Graph Database Watermarking with Pseudo Nodes

Group 2, Github repository: gdb_watermarking

Saurav Dosi, Abhinav Santhosh, Jay Pandya and Kavimayil P K.

***Abstract*— This work implements Graph Database Watermarking technique using pseudo nodes to verify ownership of the suspected database. Pseudo node attributes are used to generate a hash value. These pseudo nodes are inserted into the original database. The validation algorithm verifies existence of the pseudo nodes and the watermark inside them to declare the ownership. In this work, we successfully reproduce the results from the original work by using synthesized data to claim the ownership. We further make the implementation more accessible for human intervention required for the schema analysis. Please find the implementation in the GitHub repository attached above.**

I. INTRODUCTION

The exponential growth of data-driven technologies and the rise of data marketplaces have elevated the importance of protecting database ownership against unauthorized distribution and misuse. Graph databases, designed for managing complex, interconnected datasets, have become increasingly vital in diverse applications, from social networks to recommendation systems. Unlike traditional relational databases, graph databases are schema-less and rely heavily on intricate relationships between nodes and edges, presenting unique challenges for security measures such as watermarking. Watermarking is a well-established technique for embedding ownership information into data while maintaining usability. While extensive research has been conducted on watermarking for relational databases and media, its application to non-relational databases, particularly graph databases, remains underexplored. Existing methods, such as Zhuang et al.'s approach for MongoDB[1], provide robustness through techniques like pseudo-document embedding but are constrained by their dependence on computationally intensive genetic algorithms. This reliance limits scalability and makes these techniques impractical for real-time or large-scale graph databases. Thus, there is a significant need for a watermarking solution that is not only efficient and scalable but also preserves the database's structural and functional integrity while offering strong resilience against attacks like deletion, tampering, and guessing. The problem of embedding ownership information in graph structures without compromising database integrity or performance remains underexplored.

II. OUR CONTRIBUTIONS

We introduce an improved watermarking scheme for graph databases that builds upon the method proposed by Zhuang et al.[1]. By utilizing randomization techniques to generate pseudo-nodes for watermark embedding, we eliminate the reliance on computationally intensive genetic algorithms, thereby reducing both computational complexity and enhancing scalability. We implemented our method within a Neo4j environment and conducted performance evaluations using a real-world dataset, optimizing the watermarking process for various graph structures. The performance analysis includes detailed runtime benchmarks, where watermark generation for a graph with 5000 nodes was completed in 1.3 seconds. Moreover, we rigorously tested the security and robustness of the scheme, demonstrating its resilience against deletion, and insertion attacks. Our results show that the method provides a secure and efficient solution for ownership verification, with no disruption to the original graph structure. This work offers a significant advancement in watermarking for graph databases by providing a technically efficient, secure, and scalable solution for protecting data ownership.

III. RELATED WORK

This section reviews key contributions in database security, particularly focusing on watermarking and access control techniques for relational, NoSQL, and graph databases.

A. Document Database Watermarking

Zhuang et al. [1] introduced a watermarking technique for NoSQL document databases that leverages genetic algorithms to generate pseudo-documents, which are synthetic entries embedded with watermarks. These pseudo-documents are structured into a k-connected graph and are designed to closely resemble legitimate entries, making them difficult to detect. The genetic algorithm optimizes the balance between similarity to real data and uniqueness, ensuring effective embedding. Although this method demonstrates strong resistance to tampering, its reliance on computationally demanding genetic algorithms significantly limits scalability and suitability for real-time or high-frequency applications. Building on this foundation, our approach replaces genetic algorithms with a randomized generation of pseudo-nodes, preserving robustness while substantially enhancing computational efficiency and scalability, making it more viable for real-time and high-throughput environments.

B. Relational Database Watermarking

Kamran and Farooq [2] provide a comprehensive survey of watermarking techniques tailored for relational databases, emphasizing robust embedding strategies such as bit-resetting, data statistics modification, and constrained content alteration. These methods effectively secure structured datasets by embedding watermarks into tuples or

attributes, ensuring resilience against insertion, deletion, and modification attacks. However, their reliance on fixed schemas and structured data inherently limits their applicability to schema-less graph databases, where data is represented as interconnected nodes and relationships. Our proposed approach extends these principles to graph databases by leveraging pseudo-nodes—synthetically generated nodes embedded with high-entropy secrets—to create a robust watermarking mechanism. This method preserves the functional integrity of the database while ensuring ownership protection and optimizing computational performance, addressing the challenges posed by the flexible and interconnected nature of graph data.

