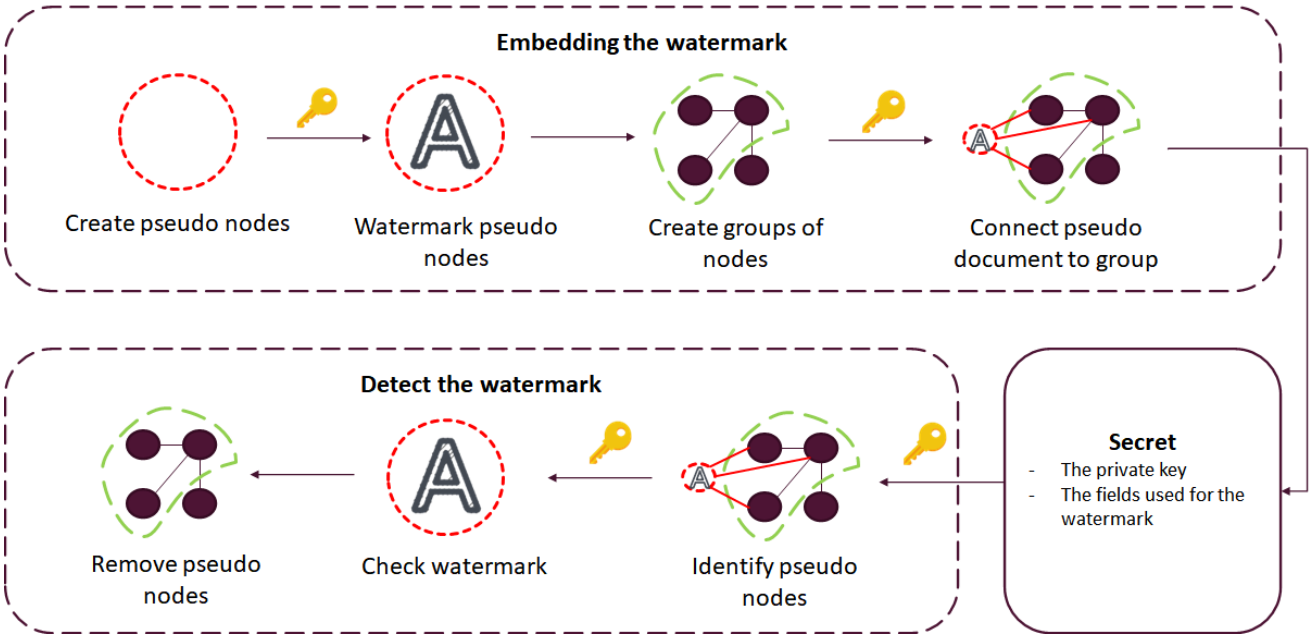

Figure 3: Watermarking method

To resolve possible collisions, if there is a document at that place already and the document is not a pseudo document, a swap takes place, where the pseudo document takes the primary key of the real document, and the real one takes the next free primary key.

5.2 Extracting the watermark

The private key is input into a PRNG to extract the watermark, and the output is treated as 32-bit unsigned integers. The integers are again put tough the same formula $PK = i \mod N$ which gives a list of locations of nodes inside the database. These nodes are checked for a watermark. As the list of possible pseudo documents can also have real documents at the end of it, the list is checked incrementally until five documents have been observed consecutively without a watermark. Documents that are missing are not counted, as this could indicate a possible attack on the watermark.

6 Experimental Setup and Results

6.1 Attacker model

To attack the algorithm, a clear model for an attacker needs to be established. The model defines an attacker's role, resources, goals, and methods.

Attacker

An attacker is considered any adversary who tries to purposefully find, delete, modify or interact in any other way with the watermark. Due to this definition, an attacker does not have to have malicious intent but to knowingly and purposefully interact with the watermark. Additionally, an attacker does not have infinite resources or knowledge that can be used to break algorithms, which are considered cryptographically secure. [16]

Goals

The ultimate goal of the attacker is to find all watermarked nodes. These goals can be categorized into two groups: the attacker may try to find the watermark in the graph using graph analysis or brute force the private key. The first attack uses analysis of the properties of the graph alongside analysis of the data, which is stored inside the graph. Theoretically, it would allow an attacker to delete the watermark or insert new documents inside the database. The latter technique is more computationally intensive; however, it would allow an attacker full control over the watermark.

