January 2016

Privacy, Access Control, and Integrity for Large Graph Databases

Muhammad Umer Arshad
Purdue University

Graduate School Form 30
(Revised 08/14)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Muhammad Umer Arshad

Entitled: Privacy, Access Control, and Integrity for Large Graph Databases

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

ARIF GHAFOOR, Co-Chair

KRISHNA MADHAVAN, Co-Chair

WALID G. AREF

YUNG-HSIANG LU

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification/Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): ARIF GHAFOOR, Co-Chair
~~KRISHNA MADHAVAN, Co-Chair~~

Approved by: V. Balakrishnan 08/31/2016

Head of the Department Graduate Program Date

PRIVACY, ACCESS CONTROL, AND INTEGRITY FOR LARGE GRAPH DATABASES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Muhammad Umer Arshad

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2016

Purdue University

West Lafayette, Indiana

To my dear parents, Shaista Bano and Arshad Mumtaz, for a lifetime of sacrifice.

To my beloved sisters, for their profound love and prayers.

ACKNOWLEDGMENTS

All praise is due to Allá*h*, Lord of the Worlds, without whom I would not exist, and without whom I would not have begun and could not have completed this dissertation.

To my parents, Shaista Bano and Dr. Arshad Mumtaz, I thank you for the large sacrifices you made for me. The living example of your lives taught me how to succeed in doing hard things, and inspired me to reach this far. I often thought this task was beyond me, but you both knew it was within my reach. My mother's prayers and her words of wisdom guided me through tough times during this journey. Despite limited resources, my father's selfless life of hard work, devotion, and dedication to provide quality education to all of his children has been my inspiration. He fostered in me a tenacious drive to learn and purse excellence in all I should do. His exemplary passion for teaching students of medicine is highly admired by his students, and I hope someday I may be a small fraction so well regarded. I love you dad so much. My parents, I fall short of words to express my gratitude towards you.

To my dear sisters, Dr. Ayesha, Amana, and Fatima, I will always be there for you and am glad each of you went a long way during all those years I have been away. My time with Amana here at Purdue University was the best time I have had here. To my uncle, Dr. Muhammad Naeem Ayyaz, I convey my utmost respect and admiration of him for always sharing his words of wisdom with me. To my maternal aunts, I thank them for the pure love they have given me.

To my advisors, Prof. Arif Ghafoor and Prof. Krishna Madhavan, I express my deepest gratitude for their invaluable guidance and support at each step of my graduate studies at Purdue University. Prof. Ghafoor gave me the freedom to explore the field while training me to become an independent thinker, and constantly encouraged me to dig deeper into my research ideas. From him, I also learned the skill which is the capacity to be useful to others. I am forever indebted to Prof. Krishna Madhavan

for his unequivocal help and encouragement, especially during the later part of my Ph.D. He is an effective communicator, a strategic thinker, and above all a very nice human being. Thank you again Prof. Madhavan, for channeling my energies in the right direction to accomplish my dissertation.

I have benefited immensely from Prof. Walid Aref's unique approach to systems-oriented research. The influence of his work has been instrumental to my success at multiple industrial positions. He has been a great source of inspiration, and I have always learned from him in our research interactions. He has had a profound impact on my thought process and overall personality.

I thank Prof. Yung-Hsiang Lu for serving on my doctoral advising committee as well as on all my examining committees. His questions were always intriguing, and in formulating my reply, they helped me to clarify my ideas. I appreciate his candid constructive feedback, comments, and our many insightful discussions.

I am also fortunate to have worked with Prof. Elisa Bertino from whom I learned much despite our short research interaction. She always respected and encouraged me, and it was a pleasure interacting with her. I am grateful for her continuing support and am deeply moved by her strong work ethics.

I am obliged to Dr. Ashish Kundu, IBM T. J. Watson Research Center, for introducing me to some challenging research problems and for being an excellent research collaborator.

My internships at VMWare, Yahoo!, and EMC2 have been a great learning experience for me. I thank my mentors, Dr. Raj Yavatkar, Don Newell, Boaz Shaham, Lars Anderson, Tavit Ohanian, and Ann Wong for many stimulating discussions.

My special thanks go to Dr. Nasir Bilal for always motiving me, for introducing me to some life-changing books, for being my swimming instructor, and for all the wonderful times we have had together at Purdue University.

So many more colleagues and friends must go unacknowledged here. Know that you are remembered for your friendship and support.

In closing, I restate that all praise is due to Allá*h*, Lord of the Worlds.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

PPM	Privacy Protection Mechanism
ACM	Access Control Mechanism
HMAC	Hash Message Authentication Code
PPT	Probabilistic Polynomial Time

ABSTRACT

Arshad, Muhammad PhD, Purdue University, December 2016. Privacy, Access Control, and Integrity for Large Graph Databases. Major Professors: Arif Ghafoor and Krishna Madhavan.

Graph data are extensively utilized in social networks, collaboration networks, geo-social networks, and communication networks. Their growing usage in cyberspaces poses daunting security and privacy challenges. Data publication requires privacy-protection mechanisms to guard against information breaches. In addition, access control mechanisms can be used to allow controlled sharing of data. Provision of privacy-protection, access control, and data integrity for graph data require a holistic approach for data management and secure query processing. This thesis presents such an approach. In particular, the thesis addresses two notable challenges for graph databases, which are: i) how to ensure users' privacy in published graph data under an access control policy enforcement, and ii) how to verify the integrity and query results of graph datasets.

To address the first challenge, a privacy-protection framework under role-based access control (RBAC) policy constraints is proposed. The design of such a framework poses a trade-off problem, which is proved to be NP-complete. Novel heuristic solutions are provided to solve the constraint problem. To the best of our knowledge, this is the first scheme that studies the trade-off between RBAC policy constraints and privacy-protection for graph data. To address the second challenge, a cryptographic security model based on Hash Message Authentic Codes (HMACs) is proposed. The model ensures integrity and completeness verification of data and query results under both two-party and third-party data distribution environments. Unique solutions based on HMACs for integrity verification of graph data are developed and detailed security analysis is provided for the proposed schemes. Extensive experimental evaluations are conducted to illustrate the performance of proposed algorithms.