C. Dynamic Watermarking for NoSQL Databases

Khanduja et al. [3] propose a dynamic watermarking technique for NoSQL databases that injects watermarks at the tuple level, enabling efficient real-time tamper detection by recalculating the watermark independently for each tuple. While this approach is effective for detecting small-scale modifications, it is inherently vulnerable to large-scale deletions due to its localized embedding strategy. In contrast, our proposed method embeds pseudo-nodes across the graph structure, leveraging the interconnected nature of graph databases to distribute the watermark. This distributed embedding enhances resilience against large-scale deletions and guessing attacks, ensuring robust ownership protection and structural integrity in highly interconnected data environments.

D. Access Control in Graph Databases

Morgado et al. [4] present an access control framework for graph databases that integrates user permissions directly into the graph structure, utilizing metadata-driven relationships to govern Data Definition Language (DDL) and Data Manipulation Language (DML) operations. While this model effectively enforces fine-grained access control in graph-oriented systems, it does not address ownership verification or ensure resilience against tampering and unauthorized redistribution. Our proposed method complements such access control mechanisms by embedding ownership indicators within the graph structure through pseudo-nodes. These pseudo-nodes provide a robust means of watermarking, ensuring data provenance and safeguarding against ownership disputes and malicious modifications.

E. Graph Database Security Models

Mohamed et al. [5] conduct a comprehensive survey of access control mechanisms for graph databases, with a particular focus on policy-driven approaches such as Attribute-based Relationship-Based Access Control (AReBAC) and eXtensible Access Control Markup Language for Graphs (XACML 4G). These methods enable fine-grained security through graph pattern matching and conditional policy enforcement. However, while effective for regulating access and maintaining data confidentiality, these models do not provide mechanisms for ownership protection or traceability. Our proposed watermarking technique addresses this limitation by embedding robust, non-intrusive markers—pseudo-nodes—directly into the graph structure. This approach ensures data provenance, safeguards against unauthorized replication, and provides verifiable ownership without disrupting the database's functionality or performance.

F. Blockchain-Based Solutions

Blockchain integration in graph databases ensures data integrity by leveraging immutable ledgers that record all changes transparently, making unauthorized modifications detectable. This approach excels at tamper detection and providing a clear audit trail, which is particularly useful in systems where data provenance and integrity are critical. However, blockchain-based solutions are inherently resource-intensive, requiring substantial computational power and storage, especially as the database scales. Additionally, these methods primarily focus on ensuring data consistency and traceability, lacking the capability to embed ownership indicators directly within the graph structure. In contrast, our watermarking method provides a lightweight and scalable solution specifically designed for ownership verification in graph databases. By embedding pseudo-nodes as non-intrusive markers within the graph, our technique ensures robust ownership validation while maintaining the structural and operational integrity of the database. Unlike blockchain, which can introduce latency and complexity, our method operates with minimal performance overhead, offering an efficient alternative that balances security with practical usability in high-frequency or real-time applications. This makes our approach particularly suitable for environments where computational efficiency and scalability are as critical as ownership verification.

IV. Method

A. Background

SQL and NoSQL databases differ primarily in terminology and structure. In an SQL database, information is stored in tables composed of rows (or records) and columns, adhering to a well-defined schema. Conversely, NoSQL databases use collections that store data in key/value pairs, offering flexibility in data representation. The relationship between documents varies across NoSQL systems: for instance, document-based databases may use embedding or referencing, while graph databases represent relationships using edges between nodes. This flexibility in NoSQL databases poses a challenge for watermarking, as there is no guarantee that a document follows a fixed structure. In graph databases, nodes (documents) are linked by directed edges, with the possibility of storing additional data about the relationships.

B. Watermarking Overview

The watermarking technique described in this paper is based on a model where a high-entropy watermark is embedded into the graph database to prove ownership. The process involves two main algorithms:

1) **Watermark Embedding:** Embeds a watermark within the graph database while preserving its original functionality.
2) **Watermark Extraction:** Extracts the watermark from a potentially tampered database to verify the ownership.