Another goal, which can be considered by an attacker, is wiping the information which contains the watermark. Although possible, this method can also delete a substantial amount of truthful information from the database, which would decrease its value. Due to this, this goal is not considered in this paper.

Methods

For this model, it is assumed that the attacker already has full access to the public data, which might also be just a part of the full database. The attacker also has access only to one watermarked copy of the database and does not have access to the original database or the key for the watermark. Lastly, an important assumption is that the attacker cannot check the watermark, as in an oracle attack, to check if the watermark is still valid. [17]

The attacks tested are deletion and modification. A python script, acting as an attacker, is given access to an already watermarked database for deletion, and modification attacks. The script then performs an attack and hands the control over to the verification algorithm, which then tries to verify the watermark. If the verification process does not detect modifications or does not detect the watermark, then the attack is

Algorithm 1 Divide into groups

function GENERATEGROUPSIZES($target\_sum, lower\_range, upper\_range, max\_tries$) ▷ Generate a set of random numbers, all summing up to a target sum
  if $max\_tries <= 0$ **then** ▷ Base case: check if the maximum tries count was exceeded
    $Exit$("No groups were found satisfying the parameters")
  end if
  $returnList \leftarrow []$

  while $target\_sum - Sum(returnList) > upper\_range$ **do** ▷ Fill the list with random numbers, until you hit the upper bound of the range
    returnList.append(randint(RandomNumber())
  end while
  $remainder \leftarrow target\_sum - Sum(returnList)$
  if *remainder* $< lower\_range$ **then** ▷ If the number is not in the range, run the algorithm again
    $returnGenerateGroupSizes(target\_sum, upper\_range, lower\_range, max\_tries - 1)$
  else
    returnList.append($remainder$)
    return $returnList$
  end if
end function

successful. For the final attack, a series of values are computed for the graph, based on which the script tries to delete the watermark. After that, the same verification process is performed to see if the attack was successful.

For an attack, a score is calculated based on the percentage of successful detection attacks. The following formula is used:

$$\%_{success} = \sum_{A_u \in A} \frac{A_u}{A} \tag{3}$$

where A_u denotes the number of unsuccessful attacks and A denotes the number of attacks performed.

6.2 Experimental Setup

Codebase

An implementation of the algorithm was written in python and published on GitHub under the MIT license for public use. [18] Although the code was written to exactly match the algorithm in the paper, it is important to note some differences between the algorithm and its implementation in python. Since the database does not allow a document to change its primary key, the algorithm embeds the pseudo document without taking into account the primary key of the newly created document. Two methods could circumvent this: copy-pasting real documents and exporting and re-importing the database; however, currently, it is not considered a vital detail for testing the algorithm. This is why the codebase is also designed to be scalable and futureproof, making it easy to improve or completely rewrite the steps of the algorithm.

Database

To test the watermarking technique, a public dataset is used. The dataset, named UK Companies, is part of the default example datasets by Neo4j.[19] The dataset contains information about public companies in the United Kingdom, their properties, owners, and political donations. Figure 4 shows a constructed schema of the database.

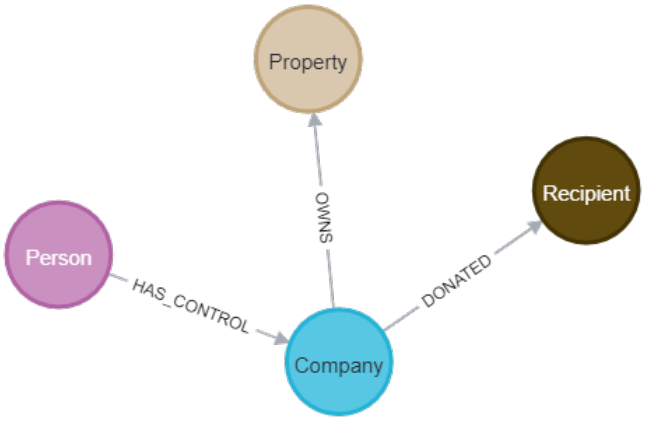

Figure 4: UK Companies Database Schema

As shown in figure 4, most documents share an edge with a document of type *Company*. Thus, every node of type *Person*, *Property*, or *Recipient* is watermarked by a node of type *Company*. In turn, watermarking nodes of type *Company* is done by pseudo nodes from the other three types. This ensures that every node inside the database can be watermarked.