1. INTRODUCTION

1.1 Motivation

Graph data has become increasingly important in recent years because of its widespread use in various applications. Some leading examples are online social networks (OSN), Web, collaboration networks, and communication networks [1]. The nodes of a graph represent entities, while their connections capture various relationships among them. The semantics assigned with nodes and links in the graph data vary significantly across application domains. For example, a social network is usually represented by a set of users, where links may capture friendship relationships; a co-authorship network, on the other hand, describes scientific publications and their collaboration links, etc.

The analysis of published graph data is used extensively by researchers in different disciplines to extract useful knowledge and information. For example, epidemiologists study disease spread patterns based on users' social contact information; sociologists and psychologists can verify the social structure and human behavior pattern; mining algorithms are used to discover various patterns in these graphs; and advertisers can accurately infer users' preference profiles for targeted advertisements [2], [3]. This data is published to stakeholders and authorized users.

Due to strong correlation among users' social identities, privacy poses a major challenge in data storage, processing, and publishing. The sensitive nature of data raises privacy challenges as users' private information may be revealed in published graph data [4]. Privacy-preservation for sensitive data entails enforcement of privacy policies and the provision for sufficient protection against *identity disclosure* [5].

Data anonymization has been studied extensively and adopted widely for protecting users' privacy in graph data publishing [2], [3]. Simply removing the node

identifiers in social networks does not provide protection against structure-based re-identification attacks [4]. Backstorm et al. [4] present a family of active and passive attacks that work based on the uniqueness of some small random subgraphs embedded in a network. The adversary may link this distinctive structure, random subgraph, to some set of targeted individuals. In the anonymized published graph, the adversary then traces the injected subgraph in the original graph. In case of only one such subgraph in the original graph, the targets that are connected to this subgraph can be successfully re-identified and the edges between them are disclosed. In [6], Narayanan et al. present a scalable two-phase de-anonymization (DA) process for social networks. In the first phase, some seed nodes are identified between the anonymized and auxiliary graphs. In the second phase, the identified seed nodes are used in an iterative DA propagation process based on both graphs' structural characteristics. A detailed comparison of different protection schemes against de-anonymization attacks is given in [7].

Due to the high cost of hosting large volumes of data and performing data-intensive computations, the *owners* of graph databases often outsource their data to a third-party service provider [8] that offers data services on behalf of the data *owners* [9]. Generally, outsourcing also offers performance-oriented and scalable data services [10]. A leading example is the cloud computing paradigm. Other examples include Amazon EC2, Amazon AWS, Google Cloud Service, and "Database-as-a-Service" [8], [10], [11].

However, data outsourcing can pose serious data security challenges. The biggest challenge is to ensure integrity of the data in the presence of untrusted service providers [8], [9], [12]. Any tampering with data or query results presented to a user can be perceived by the user as a violation of the Quality of Service (QoS) [13] integrity requirements. However, verifying the integrity of graph data poses a significant security challenge [14].

1.2 Research Contributions

In this dissertation, we address the aforementioned challenges of privacy, access control, and data integrity for graph datasets. In particular, we make two main research contributions.

A privacy mechanism for access-controlled graph databases: A framework for privacy-enhanced access-controlled graph dataset is presented. The framework provides privacy protection through k-anonymization under the access restrictions imposed by the RBAC policy. The k-anonymous bi-objective graph partitioning (k-BGP) problem is formulated and hardness results are presented. Efficient heuristics have been developed to solve the problem. A detailed security analysis of the scheme is conducted and the proposed algorithms has been empirically evaluated.

Integrity verification of data and query results for graph databases: Two security notions – HMACs for graphs for two-party data sharing, and redactable HMACs for graphs for third-party data sharing are developed. The proposed schemes can support "fail-stop" and "fail-warn" integrity assurance (Section 4.2.4) mechanisms that can result in substantial saving in the cost incurred for integrity verification and data re-transfer of compromised graphs. Formal definitions and constructions of HMACs for graphs and redactable HMACs for graphs are provided. The proposed schemes are shown to be secure to protect the graphs and redacted graphs from being compromised. Experimental results on real-world graph datasets demonstrate that HMACs for graphs and redactable HMACs graphs are highly efficient compared to digital signature-based schemes for graphs and the proposed schemes are linear in the number of vertices and edges in the graph. Therefore, the proposed schemes are efficient both in processing time and in the transmission of result set $\mathcal{R}$ and verification objects $\mathcal{VO}$ to the client/user.

1.3 Organization

The remainder of this dissertation is organized as follows:

In Chapter 2, relevant background concepts related to privacy, role-based access control (RBAC), and Hash Message Authentication Codes (HMACs) are introduced. In Chapter 3, a framework for privacy-preserving access-controlled graph datasets is presented. Privacy and access constraints are formulated as the k-anonymous bi-objective graph partitioning (k-BGP) problem. Hardness results are presented and empirical evaluation is conducted for the proposed heuristics. In Chapter 4, the problem of graph data and query results integrity verification using HMACs is investigated. Efficient integrity verification schemes are proposed. A detailed security analysis of schemes is presented along with empirical evaluation on real-world graph datasets. Chapter 5 concludes the dissertation with suggestions for future work.

2. BACKGROUND

2.1 The Data Model

We consider a simple undirected graph data model, $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of nodes and $E \subseteq \binom{V}{2}$ is the set of edges[1]. Each node corresponds to an individual in the underlying group of people, while an edge that connects two nodes describes a relationship between two corresponding individuals.

In addition to the *structural data* that is given by E, each node is described by a set of attributes (*descriptive data*) that can be classified in the following three categories:

- *Identifier.* Attributes, e.g., name and ssn, that uniquely identify an entity. These attributes are completely removed from an anonymized graph.
- *Quasi-identifier (QI).* Attributes, e.g., birth date, zip code and gender, that can be joined with external information available to some adversary to reveal the personal identity of an individual.
- *Sensitive attribute.* Attributes, e.g., disease and income, that are assumed to be unknown to an intruder. They are assumed to cause a privacy breach if associated to a unique individual.