C. Watermark Embedding Process

Algorithm 1 Watermark Embedding Algorithm

Require: N (total number of nodes), $g_{\min}$ (minimum group size), $g_{\max}$ (maximum group size)
Require: R (required fields), O (optional fields)
Require: K (private key)
Ensure: Watermarked graph with pseudo-nodes and watermark secret

Generate Group Partitions:
Compute the number of groups G such that:

$$G \leq \left\lceil \frac{N}{g_{\min}} \right\rceil \text{ and } G \geq \left\lfloor \frac{N}{g_{\max}} \right\rfloor$$

Assign nodes to groups $G_1, G_2, \ldots, G_G$ ensuring group sizes lie within $[g_{\min}, g_{\max}]$.
Create Actual Groups:
for each group $G_i \in \{G_1, G_2, \ldots, G_G\}$ **do**
    Select nodes from N such that $|G_i| \in [g_{\min}, g_{\max}]$
end for
Generate Pseudo Nodes:
for each group G_i **do**
    Use required fields R and optional fields O to create a pseudo node v_{p_i}:

$$v_{p_i} = \{r_1, r_2, \ldots, r_m, o_1, o_2, \ldots, o_n\} \quad r_i \in R,\ o_j \in O$$

end for
Watermark Pseudo Nodes:
for each pseudo node v_{p_i} **do**
    Generate a unique watermark ID w_i for v_{p_i}.
    Compute the hash h_i:

$$h_i = H(v_{p_i}, w_i, K)$$

    Store w_i and h_i for validation.
end for
Insert Pseudo Nodes:
Add pseudo nodes $\{v_{p_1}, v_{p_2}, \ldots, v_{p_G}\}$ back to the original graph.
Generate Watermark Cover IDs:
for each node $v \in N$ (excluding pseudo nodes) **do**
    Assign a unique watermark cover ID c_v.
end for
Watermark Secret:
Store and share the watermark secret S:

$$S = \{K,\ \{w_i | \forall v_{p_i}\}\}$$

=0

The watermark embedding algorithm operates with:

1) **Original Graph Database (D):** The unaltered graph database.
2) **Private Key (K):** A high-entropy secret key used to generate the watermark.

The steps involved in embedding are described in detail below.

1) Group Generation: The database is partitioned into groups of nodes based on predefined sizes. This is done using a `GroupGen` algorithm that randomly divides the nodes into subgroups. Example: Given a sum of sizes (e.g., 15), the algorithm splits it into smaller groups, such as [3, 4, 3, 5].

2) Pseudo-Document Generation (`PseudoGen`): Pseudo-documents are synthetically created by extracting fields from real documents in the database. These pseudo-documents are not part of the core data but are used solely for embedding the watermark. Each pseudo-document mimics real database entries but is designed to hold the watermark without altering the actual data.

3) Marking (`Mark`): Each pseudo-document is marked with a hash derived from the private key (K) and specific fields of the pseudo-document. This ensures that the watermark is unique and linked to the key. For numerical fields, the watermark is applied via modular arithmetic:

$$\text{new\_field} = H(\text{fields} \mid\mid K) \mod \text{field\_max}$$

Here, H represents a hash function, and field_max is the maximum possible value for that field.

4) Linking (`Link`): The marked pseudo-documents are then linked back to the original documents. For graph databases like Neo4j, this is done by adding directed edges between the original nodes and the pseudo-documents. This ensures that the watermark can be traced to the appropriate sections of the graph, establishing ownership of the database.

5) Return Watermarked Database (D_x): The result is a watermarked graph database (D_x), which includes both the original nodes and the added pseudo-documents. The pseudo-documents, identified by their unique identifiers, carry the watermark, which can later be extracted and verified.

D. Watermark Extraction Process

The watermark extraction process involves verifying the watermark in a suspected database (D') using the private key (K) and a list of pseudo-document identifiers. The steps involved in the extraction process are described below.

1) Generating Fake Data: Process required generating fake data using the Faker library to simulate an insertion attack, creating records that resemble the original dataset. Original, watermarked data was then inserted into this fake data to test watermark detection under realistic conditions. The goal was to evaluate whether the watermarking technique can still validate the watermark when mixed with synthetic data.

2) Retrieve Pseudo-Documents: The algorithm retrieves the pseudo-documents from the suspected watermarked database (D') using their unique identifiers (ID_p).