Due to the schematic nature of NoSQL databases, there is no defined structure for each document. However, a lot of the documents still share the same fields. An example of a Company document can be seen below:

```
{
    "companyNumber": "04179322",
    "name": "CURO TRANSATLANTIC LIMITED",
    "mortgagesOutstanding": 1,
    "countryOfOrigin": "United Kingdom",
    "incorporationDate": "2001-03-14",
    "category": "Private Limited Company",
    "SIC": "64999 - Financial intermediation
        not elsewhere classified",
    "status": "Active"
}
```


Figure 5: Example of a company document from the database

Setup

Two types of setups are used for the experiment: the first focuses on the algorithm's performance, while the second performs an analysis of the algorithm itself. Additionally, each setup is performed inside a virtualized environment using Docker [20] to guarantee minimal interaction between the different setups and internal processes inside the computer.

The first setup uses multiple ways to monitor the time and resources consumed by the watermarking algorithm. Log files are analyzed for the average time for each group to be watermarked, and monitoring software is attached to the docker container to monitor resource consumption. Delays and resource usage of the database is not considered since it is assumed that the database performs similarly for each test and is hosted locally. The results are manually analyzed, and the speed of the algorithm is deducted using the following formulas:

$$speed_{document} = \frac{N_d}{t} \tag{4}$$

$$speed_{group} = \frac{N_g}{t} \tag{5}$$

where t is the time in seconds it took for the algorithm to finish, N_d is the number of documents in the database, and N_g is the number of groups that were formed by the algorithm. The formula measures the number of documents watermarked per second - $speed_{document}$; and the number of groups watermarked per second - $speed_{group}$. Since the algorithm randomly partitions the documents into groups and the most work is done per group, $speed_pergroup$ is considered a more important metric for performance. Additionally, the watermark verification is tested using the same formulas for embedding; see equation 4.

Additional testing is conducted on the impact of parameters on the general performance of the algorithm. Parameters such as min_group_size, max_group_size, and $watermarked\_document\_fields$ are considered to have a high impact on the performance.

The second setup focuses more on evaluating the watermark in terms of security, robustness, and usability. The security and robustness of the algorithm are evaluated using a deletion and modification attack. At the same time, usability is assessed by monitoring the number of pseudo nodes introduced and their influence on common characteristics of the dataset.

The robustness of the watermark is tested with a deletion attack. First, a script, acting as a malicious actor, randomly picks nodes and deletes them. After that, a verification script is run to determine the amount of watermarked nodes that were deleted or not detected. This sequence continues until the verification script does not detect the watermark anymore. The algorithm is explained in more detail in the appendix 5. At every step, the amount of deleted nodes and the percentage of watermarked nodes is recorded:

$$\begin{aligned} \%_{deleted} &= \frac{N}{N_i} \\ \%_{watermarked} &= \frac{N_w}{N_w i} \end{aligned} \tag{6}$$

where N is the number of nodes in the database, N_i is the number of nodes initially in the database, N_w is the number of discovered watermarked nodes, and $N_w i$ is the initial number of watermarked nodes.

The security of the watermark is tested with a modification attack. A script, acting as a malicious actor, randomly selects several documents whose values are modified 6. After modifications, a verification script verifies the watermarked nodes. The sequence continues until the watermark is not detected by the verification script anymore. At each iteration, the amount of modification and the percentage of watermarked nodes is recorded:

$$\begin{aligned} F &= \sum_{d \in D} f_d \\ \%_{modified} &= \frac{F_m}{F} \\ \%_{watermarked} &= \frac{N_w}{N_w i} \end{aligned} \tag{7}$$

where f_d is the number of fields in document d, D is the set of all documents inside the database, F_m is the number of modified fields, N_w is the number of discovered watermarked nodes. and $N_w i$ is the initial number of watermarked nodes.