The combination of QIs could be used for unique identification by mean of *linking attacks* [15]. Hence, they should be generalized in order to thwart such attacks.

Definition 2.1.1 *Let $A_1, A_2, \ldots, A_d$ be a collection of QI attributes. Then a graph is defined as $\mathcal{G} = \langle V, E, \mathcal{T} \rangle$, where $E \in \binom{V}{2}$ is the structural information (edges),*

[1] $\binom{V}{2}$ denotes the set of all unordered pairs of elements from V.

Table 2.1.: Generalization for k anonymity

	$\mathbf{QI_1}$	$\mathbf{QI_2}$	$\mathbf{S_1}$
ID	**Age**	**Zip**	**Income**
1	10	25	120,000
2	20	35	95,000
3	30	45	110,000
4	35	15	150,000
5	40	40	290,000
6	50	60	75,000
7	55	20	225,000
8	60	55	350,000
9	65	25	175,000

(a) Sensitive table $\mathcal{T}$

$\mathbf{QI_1}$	$\mathbf{QI_2}$	$\mathbf{S_1}$
Age	**Zip**	**Income**
10-20	25-35	120,000
10-20	25-35	95,000
30-40	40-45	110,000
35-65	15-25	150,000
30-40	40-45	290,000
50-60	55-60	75,000
35-65	15-25	225,000
50-60	55-60	350,000
35-65	15-25	175,000

(b) 2-anonymous table $\overline{\mathcal{T}}$

describing relationships between V pairs, and $\mathcal{T} = \{T_1, \ldots, T_N\}$, where $T_i \in A_1 \times \ldots \times A_d, 1 \leq i \leq N$ are the descriptive data associated with nodes in V.

2.2 Graph Anonymization Definitions

Consider the anonymization of a given graph $\mathcal{G} = \langle V, E, \mathcal{T} \rangle$ by partitioning as given in [16], [17], [18]. Let $V_{\mathcal{P}} = \mathcal{P} = \{P_1, \ldots, P_M\}$ be a partition of V into disjoint subsets or partitions, i.e., $V = \bigcup_{i=1}^{M} P_i$ and $P_i \cap P_j = \phi$ for all $1 \leq i \neq j \leq M$, and $E_{\mathcal{P}} \subseteq \binom{V_{\mathcal{P}}}{2}$ be a set of edges on $V_{\mathcal{P}}$, where $\{P_i, P_j\} \in E_{\mathcal{P}}$ iff there exists $v_n \in P_i$ and $v_m \in P_j$ such that $\{v_n, v_m\} \in E$.

Definition 2.2.1 (k-Anonymity Property) *A graph satisfies the k-anonymity property if each partition $P_i \in \mathcal{P}$ contains k or more nodes [16].*

Definition 2.2.2 (Super-node) *In the anonymized published graph (e.g., Fig. 3.1(b)), each partition, say $P_i \in \mathcal{P}$, is replaced by a pair of items, $(|P_i|, |E_{P_i}|), 1 \leq i \leq M$, where $|P_i|$ is the number of nodes in a particular partition (i.e., the number of original V-nodes as part of that partition), and $|E_{P_i}|$ is the number of edges in E that connect nodes within Partition $P_i, 1 \leq i \leq M$. This new published node is termed super-node.*

Definition 2.2.3 (Super-edge) *In the anonymized published graph (e.g., Fig. 3.1(b)), each edge, say $(P_i, P_j) \in E_{\mathcal{P}}$, is labeled by a weight $|E_{P_i,P_j}|$, which stands for the number of edges in E that connect a node in P_i to a node in P_j. This new published edge is termed a super-edge.*

We assume that all the QI attributes have numerical values and use the hierarchical-free generalization [19] that generalizes the set of tuples present in a partition, say P_i, with the smallest interval that includes all the initial values, also called the *minimal covering tuple*, for that partition.

Definition 2.2.4 (Anonymized graph [16]) *Let $\mathcal{G} = \langle V, E, \mathcal{T} \rangle$ be a graph with vertex attributes, and let $\overline{A}_1, \ldots, \overline{A}_d$ be the generalization taxonomies for d QI attributes $A_1, \ldots, A_d$. Then, given a partitioning $V_{\mathcal{P}}$ of V, the anonymized graph is defined as $\mathcal{G}_{\mathcal{P}} = \langle V_{\mathcal{P}}, E_{\mathcal{P}}, \overline{\mathcal{T}} \rangle$, where:*

- *$E_{\mathcal{P}} \subseteq \binom{V_{\mathcal{P}}}{2}$ is a set of edges on $V_{\mathcal{P}}$, where $\{P_i, P_j\} \in E_{\mathcal{P}}$ iff there exists $v_n \in P_i$ and $v_m \in P_j$ such that $\{v_n, v_m\} \in E$;*
- *The partitions in $V_{\mathcal{P}}$ are labeled by their sizes and the number of their intra-cluster edges $(|P_i|, |E_{P_i}|)$, while the edges in $E_{\mathcal{P}}$ are labeled by the corresponding number of inter-cluster edges, $|E_{P_iP_j}|$, in E where $1 \leq i \neq j \leq M$;*
- *$\overline{\mathcal{T}} = \{\overline{T}_1, \ldots, \overline{T}_M\}$, where $\overline{T}_i$ is the minimal record in $\overline{A}_1 \times \ldots \times \overline{A}_d$ that generalizes all QI tuples of individuals in $P_i, 1 \leq i \leq M$. Table 2.1(b) shows a 2-anonymous partitioning for a dataset with QI attributes Age and Zip.*

2.3 Role-based Access Control

Role-based access control (RBAC) allows defining permissions on objects based on roles in an organization. An RBAC policy configuration is composed of a set of Users ($\mathcal{U}$), a set of Roles ($\mathcal{R}$), and a set of Permissions ($\mathcal{P}$). For the graph model, we assume that the set of permissions for a role are the selection predicates on the QI attributes

Table 2.2.: Access control policy

Role	Permission	Authorized Query Predicate (View)
Role1	X	Age = 15-45 $\wedge$ Zip = 20-30
Role2	Y	Age = 30-45 $\wedge$ Zip = 25-45
Role3	Z	Age = 50-60 $\wedge$ Zip = 55-60

that the role is authorized to execute [20]. Among the authorized tuple subset, a user is free to set any selection condition on the sensitive attribute. The user-to-role assignment (UA) is a user-to-role ($\mathcal{U} \times \mathcal{R}$) mapping and the role-to-permission assignment (PA) is a role-to-permission ($\mathcal{R} \times \mathcal{P}$) mapping.