3) Check Watermark Validity: The algorithm checks if the watermark in the pseudo-documents matches the original watermark generated with the private key (K). This involves comparing the hash values of the fields in the retrieved pseudo-documents with the expected hash, calculated using the private key (K):

$$H(\text{fields} \mid\mid K) \mod \text{field\_max}$$

If the hashes match, the watermark is verified; otherwise, it is rejected.

4) Return Verification Result: If the watermark is valid, the algorithm returns `accept`; otherwise, it returns `reject`, indicating whether the database is genuinely watermarked.

Algorithm 2 Watermark Validation Algorithm

Require: D (suspected data), K (private key), $\{v_{p_i}\}$ (watermarked pseudo-node IDs), $\{c_v\}$ (watermark cover IDs)
Ensure: Verification of ownership of the data
Check for Watermark Cover in Nodes:
for each node $v \in D$ **do**
  if v contains a watermark cover c_v **then**
Proceed to Pseudo-Node Validation
    if $c_v \in \{v_{p_i}\}$ **then**
Compute hashed secret h_v using the hash function:

$$h_v = H(v, w_v, K)$$

Retrieve the expected hashed secret $h_{expected}$.
      if $h_v = h_{expected}$ **then**
Watermark Found: Ownership of the data is verified.
Exit algorithm.
      else
Watermark Invalid: Continue to next node.
      end if
    else
Watermark Cover Invalid: Continue to next node.
    end if
  else
No Watermark Cover Found: Continue to next node.
  end if
end for
Result: If no watermark is found, ownership cannot be verified.
=0

E. Master Algorithm

We use the following algorithm to execute the required processes in the correct order, by using only one script. This script prints the database summary, keys and partial keys along with numerical attributes in the database. This makes it easier for the schema analysis to be performed by the human.

Algorithm 3 Master Algorithm

Input: $G(V, E)$, grouping parameters $\{g_{\min}, g_{\max}, n_{\text{total}}\}$, fields $\{F_r, F_o\}$, watermark cover $\mathcal{C}$
Output: Watermarked graph $G'(V', E')$, validation result
Step 1: Connect to the Graph Database
Establish a connection using credentials $\mathcal{C}_{\text{db}}$:

$$\text{connect}(G, \mathcal{C}_{\text{db}})$$

Step 2: Analyze Database Schema
Print schema summary:

$$\text{schema\_summary}(G)$$

Step 3: Select Nodes for Watermarking
Define $V_w \subset V$ based on node type:

$$V_w = \text{select\_nodes}(G, \text{type})$$

Step 4: Select Grouping Parameters
Choose $\{g_{\min}, g_{\max}, n_{\text{total}}\}$ to define watermark groups:

$$\text{group\_params} = (g_{\min}, g_{\max}, n_{\text{total}})$$

Step 5: Define Fields and Watermark Cover
Specify fields and coverage:

$$\{F_r, F_o\}, \mathcal{C}$$

Step 6: Generate Private Key
Generate key k:

$$k = \text{generate\_key}()$$

Step 7: Embed Watermark
Generate pseudo-nodes P and embed into G:

$$G' = \mathcal{W}(G, P, k, \mathcal{C})$$

Step 8: Share Watermark Secret
Provide k and P to the user for verification:

$$\text{share\_secret}(k, P)$$

Step 9: Synthesize Data for Verification
Generate synthesized dataset D_{syn}:

$$D_{\text{syn}} = \text{synthesize\_data}(G')$$

Step 10: Validate Watermark
Validate watermark in D_{syn}:

$$\text{result} = \mathcal{V}(D_{\text{syn}}, k, P)$$

if result = success **then**
  Print: "Watermark validation successful."
else
  Print: "Watermark validation failed."
end if
=0

1) Notation:

- $G(V, E)$: Original graph database, where V represents

the nodes and E the edges.

- $V_w \subset V$: Subset of nodes selected for watermarking.
- P: Set of pseudo-nodes used for watermarking.
- K: Private key for generating and embedding the watermark.
- F_r, F_o: Required and optional fields for pseudo-nodes.
- $\mathcal{W}$: Watermark embedding function.
- $\mathcal{V}$: Watermark validation function.