The usability of the watermark is evaluated by several key factors: the number of pseudo documents added, the average, mean, and median of certain numerical values, the number of edges per node, etc. To ensure the usability of the database after the watermark, manual analyses are performed on predefined values, looking for changes introduced by the watermark.

To ensure that the different test setups do not interfere with each other, each test is performed inside an isolated docker container. This guarantees that there is one database instance per test, which cannot be accessed from the outside, and that every database is identical at the start of each test. For each test, a container for the database is started, after which the container with the python script is started. The container outputs its findings in a log file, logging results, errors, and other information related to this research.

Host

Although the tests are run inside a virtual environment, the performance of the algorithm highly depends on the specifications of the host. The algorithm was run on a PC with a

12-core processor, 16GB of RAM, and a 64-bit Windows 11 operating system; more information about the host's specifications is located inside Appendix 5.

6.3 Results

To generate results, the tests were run 67 times, each test isolated in a new virtual environment. After each test, the log files are sent back to the host, before the environment is powered down and deleted.

Performance

The algorithm's performance was measured in the number of seconds the algorithm took to watermark all groups. Because the number of groups is not fixed, we noted the number of groups that needed to be watermarked and the time it took to watermark them (figure 6). The number of watermarked

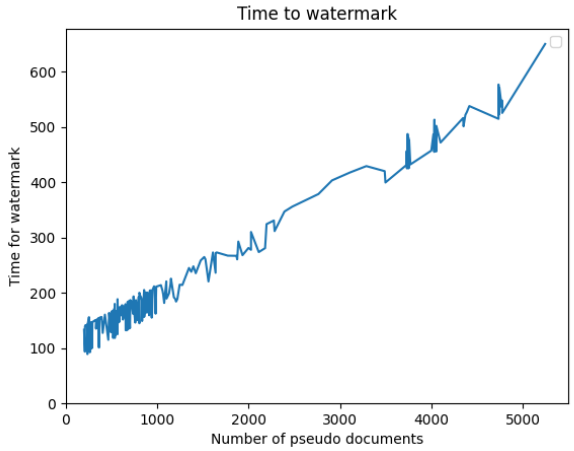

Figure 6: Performance of the algorithm

groups is plotted on the x-axis, while the time it took for the algorithm to watermark all groups is plotted on the y-axis. The graph shows that the amount of groups that need to be watermarked directly impacts the algorithm's performance. The plot also shows that the algorithm follows a linear time complexity, indicating that the algorithm can be scaled for large databases without compromising performance. Additional calculations show that the implementation watermarks 5.28 groups per second on average. It is important to note that the algorithm does not have multi-thread support, which could significantly increase its performance.

Usability

Usability is measured by calculating the number of pseudo nodes introduced and dividing it by the number of total nodes, essentially obtaining the ratio of pseudo nodes inside the database. To visualize this, a graph is created where the ratio is plotted on the x-axis while the percentage of nodes that needed to be deleted to erase the watermark is plotted on the y-axis (figure 7). The graph shows the number of nodes that should be introduced to achieve a certain level of robustness. Unfortunately, even with as much as 8 percent of the nodes being watermarked, the algorithm still under-performs, compared to the one introduced in [10].

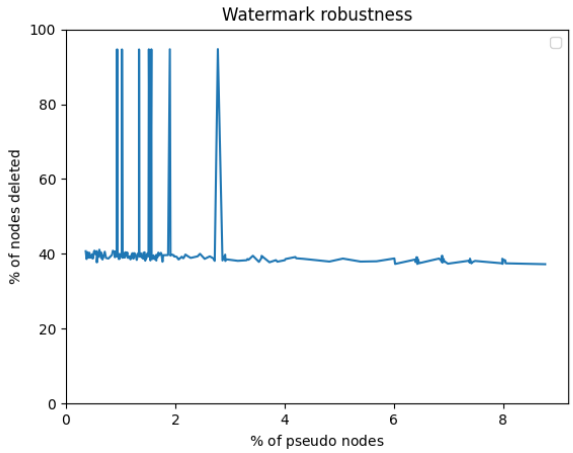

Figure 7: Watermark usability

Parameter impact

Test to measure the impact of the parameters on the performance; the input parameters were recorded alongside the time it took for the algorithm to watermark the database.