Definition 2.3.1 (RBAC Policy) *An RBAC policy ρ is a tuple $\langle \mathcal{U}, \mathcal{R}, \mathcal{P}, UA, PA \rangle$.*

In practice, when a user assigned to a role executes a query, the tuples that satisfy the conjunction of query predicate and the permission are returned [5], [21]. Consider for example Table 2.2 where Role1 has been assigned permission X with authorized query predicate Age = 15-45 $\wedge$ Zip = 20-30.

2.4 Message Authentication Codes (MACs)

A MAC is a cryptographic checksum on data that takes as input a message m and a secret key k and produces an output called *authentication* tag $t = H(k, m)$.

The Hash Message Authentication Code (HMAC) algorithm is a shared-key security algorithm that uses a cryptographic hash function as an underlying function and is used to verify data integrity and data-origin authentication. HMAC can be used with any iterative cryptographic hash function (e.g., MD5, SHA-1, etc.) in combination with a shared secret key. HMAC has been implemented in widely used security protocols including SSL, TLS, SSH, and IPsec [22]. It is also used as a PRF[2] for key-derivation, as in TLS [23] and IKE (the Internet Key Exchange protocol of

[2]A PRF is an efficient deterministic function and takes two inputs k and m. Its output is computationally indistinguishable from truly random output.

IPsec) [24]. HMAC is also used as a PRF in a standard for one-time passwords [25]. This is the basis for Google authenticator . The main operation of HMAC is:

$$HMAC_k(m) = H((k \oplus \texttt{opad})||H((k \oplus \texttt{ipad})||m)), \tag{2.1}$$

where `opad` (outer padding) is a constant byte `0x36`, `ipad` (inner padding) is a constant byte `0x5c` [26], and $\oplus$ is bitwise eXclusive-OR (X-OR) operator.

Attacks

The most common attack on MACs is a *forgery attack*, in which an adversary can produce a valid (message, tag) pair without knowing the secret key k. For MACs that are based on *iterative* hash functions and use a compression function $f : \{0,1\}^{n+m} \rightarrow \{0,1\}^n$, there is a birthday-type forgery attack [27] that requires about $\mathcal{O}(2^{n/2})$ MAC queries to its generation oracle, where n is the length of authentication tag.

Security

The cryptographic strength of HMAC depends on the properties of the underlying hash function [28]. As we have mentioned, the most common attack against HMACs is brute force to uncover the secret key. To have a secure MAC function, we want to have *unforgeability*; that is, without knowing the secret key k, it should be hard for an adversary $\mathcal{A}$ to find a pair (m, t) such that $t = MAC_k(m)$, even if $\mathcal{A}$ has access to some other valid (message, tag) pairs. Unfortunately, for a secure hash function $MAC_k(m) = H(k||m)$ does not guarantee that the MAC function is unforgeable. Since H is computed using the Merkle–Dagmard construction, the graph MAC designed in this way is completely insecure, as it is quite easy, given a valid pair (m, t), to create an (m', t'), which is still valid. HMAC $HMAC_k(m) = H((k \oplus \texttt{opad})||H((k \oplus \texttt{ipad})||m))$ avoids the above problem using *two* layers of hashing [26].

3. A PRIVACY MECHANISM FOR ACCESS CONTROLLED GRAPH DATA

3.1 Introduction

Data anonymization schemes provide privacy-protection for published graph data. However, the data publisher may use an *authorization mechanism* for controlling access to data by group of users [29]. Access control policies provide additional safeguard against data breaches and are used to ensure that only authorized published information is available to end-users based on their assigned role. Roles are abstract descriptions of privileges for users accessing data in OSNs [30]. We assume a *Role-Based Access Control* (RBAC) [31] administration model for the policy enforcement. RBAC assigns access privileges to end-users based on their predefined roles. A leading example in OSN services is Facebook[1], which provides privacy features by allowing the user to dictate access to their private information by employing fine-grained access control policies [32], [33]. In OSN, either a centralized authority, a reference monitor, decentralized authorities, or users themselves can carry out policy enforcement. We consider a graph data publishing framework that provides safeguard against data privacy breach through anonymization while enforcing access rules to satisfy the security protection requirements specified by the data publisher.

Since k-anonymization is a generalization approach, at the time of creating k-anonymous partitions, we show that access control privileges might need be *relaxed* to ensure k-anonymity privacy requirement with a relatively stronger guarantee. The issue is, in order to accommodate imprecision bound false-positive tuples need to be reduced that result in increased average partition sizes. Relaxing access control requirement implies a slight increase in the scope of the privilege set associated with a

[1] `https://www.facebook.com/policy.php`

role. Likewise, under *strict* policy the privacy is relatively weak compared to relaxed semantics as we try to reduce false-negative tuples resulting in decreased average partition sizes. This exhibits a trade-off between privacy and access control. However, relaxing access control requirement should be bounded by access control administrator. Discussion on access control model and policies is given in Sections 3.2.1 and 3.3.2. Generally, high privacy is achieved at the cost of increased information loss [34]. A key challenge is to ensure k-anonymity privacy protection of individuals within published graph data and preserve data utility while enforcing an access control policy. Formally, given a set of roles with their associated imprecision bounds and a k-anonymity requirement, the challenge is anonymize dataset such that maximum number of roles satisfy their imprecision bounds and minimum information loss is incurred. For this, we propose a k-anonymous Bi-objective Graph Partitioning (k-BGP) problem and give hardness results (Section 3.3.1). This is a unique problem that has not been considered earlier.