V. Experimental Setup

A. Dataset

The experiments were conducted using the UK Companies dataset. This dataset includes information about 35,000 public companies in the United Kingdom, including their properties, owners, and political donations. It contains four types of nodes with the following label distributions:

- **Person:** 23,606 entries
- **Company:** 26,226 entries
- **Recipient:** 9 entries
- **Property:** 4,728 entries

Every `Person`, `Recipient`, or `Property` node is linked to at least one `Company` node, enabling watermarking coverage across the database. Pseudo-nodes used for watermarking are linked to these existing nodes, ensuring robustness and full integration with the graph structure.

B. Database Implementation

The database was implemented and managed using **Neo4j Desktop**, an industry-standard graph database management system. Neo4j's flexibility and graph-native storage capabilities made it well-suited for managing the interconnected data required for this experiment. The **APOC** plugin suite was installed to enable advanced functionality, such as parsing and manipulating dates during watermark embedding. APOC (Awesome Procedures on Cypher) enhances Neo4j's querying and scripting capabilities.

C. Environment and Tools

All algorithms were implemented in **Python 3.12** within a Jupyter Notebook environment. The following libraries were used:

- `py2neo` for interacting with the Neo4j database.
- `pandas` for data manipulation and analysis.
- `hashlib` for generating cryptographic hashes.

D. Source Code

The source code and execution steps are provided in the GitHub repository: https://github.com/sauravdosi/gdb_watermarking

VI. Evaluation

While evaluating, we stick to a sample size of 1000 records across all the evaluations due to computational restrictions of the machine on which the database was loaded. However, we observe the exact same trends as depicted in the original paper. This proves that our implementation was successful.