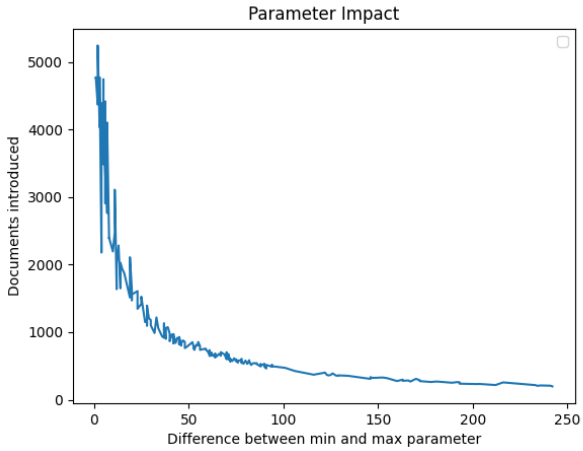

Figure 8: Impact from parameter choice on watermarking performance

Figure 8 indicates that the choice of parameters highly impacts the algorithm's performance. This is related to the grouping of the nodes, as smaller numbers for both parameters lead to more groups being created, leading to more work for the algorithm. A 3D plot showing this effect can be found in Appendix 10

6.4 Deletion and modification attacks

Both a deletion and a modification attack are performed to test the security and robustness of the algorithm. The results are then plotted according to the scoring system discussed earlier with equations 5 and 6, where the number of watermarked nodes is plotted along the x-axis, and the score of the algorithm is plotted along the y-axis.

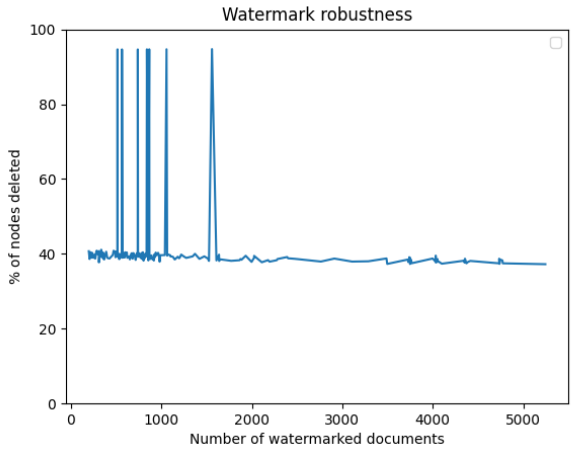

Figure 9: Deletion attack results

The deletion attack shows that the watermark can withstand around 40% of documents being deleted before becoming undetectable. This result is good for the range between 0 and 1000 watermarked documents; however, the score remains constant when the number of watermarked documents increases.

7 Responsible research

Watermarking techniques could pose a threat to user privacy and security. Security and privacy software often work on the communication channel to ensure the user stays anonymous. However, as this technique could open a new covert communication channel unseen by security and privacy protection software, it could pose a risk of sending identifiable data over it. For example, a rogue program purposefully watermarks every piece of information using an identifiable string as a key. As the watermark is designed to be undetectable, it remains undetected until the server on the other side detects it and reveals the source of the data, which was supposed to be anonymous.

The most common use case for the watermarking technique is for data authorization. While this algorithm can embed a robust watermark, it can still not be used as definitive proof of who produced the watermark. For example, suppose a watermarked database is made public against the author's permission. In that case, they can use the watermark detection technique to identify how and from whom the database was leaked. However, this cannot be used in court as evidence, as the algorithm was not designed and tested for these scenarios.

8 Conclusions

We introduce a new graphical database watermarking algorithm based on inserting watermarked pseudo nodes. This novel approach, first introduced by [10], promises to create a robust, non-detectable, and secure watermark, resilient to modification, deletion and insertion attacks. Several flaws, such as the genetic algorithm and the graphical properties of the watermark, were identified after analyzing the algorithm in [10]. A new algorithm was designed and implemented to tackle these flaws and then tested against modification and deletion attacks. The test results showed that the algorithm under-performed in contrast to the algorithm introduced by [10]. This means that further research needs to be done for a robust and secure algorithm to be introduced for graph database watermarking.