The chapter makes the following contributions:

- We formulate the k-BGP problem and give hardness results. Two heuristics TSH1 and TSH2 are developed to solve the constraint problem.
- We provide empirical evaluation of the proposed heuristics with a benchmark algorithm [35] from design perspective in terms of meeting privacy and access control requirements with minimum information loss.
- Within the context of k-BGP problem, we present an architecture framework elaborating how access control and privacy can be integrated (Section 3.3.2).
- We evaluate the proposed framework from security perspective and present a probabilistic analysis for re-identification risk.

The rest of this chapter is organized as follows. Section 3.2 presents the needed definitions and discusses the information loss measure. The problem formulation and the access control framework are discussed in Section 3.3. In Section 3.4, we present

the proposed heuristics for k-BGP problem. Section 3.5 provides performance evaluation and security analysis. Section 3.6 overviews the related work and Section 3.7 contains the summary of research contributions.

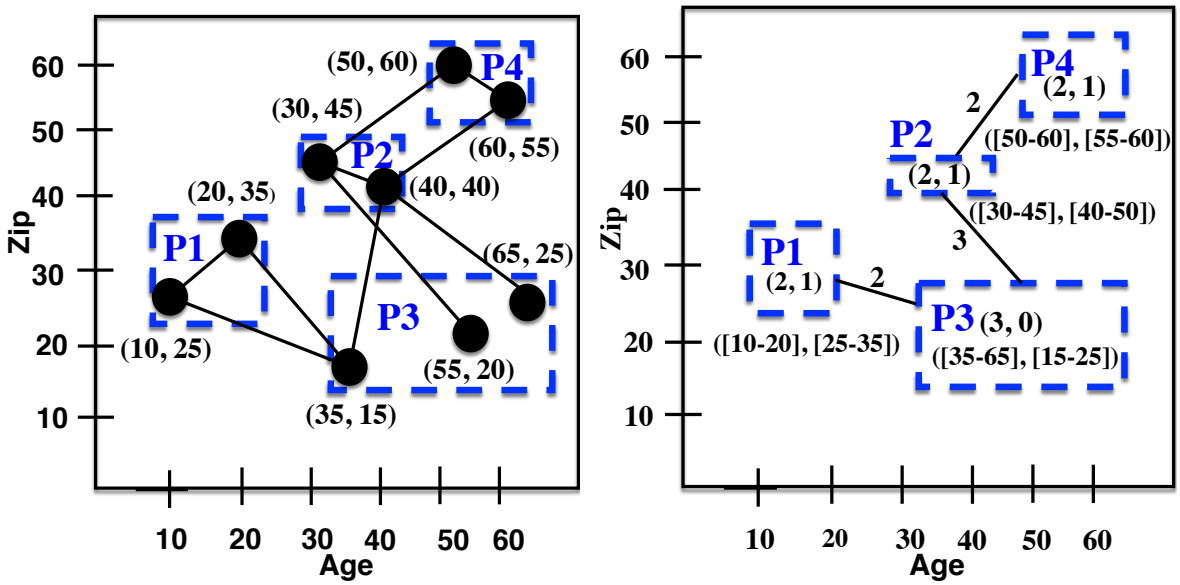
3.2 Background

3.2.1 Access Control Model for Graph Data

In this section, we discuss the semantics of role/query predicate evaluation with respect to access control. For the query predicate evaluation over a graph, say $\mathcal{G}$, a vertex is added to the output result if all its attribute values satisfy the query predicate. Moreover, the edges between the result vertex set are also returned as an output. Here, we only consider conjunctive queries, where each query represents the d-dimensional hyper-rectangle. The semantics for query evaluation on an anonymized graph $\mathcal{G}_{\mathcal{P}}$ need to be defined. When a partition, say P, is fully included in the query region, all the partition nodes and their associated edges are returned as part of the query result. However, when a partition and a query partially overlap, there is an uncertainty in the query evaluation. In this case, there can be several possible semantics. The following three options are generally used:

1. *Uniform.* Assuming the uniform distribution of nodes in the overlapping partitions, the result returns the nodes according to the ratio of overlap between the query and the partition, and the edges between these nodes. Most of the literature uses the uniform distribution semantics to compare anonymity techniques over selection tasks [19].

2. *Overlap.* This includes all nodes and their associated edges in the partitions that overlap the role/query. This option will add false positives to the original role/query result.



(a) Original partitioned graph

(b) Anonymized graph $\mathcal{G}_{\mathcal{P}}$

Fig. 3.1.: A graph and its corresponding published view.

3. *Enclosed.* This discards all nodes and their associated edges in all those partitions that partially overlap the role/query region. This option yields false negatives with respect to the original role/query result.

For the remainder of this paper, we assume *Overlap semantics* as defined above.

Example 1 *Consider the social network graph in Fig. 3.1(a) with 9 vertices and 10 edges with each vertex containing the two QI attribute values Age and Zip for individuals in the graph. Table 2.1(b) shows the 2-anonymous partitioning of these vertex attributes. In the anonymized published graph in Fig. 3.1(b), Partition (supernode)* P_1 *contains two verities and one edge represented as* $(2, 1)$*; moreover, the min and max values of the QI attributes are represented as a generalized tuple* $([10-20, 25-35])$*. similarly, Partition* P_2 *is represented by the pair* $(2, 1)$ *and the generalized tuple* $([30-40, 40-45])$*;* P_3 *is represented by the pair* $(3, 0)$ *and the generalized tuple* $([35-65, 15-25])$*; and Partition* P_4 *is represented by the pair* $(2, 1)$ *and generalized tuple*

Table 3.1.: Published graph view for roles

Roles	**Super Nodes**	**Generalized Tuples**		**SuperEdges**
		Age	**Zip**	
Role1 → X	$P_1(2,1)$	10-20	25-35	$\|E_{P_1P_3}\| = 2$
	$P_3(3,0)$	35-65	15-25	
Role2 → Y	$P_2(2,1)$	30-40	40-45	$\|E_{P_2P_3}\| = 3$
	$P_3(2,1)$	35-65	15-25	
Role3 → Z	$P_4(2,1)$	50-60	55-60	NULL