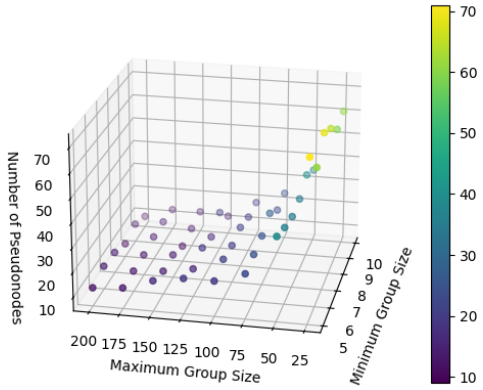

Fig. 1. Effect of Group Parameters on Pseudonode Count

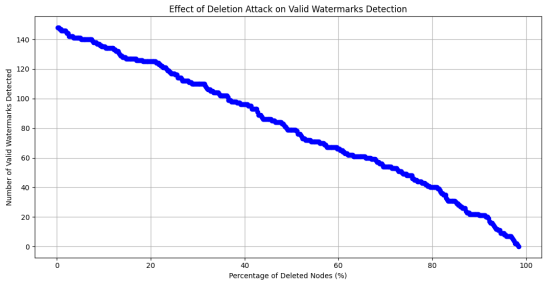

Fig. 2. Effect of Deletion Attack on Pseudonode Count

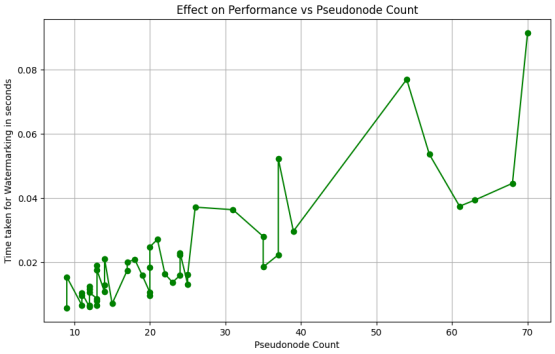

Fig. 3. Time Taken to Watermark vs Pseudonode Count

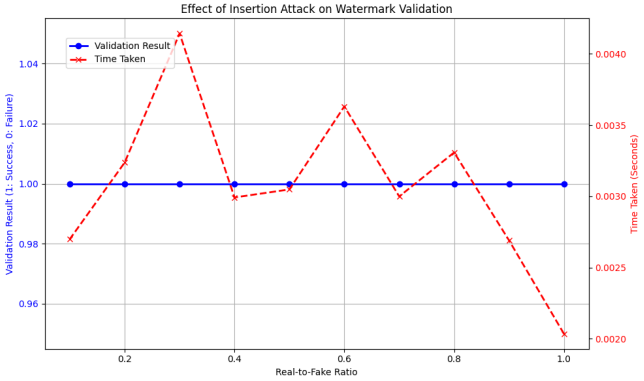

Fig. 4. Insertion Attack Analysis

A. Performance

The algorithm's performance was analyzed by measuring the time required to watermark all groups, plotted against the number of pseudo-nodes processed. The x-axis represents the pseudo-node count, while the y-axis corresponds to the time taken in seconds. As depicted in Figure 3, the execution time increases linearly with the number of pseudo-nodes, indicating that the algorithm exhibits O(n) time complexity. This behavior demonstrates the scalability of the approach, as it can efficiently handle larger datasets without significant performance degradation. Empirical measurements show that the algorithm processes an average of 1500 pseudo-nodes per second under sequential execution, without leveraging parallel or multithreaded optimizations.

B. Usability

Similar to previous graph database watermarking techniques, our method introduces distortion to the original database by inserting pseudo-documents. To assess the level of distortion after embedding, we evaluated the number of pseudo-documents inserted for various minimum and maximum group sizes. As shown in Figure 1, increasing the difference between the maximum and minimum group sizes results in a decrease in the number of pseudo-documents introduced, thereby reducing the distortion in the original database. Conversely, smaller group sizes, which lead to a smaller difference between the minimum and maximum values, result in a higher number of pseudo-documents being generated. It is important to note that while a higher number of pseudo-documents increases the robustness of the watermark, it also contributes to greater distortion in the database.

C. Deletion Attack

In this attack type, the attacker randomly deletes records from the original dataset. The objective is to assess whether the watermarking mechanism can still successfully detect the watermark despite the removal of some records. Based on the analysis with PyPlot in Figure 2, the results demonstrate that even a single watermarked pseudo document is sufficient to validate the watermarking of the original dataset. Therefore, a deletion attack would not be effective unless the attacker manages to completely remove all watermarked nodes. Achieving this would require the deletion of nearly the entire dataset, making such an attack highly impractical and make the system with this approach more robust. This demonstrates a strong aspect of our algorithm's performance, as it reliably detects 20% of the watermarked pseudo-nodes or documents even after more than 80% of the nodes are deleted.

D. Insertion Attack

In this attack type, the attacker attempts to insert fake data into the original dataset, with the aim of breaking or invalidating the watermark embedded in the original data. The attacker does this by generating fake records with a real-to-fake ratio ranging from 1 (all original data) to 0.1 (mostly fake data). The objective is to see if the watermarking mechanism can still detect the watermark even when the dataset contains a significant proportion of fake records. Based on the analysis in Fig. 4, the results demonstrate that the watermarking system remains highly effective even when the real-to-fake ratio is reduced to as low as 0.1 (10% original data). The watermarking algorithm continues to validate the watermark successfully in a very short time frame, often within 0.004 seconds for 1000 records of data, even when the fake data ratio is high. This indicates that the watermark detection is robust to insertion attacks, and the system is resilient against the introduction of fake data.

VII. Conclusion

In this work, we have developed and evaluated a novel watermarking technique for graph databases, designed to overcome the limitations of prior approaches, particularly in terms of optimization complexity. Our technique employs a randomized embedding approach that mitigates the risk of detection and removal of pseudo-documents inserted into the database, ensuring greater security and robustness compared to traditional methods. We have demonstrated that our approach strikes a balance between performance and usability. By allowing users to make schema decisions without needing direct access to the data, our method enhances flexibility and control over the watermarking process. This user-centric design ensures that the technique can be adapted to a wide range of database structures, making it practical for real-world applications. Through rigorous implementation and evaluation on a real-world database, we have shown that our technique is robust against a variety of attack scenarios, while maintaining efficiency in terms of processing time and database distortion. The results also validate that the approach is secure, with minimal vulnerability to common attacks, such as node deletion or manipulation. While the computational power of the system on which the database was installed limited the sample size used for testing, the results were consistent with our theoretical expectations, confirming the scalability of the method. This paves the way for further validation on larger datasets and in more complex environments. Future work will involve comparing our

technique with other existing graph database watermarking solutions in terms of performance, usability, and robustness. Overall, this work represents a significant step forward in the field of graph database watermarking, offering a secure, efficient, and user-friendly solution to protect data integrity in graph-based systems.