9 Further work

Remove the need for manual analysis

Unfortunately, the lack of schemes in NoSQL databases introduces the need for manual analysis of the database before it can be watermarked. This is an intensive process and requires deep knowledge of how the technique works and knowledge of python to implement the algorithm for the database. Automatic scheme analysis and automatic watermark embedding can greatly improve the usability of the algorithm, removing the chance of configuration mistakes.

Watermarking edges

As the graph database can also hold data inside the edges, edges can be watermarked to increase robustness against deletion and modification attacks.

Random distribution of IDs

As this algorithm was implemented for the Neo4j database, it is also limited to the constraints enforced by the database. One such constraint does not permit a document's primary key to be modified. This can be circumnavigated by cut-pasting documents from the database back to the database, essentially giving a document a new primary key. However, this technique is not recommended, as it could potentially decrease the detectability of the watermark.

Algorithm is not limited to the database type

Although this algorithm was implemented for a graphical database, many other databases can be reduced to a graph representation, as demonstrated in [10]. The technique in this paper can then be used to watermark the graph, greatly increasing the use cases for it.

Improve methods for embedding

Although this method was proposed with a simple watermark embedding algorithm, improving the way it is embedded can greatly increase its imperceptibility. Due to the design of the algorithm, the watermark embedding and verification algorithms can be treated as black boxes, and the way the rest of the algorithm works is not influenced.

References

[1] S. Kumar, B. K. Singh, and M. Yadav, "A recent survey on multimedia and database watermarking," *Multim. Tools Appl.*, vol. 79, no. 27-28, pp. 20149–20197, 2020.

[2] E. N. M. Ibrahim, N. L. M. Noor, and S. Mehad, "Trust or distrust in the web-mediated information environment (W-MIE): A perspective of online muslim users," *J. Enterp. Inf. Manag.*, vol. 22, no. 5, pp. 523–547, 2009.

[3] M. Winslett, T. Yu, K. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu, "Negotiating trust in the web," *IEEE Internet Computing*, vol. 6, no. 6, pp. 30–37, 2002.

[4] S. Katzenbeisser and F. A. P. Petitcolas, *Information hiding techniques for steganography and digital watermarking*. Artech House computer security series, Boston: Artech House, 2000.

[5] C. Dr. Shannon Block, "Why amazon, google, netflix and facebook switched to nosql?," Mar 2019.

[6] M. Bach and A. Werner, "Standardization of nosql database languages," in *Beyond Databases, Architectures, and Structures - 10th International Conference, BDAS 2014, Ustron, Poland, May 27-30, 2014. Proceedings* (S. Kozielski, D. Mrozek, P. Kasprowski, B. Malysiak-Mrozek, and D. Kostrzewa, eds.), vol. 424 of *Communications in Computer and Information Science*, pp. 50–60, Springer, 2014.

[7] J. J. Miller, "Graph database applications and concepts with neo4j," in *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*, vol. 2324, 2013.

[8] "The professional client, ide and gui for mongodb," Jan 2023.

[9] T. M. Thanh, N. H. Thuy, and N. Huynh, "Key-value based data hiding method for nosql database," in *10th International Conference on Knowledge and Systems Engineering, KSE 2018, Ho Chi Minh City, Vietnam, November 1-3, 2018* (N. T. Thuy, S. Tojo, T. Hanh, M. L. Nguyen, T. M. Phuong, and V. N. Q. Bao, eds.), pp. 193–197, IEEE, 2018.

[10] X. Zhuang, X. Luo, and L. He, "Document database watermark algorithm based on connected graph," in *2022 International Conference on Computing, Communication, Perception and Quantum Technology (CCPQT)*, pp. 212–218, 2022.

[11] MongoDB, "Documents," 2022.

[12] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multim. Tools Appl.*, vol. 80, no. 5, pp. 8091–8126, 2021.

[13] A. Bookstein, V. A. Kulyukin, and T. Raita, "Generalized hamming distance," *Inf. Retr.*, vol. 5, no. 4, pp. 353–375, 2002.