$([50-60, 55-60])$. *Now, consider the inter-partition edges in the published anonymized graph. There are two edges between Partitions* P_1 *and* P_3 *represented by* $|E_{P_1,P_3}| = 2$*; Similarly,* $|E_{P_2,P_3}| = 3$ *and* $|E_{P_2,P_4}| = 2$*. According to an access control policy, as given in Table 2.2, with permission set* $\{X, Y, Z\}$ *and its associated authorized query predicates, the published graph view for role set* {*Role1, Role2, Role3*} *is as given in Table 3.1. Since permission* X *assigned to Role1 overlaps two Partitions* P_1 *and* P_3*, Role1 gets access to two super-nodes* $P_1(2,1)$ *and* $P_3(3,0)$ *and one super-edge* $|E_{P_1P_3}| = 2$ *as part of the published graph along with their generalized tuples. Notice that the published super-nodes contain the information about the number of nodes and edges present within the partition. However, the access control policy ultimately determines how much access to shared published data is allowed.*

In this section, we give the definitions for role imprecision bound and describe the information loss measure for the whole anonymized graph data.

3.2.2 Imprecision Bound for Roles

Let v_n be a vertex in graph $\mathcal{G}$ with d QI attributes, $A_1, \ldots, A_d$. Vertex v_n can be expressed as a d-dimensional vector $\{v_n(1), \ldots, v_n(d)\}$, where $v_n(j)$ is the value of the jth attribute. Let D_{A_i} be the domain of QI attribute QI_i, then $v_n \in D_{A_1} \times \ldots \times D_{A_d}$. Any d-dimensional partition P_i of the QI attribute domain space can be defined as a d-

dimensional vector of closed intervals $\{I_1^{P_i}, \ldots, I_d^{P_i}\}$. The closed interval $I_j^{P_i}$ is further defined as $[a_j^{P_i}, b_j^{P_i}]$, where $a_j^{P_i}$ is the start of the interval and $b_j^{P_i}$ is the end of interval. To publish a partition, each node v_n in a Partition, say P_i, is replaced by the minimum bounding intervals $\{I_1^{P_i}, \ldots, I_d^{P_i}\}$ of the partition to which the node belongs. A vertex, say v_n, belongs to a Partition, say P_l, if $\forall v_n(i), v_n(i) \in I_i^{P_l} : a_i^{P_l} \leq v_n(i) \leq b_i^{P_l}$.

Consider a set of roles $\mathcal{R}$, where $R_i \in \mathcal{R}$ is defined by a Boolean function of predicates on the set of QI attributes $A_1, \ldots, A_d$. A role defines a space in the domain of QI attributes $D_{A_1} \times \ldots \times D_{A_d}$ and can be represented by a d-dimensional rectangle or a set of non-overlapping d-dimensional rectangles. To simplify the notation, we assume that a role, say R_j, is a single d-dimensional rectangle represented by $\{I_1^{R_j}, \ldots, I_d^{R_j}\}$. A vertex, say v_n, belongs to R_j if $\forall v_n(i), v_n(i) \in I_i^{R_j} : a_i^{R_j} \leq v_n(i) \leq b_i^{R_j}$. Role R_j and Partition P_l overlap if $\forall I_i^{R_j}, \forall I_i^{P_l}, a_i^{R_j} \in I_i^{P_l}$ or $a_i^{P_l} \in I_i^{R_j}$.

Definition 3.2.1 (Role Imprecision) *Role imprecision is defined as the difference between the number of nodes returned by a role/query evaluated on an anonymized graph $\mathcal{G}_\mathcal{P}$ and the number of nodes for the same role/query on the original graph $\mathcal{G}$. The imprecision for role/query R_i is denoted by I^{R_i},*

$$I^{R_i} = |R_i(\mathcal{G}_\mathcal{P})| - |R_i(\mathcal{G})|, \text{ where}$$
$$|R_i(\mathcal{G}_\mathcal{P})| = \sum_{\forall P_j \in \mathcal{P} \text{ overlaps } R_i} |P_j|.$$

The Role R_i is evaluated over $\mathcal{G}_\mathcal{P}$ by including all the nodes in the $P \in \mathcal{P}$ that overlap the role region.

Definition 3.2.2 (Role Imprecision Bound) *The role imprecision bound, denoted by B^{R_i}, is the maximum tolerable imprecision by a a role R_i and is preset by the access control administrator.*

3.2.3 Information Loss and Utility Measure for Anonymized Graph Data

Given a graph, say $\mathcal{G} = \langle V, E, \mathcal{T} \rangle$, and a partitioning, say $\mathcal{P}$, of $\mathcal{G}$'s nodes, the information loss $IL(\mathcal{P})$ associated with replacing $\mathcal{G}$ by the corresponding partitioned network, $\mathcal{G}_{\mathcal{P}} = \langle V_{\mathcal{P}}, E_{\mathcal{P}}, \overline{\mathcal{T}} \rangle$, is defined as the weighted sum of two metrics,

$$IL(\mathcal{P}) = w.IL_D(\mathcal{P}) + (1 - w).IL_S(\mathcal{P}), \tag{3.1}$$

where $w \in [0, 1]$ is a weighting parameter, $IL_D(\mathcal{P})$ is the *descriptive* information loss that is caused by generalizing the exact QI records $\mathcal{T}$ to $\overline{\mathcal{T}}$, while $IL_S(\mathcal{P})$ is the *structural* information loss that is caused by collapsing all nodes of V in a given partition of $V_{\mathcal{P}}$ to one super-node.