A. Novel Improvements

- We provide additional control over the algorithm by letting the user make schema decisions without scrutinizing the data a lot.
- We also provide easier access to testing and reproducing the results by only changing a few parameters in the code.
- We watermark all node types, instead of just Company as done by the authors, to provide security to all the node types without altering the graph relations. This is computationally more efficient.
- We consider all the numerical attributes in the pseudo nodes as opposed to only a few selected as implemented by the authors, to produce a more robust hash value.

VIII. Contributions and Task Delegation

Kavimayil made significant contributions to the initial phase of the project by conducting a thorough literature review. This task involved analyzing existing work in the field of graph database watermarking, understanding the strengths and weaknesses of prior methods, and identifying key areas where improvements could be made. Kavimayil's review helped shape the foundation for our algorithm, ensuring that our approach was informed by the latest research. The review also provided valuable insights into the security and performance aspects of watermarking techniques, which guided the development of our novel solution.

Abhinav was responsible for the Exploratory Data Analysis (EDA) and system setup. This included analyzing the dataset to understand its structure, preprocessing it as needed, and setting up the environment for the implementation of the watermarking algorithm. Abhinav also took charge of configuring the graph database, ensuring that the system was ready for the next phases of the project. In addition, Abhinav worked closely with Jay on the testing and evaluation phase. Together, they reproduced the results from the original paper and rigorously tested the watermarking technique. Abhinav's contribution was key in evaluating the algorithm's performance, ensuring its effectiveness, and confirming its robustness against various attack scenarios.

Jay played a key role in implementing the watermark extraction algorithm. His work focused on creating an efficient and robust method to extract the watermarks embedded within the graph database. Jay's implementation ensured that the watermark could withstand various attack scenarios, such as node deletions or modifications. Additionally, Jay worked closely with Abhinav on the testing and evaluation phase. Together, they rigorously tested the algorithm to evaluate its performance, ensuring that it maintained high security while avoiding excessive distortion in the graph database. Jay's contributions were essential in proving the technique's resilience and effectiveness in real-world conditions.

Saurav led the team in developing the code, integrating and generating evaluation results. Additionally, he developed the watermarking embedding algorithm, which was a core part of the project. His work involved designing and implementing a randomized approach to insert pseudo-documents into the graph database without compromising security or performance. This method was crucial in overcoming the limitations of previous watermarking techniques that relied on overly optimized methods. Saurav also suggested potential improvements to the algorithm, offering novel enhancements that increased its robustness and usability, including features that allowed users to make schema-level decisions without needing to access the underlying data.

All team members collaborated on the final report and documentation. Each member contributed to detailing the project methodology, algorithm implementation, and results, ensuring that all aspects of the project were well-documented. The final report also included a thorough explanation of the source code, making it accessible for future research or applications. This collaborative effort ensured that the project's outcomes were comprehensively documented, with the results effectively communicated for dissemination to the academic and professional community

References

[1] Xiaodan Zhuang, Xi Luo, and Letian He. 2022. Document Database Watermark Algorithm Based on Connected Graph. In *2022 International Conference on Computing, Communication, Perception and Quantum Technology (CCPQT)*. 212–218. https://doi.org/10.1109/CCPQT56151.2022.00044

[2] Muhammad Kamran and Muddassar Farooq. 2018. A comprehensive survey of watermarking relational databases research. *arXiv preprint arXiv:1801.08271* (2018) https://arxiv.org/pdf/1801.08271.

[3] Vidhi Khanduja. 2016. Dynamic watermark injection in NoSQL databases. *Journal of Computer Science Applications and Information Technology. Symbiosis* (2016), 1–5 https://symbiosisonlinepublishing.com/computer-science-technology/computerscience-information-technology08.php.

[4] Claudia Morgado, Gisele Busichia Baioco, Tania Basso, and Regina Moraes. 2018. A Security Model for Access Control in Graph-Oriented Database. In *IEEE International Conference on Software Quality, Reliability and Security (QRS)*. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8424965

[5] Aya Mohamed, Dagmar Auer, Daniel Hofer, and Josef Kueng. 2024. Comparison of Access Control Approaches for Graph-Structured Data. In *21st International Conference on Security and Cryptography (SECRYPT), 2024*. https://arxiv.org/pdf/2405.20762