[14] K. Emam, L. Mosquera, R. Hoptroff, and a. O. M. C. Safari, "Practical synthetic data generation," 2020.

[15] Neo4j, "Neo4j Bolt driver for Java."

[16] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[17] R. Bardou, R. Focardi, Y. Kawamoto, L. Simionato, G. Steel, and J. Tsay, "Efficient padding oracle attacks on cryptographic hardware," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings* (R. Safavi-Naini and R. Canetti, eds.), vol. 7417 of *Lecture Notes in Computer Science*, pp. 608–625, Springer, 2012.

[18] T. Hristov, "Graph Database Watermarker(BEP)."

[19] W. Lyon, "UK Companies Data."

[20] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, no. 239, p. 2, 2014.

A Search parameters

Engines	Google Scholar, Scopus
Keywords	Digital watermarking, watermark, algorithm, database, NoSQL, document database, graph
Example query	Digital watermarking AND noSQL AND graph

Table 1: Search parameters

B Database specifications

Label	Amount
Person	30207
Company	35000
Recipient	65
Property	4728

Table 2: Amount of nodes per label

Type	Amount
HAS_CONTROL	60414
DONATED	130
OWNS	9456

Table 3: Amount of edges per type

C Result graphs

C.1 Impact of parameters on watermark 3D

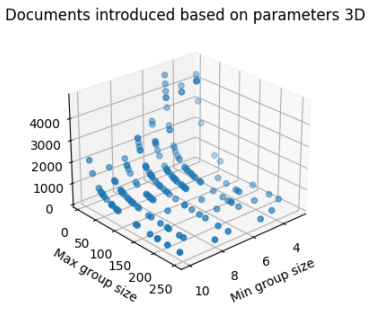

Figure 10: Impact of parameters on amount of watermarked documents

D Parameters of the algorithm

parameter	**description**
ids	The ids of the documents, which need to be watermarked
labels	The labels of the documents, which need to be watermarked
$\text{min}_{g}roup_{s}ize$	The minimum size of each group
$\text{max}_{g}roup_{s}ize$	The maximum size of each group
$\text{watermarked}_{d}ocument_{t}ype$	The type of the document, which contains the watermark
$\text{watermark}_{c}over_{f}ield$	The name of the field, which contains the watermark
$\text{watermarked}_{d}ocument_{f}ields$	The fields, which are included inside the watermarked document
$\text{watermarked}_{d}ocument_{o}ptional_{f}ields$	The fielDs, which are optionally included inside the watermarked document
$\text{watermark}_{k}ey$	The private key for the watermark
$\text{group}_{f}ind_{m}ax_{t}ries$	The amount of tries it will take for the algorithm to give up on partitioning groups
$\text{watermark}_{i}dentity$	An optional identity for the watermark, this can be the name of the person the information was leased or any other identifiable string
$\text{watermark}_{e}dge_{d}irection_{r}andomized$	If the edge direction should be randomized. If true, the algorithm will treat the graph as undirected and not take the direction of the edges into account when creating the watermark.
$\text{watermark}_{v}isibility$	Optional. If selected, the letter W will be added on the back of the type of the watermark document and edges, indicating that it is watermarked.

Table 4: Watermark algorithm parameters

E Specifications of the host computer

Part	Description
CPU	AMD Ryzen 9 3900XT 12-Core 3.80 GHz Processor
GPU	NVIDIA GeForce RTX 3070, driver version 527.37
RAM	16,0 GB
OS	64-bit Windows 11 Pro, build 22621.1105

Table 5: Host specifications

F Algorithms

F.1 Divide into groups

Algorithm 2 Divide into groups

function GENERATEGROUPSIZES($target\_sum, lower\_range, upper\_range, max\_tries$) ▷ Generate a set of random numbers, all summing up to a target sum
  if $max\_tries <= 0$ **then** ▷ Base case: check if the maximum tries count was exceeded
    $Exit$("No groups were found satisfying the parameters")
  end if
  $returnList \leftarrow []$

  while $target\_sum - Sum(returnList) > upper\_range$ **do** ▷ Fill the list with random numbers, until you hit the upper bound of the range
    returnList.append(randint(RandomNumber())
  end while
  $remainder \leftarrow target\_sum - Sum(returnList)$
  if *remainder* $< lower\_range$ **then** ▷ If the number is not in the range, run the algorithm again
    $returnGenerateGroupSizes(target\_sum, upper\_range, lower\_range, max\_tries - 1)$
  else
    returnList.append($remainder$)
    return $returnList$
  end if
end function