We use the same measure of information loss as proposed in [16]. For the descriptive information loss, we utilize the Loss Metric (LM) measure [36], [37]. Assume that an original node, say $v_n \in V$, belongs to a partition, $P_i \in \mathcal{P}$; then v_n's QI record, $T_n = (T_n(1), \ldots, T_n(d))$, is generalized to $\overline{T}_i = (\overline{T}_i(1), \ldots, \overline{T}_i(d))$, where d is the number of QI attributes. The LM associates the following loss of information with each of the nodes in a partition, say P_i,

$$IL_D(P_i) = \frac{1}{d} \sum_{j=1}^{d} \frac{|\overline{T}_i(j)| - 1}{|A_j| - 1}, \tag{3.2}$$

where $|\overline{T}_i(j)|$ is the size of the subset $\overline{T}_i(j)$ that generalizes the original value $T_n(j)$, and $|A_d|$ is the number of values in the domain of attribute A_d.

Notice that $IL_D(P_i)$ ranges between zero and one, where $IL_D(P_i) = 0$ iff all records in P_i are equal, and no generalization is applied, while $IL_D(P_i) = 1$ iff all records in P_i are so far off that all attributes in the generalized record have to be totally suppressed. The overall LM information is the result of averaging $IL_D(P_i)$ for all partitions in $\mathcal{P}$, i.e.,

$$IL_D(\mathcal{P}) = \frac{1}{N}. \sum_{i=1}^{M} |P_i|.IL_D(P_i). \tag{3.3}$$

No generalization means maximum descriptive data utility, $U_D(\mathcal{P})$. Hence, $U_D(\mathcal{P})$ is defined as $U_D(\mathcal{P}) = 1 - IL_D(\mathcal{P})$.

Structural information loss can be categorized into two types:

- **Intra-partition information loss**: Given a partition, say $P_i \in \mathcal{P}$, the structure of P_i in the original graph is lost, and is replaced by the number of nodes in P_i, and the number $|E_{P_i}|$ of edges in E that connect nodes in P_i. The corresponding information loss is quantified as the probability of wrongly identifying a pair of nodes in P_i as an edge or as a non-connected pair, and it is evaluated as follows:

$$IL_{S,1}(P_i) = 2|E_{P_i}|.\left(1 - \frac{2|E_{P_i}|}{|P_i|(|P_i|-1)}\right). \tag{3.4}$$

- **Inter-partition information loss**: Given two partitions, say $P_i, P_j \in \mathcal{P}$, the structure of edges that connect nodes from P_i to nodes in P_j is lost, and is replaced by the number $|E_{P_iP_j}|$ of edges between nodes in these two partitions. The inter-partition information loss is quantified as the probability of wrongly identifying a pair of nodes in P_i and P_j as an edge or as a non-connected pair, and is evaluated as follows:

$$IL_{S,2}(P_i, P_j) = 2.|E_{P_i,P_j}|.\left(1 - \frac{|E_{P_i,P_j}|}{|P_i||P_j|}\right). \tag{3.5}$$

Then, the overall structural information loss for partitioning $\mathcal{P} = \{P_1, P_2, \ldots, P_M\}$ is evaluated as follows:

$$IL_S(\mathcal{P}) = \frac{4}{N(N-1)}\left[\sum_{i=1}^{M} IL_{S,1}(P_i) + \sum_{1\le i\ne j\le M} IL_{S,2}(P_i, P_j)\right], \tag{3.6}$$

where the normalizing factor $\frac{4}{N(N-1)}$ guarantees that $IL_S(\mathcal{P})$ ranges between zero and one. The maximal value of one occurs when all edge counters ($|E_{P_i}|$ and $|E_{P_i,P_j}|$) fall in the middle of the intervals where they range (i.e., $|E_{P_i}| = \binom{|P_i|}{2}/2$) and $|E_{P_i,P_j}| = |P_i||P_j|/2$ for all $1 \le i \ne j \le M$).

In an anonymized Graph, say $\mathcal{G}_{\mathcal{P}} = \langle V_{\mathcal{P}} = \mathcal{P}, E_{\mathcal{P}}, \overline{\mathcal{T}} \rangle$, the structural utility $U_S(\mathcal{P})$ is defined as $U_S(\mathcal{P}) = 1 - IL_S(\mathcal{P})$. A generalized graph summarizes the structure of the original graph. Let us consider two extreme cases:

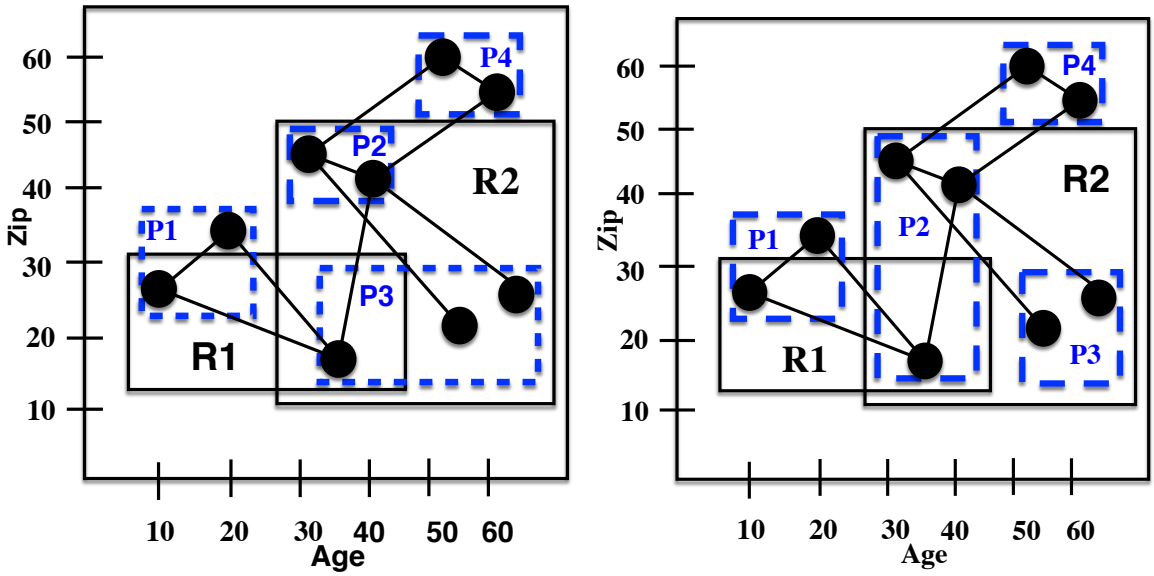
One-to-one correspondence between nodes and super-nodes: This means each super-node contains only one node (i.e., no intra-edge) and a pair of super-nodes does not contain more than one inter-edge. The original graph structure is maintained as it is. According to the structural loss formulation, the intra-partition loss, $IL_{S,1}(P_i) = 0$, for each partition as there is no intra-edge present within super-nodes; similarly, the inter-partition loss, $IL_{S,2}(P_i, P_j) = 0$, for all super-node pairs as there is at most one inter-edge present between them. This results in $IL_S(\mathcal{P}) = 0$. Thus, the minimum structural loss $IL_S(\mathcal{P})$ value corresponds to maximum structural utility U_S.