F.2 Create pseudo document

Algorithm 3 Create pseudo document

function GETDISTINCTVALUES($document\_type, field$) ▷ Get all distinct values for a field
  $allDocuments \leftarrow GetAllDocuments(document\_type)$
  $allValues \leftarrow []$
  for $document \in allDocuments$ **do** ▷ Loop over all documents
    if $document.get(field) \notin allValues$ **then** ▷ Check if the value is already inside the list
      $allValues.append(document.get(field))$ ▷ Append the value to the list
    end if
  end for
  return$allValues$
end function
procedure CREATE PSEUDO DOCUMENT($document\_type, fields$)
  $newDocument \leftarrow BlankDocument()$
  for $field \in fields$ **do**
    $allPosibilities \leftarrow GetDistinctValues(document\_type, typeof field)$ ▷ Get all possible values for this field
    $newField \leftarrow RandomChoice(allPosibilities)$ ▷ Choose a value for the field
    $ChangeDocument(newDocument, field, newField)$ ▷ Save the new field
  end for
end procedure

F.3 Connect pseudo node to the rest of the group

Algorithm 4 Connect pseudo node to the rest of the group

procedure CONNECT PSEUDO NODE($pseudo\_node, group, edge\_type\_lookup$)
  for $node \in group$ **do**
    $edgeType \leftarrow edge\_type\_lookup[typeofpseudo\_node][typeofnode]$ ▷ Determine the edge type from the lookup matrix
    $ConnectDocument(pseudo\_node, node, edgeType)$ ▷ Connect the two documents with the edge type
  end for
end procedure

F.4 Deletion Attack

Algorithm 5 Deletion Attack

function GETRANDOMDOCUMENTS($size$) ▷ Generate a random set of document IDs with a certain size
  allIDs ← *getAllIDs*()
  randIDs ← *generateRandomSubset*(*allIDs*, *size*)
  return$randIDs$
end function

function DELETION SCRIPT($iterations, step$)
  for $iterations$ **do**
    $documentToModify \leftarrow$ *GetRandomDocuments*($step$) ▷ Chose random documents to delete
    $DeleteDocument$(documentToModify) ▷ Save changes
  end for
end function

procedure DELETION ATTACK($iterations, step, database$)
  do
    $Deletionscript$(iterations, step) ▷ Call the deletion script
    $detected \leftarrow VerifyWatermark$(database) ▷ Check for the watermark
  while $detected = true$
end procedure

F.5 Modification attack

Algorithm 6 Modification attack

function GETRANDOMDOCUMENTS($size$) ▷ Generate a random set of document IDs with a certain size
    allIDs ← *getAllIDs*()
    randIDs ← *generateRandomSubset*(*allIDs*, *size*)
    return $randIDs$
end function

function GENERATERANDOMDATA($data\_type$) ▷ Generate random data from the same type as the specified
    new_data ← *GenerateRandomData*($data\_type$)
    return new_data
end function

function MODIFICATION SCRIPT($iterations$)
    for iterations **do**
        $documentToModify$ ← *GetRandomDocuments*(1)[0] ▷ Chose a random document to modify
        $fieldToModify$ ← *RandomChoice*($documentToModify.fields$) ▷ Chose a random field to modify
        $newData$ ← *GenerateRandomData*($typeof fieldToModify$) ▷ Generate new data to overwrite the previous one
        ChangeDocument($documentToModify, fieldToModify, newData$) ▷ Save changes
    end for
end function
procedure MODIFICATION ATTACK($iterations, step, database$)
    do
        $Modificationscript$(iterations, step) ▷ Call the modification script
        $detected$ ← $VerifyWatermark$(database) ▷ Check for the watermark
    while $detected = true$
end procedure