Generalized graph contains a single super-node: Under this case, the only information revealed about the input graph is its size (number of nodes) and density (number of edges). The user has absolutely no structural information available; hence we have very low structural utility U_S value. In this case, inter-partition loss component, $IL_{S,2}(P_i, P_j) = 0$ as there are no inter-edges. The overall structural loss will be determined by the single super-node, i.e., $IL_S(P) = IL_{S,1}(P) = 2e(1 - \frac{2e}{|P||P-1|})$. Therefore, structural utility can be defined as $U_S = 1 - IL_S(P)$ value, i.e., a higher structural loss means a lower structural data utility and vice versa.

3.3 Problem Description

3.3.1 The k-BGP Problem

We show that finding a k-anonymous graph partitioning that satisfies the role imprecision bounds for the maximum number of roles while achieving minimal overall information loss, $IL(\mathcal{P})$, is NP-hard. The cardinality of a Role, say R_i, is the number



(a) $IL_S = 0.42$ (b) $IL_S = 0.51$

Fig. 3.2.: Satisfying role bounds with minimum structural information loss.

of graph nodes falling within the role bounds. The constants rn and lv define a lower bound on the number of the roles that should satisfy their bounds and an upper bound on the value of information loss that an anonymization scheme is allowed to incur. The decisional version of the k-BGP problem is defined below:

Definition 3.3.1 (Decisional k-BGP Problem) *Given a Graph, say $\mathcal{G} = \langle V, E, \mathcal{T} \rangle$, with the vertices in a d-dimensional space, a set of roles $R_i \in \mathcal{R}$ with imprecision bounds B^{R_i}, and positive constants rn and lv, does there exist a k-anonymous graph partitioning of vertices such that: i) the number of roles satisfying imprecision bounds is greater than the positive constant rn, $1 \leq rn \leq |\mathcal{R}|$, and the total graph information loss, $IL(\mathcal{P})$, is less than the positive constant lv, $1 \leq lv \leq IL(\mathcal{P})$.*

Theorem 3.3.1 (Decisional k-BGP Problem is NP-complete)

Proof Refer to Appendix A. ■

Example 2 *Consider the partition set* $\mathcal{P} = \{P_1, \ldots, P_4\}$ *and the role set* $\mathcal{R} = \{R_1, R_2\}$ *in Fig. 3.2. Both the partitions as given in Figs. 3.2(a) and 3.2(b) satisfy the imprecision bounds of* 3 *and* 0 *for* R_1 *and* R_2*, respectively. We can calculate the structural information loss,* $IL_S(\mathcal{P})$*, of the partitions as follows: Partitions* P_1, P_2*, and* P_4 *have two vertices and one connecting edge between them; their intra structural information loss,* $IL_{S,1}(P_i)$*, is calculated as* $IL_{S,1}(P_1) = IL_{S,1}(P_2) = IL_{S,1}(P_4) = 2 \times 1(1 - \frac{2\times1}{2\times1}) = 0$*, while* P_3 *has three vertices with no connecting edges among them. Hence* $IL_{S,1}$ *is* $IL_{S,1}(P_3) = 2 \times 0(1 - \frac{2\times0}{3\times2}) = 0$*. There are two inter-edges between Partitions* P_1 *and* P_2*. The inter structural information loss,* $IL_{S,2}(P_i, P_j)$*, between Partitions* P_1 *and* P_2 *is calculated as follows:* $IL_{S,2}(P_1, P_2) = 2 \times 2(1 - \frac{2}{2\times3}) = \frac{8}{3}$*; Similarly,* $IL_{S,2}(P_2, P_3) = 3$ *and* $IL_{S,2}(P_2, P_4) = 2$*. Thus, the total structural information loss for the partitioning in Fig. 3.2(a) after being normalized is* $IL_S(\mathcal{P}) = \frac{23}{3} \times \frac{1}{18} = 0.42$*. For Fig. 3.2(b), the intra-structural information loss for all partitions* $IL_{S,1}(P_1) = IL_{S,1}(P_3) = IL_{S,1}(P_4) = 0$ *while* $IL_{S,1}(P_2) = \frac{4}{3}$*. The inter-partition information loss for* $IL_{S,2}(P_1, P2) = IL_{S,2}(P_2, P_3) = IL_{S,2}(P_2, P_4) = \frac{8}{3}$*. The total structural information loss for the partitioning in Fig. 3.2(b) after being normalized is then* $IL_S(\mathcal{P}) = \frac{28}{3} \times \frac{1}{18} = 0.51$*. Thus, both partitions in Fig. 3.2(a) and 3.2(b) satisfy an imprecision bound of* 3 *and* 0 *for roles* R_1 *and* R_2*, respectively. However, the overall global structural information loss of the partitioning in Fig. 3.2(a) is less than that of the partitioning in Fig. 3.2(b). Therefore, the partitioning in Fig. 3.2(a) is more preferable.*

3.3.2 Privacy-Enhanced Access Control

Fig. 3.3 presents a framework for privacy-enhanced access control mechanism for graph data where the arrows represent the direction of information flow. The *Privacy Protection Mechanism* (PPM) ensures that the privacy and role bound requirements are met while incurring a minimal information loss before the sensitive data is made

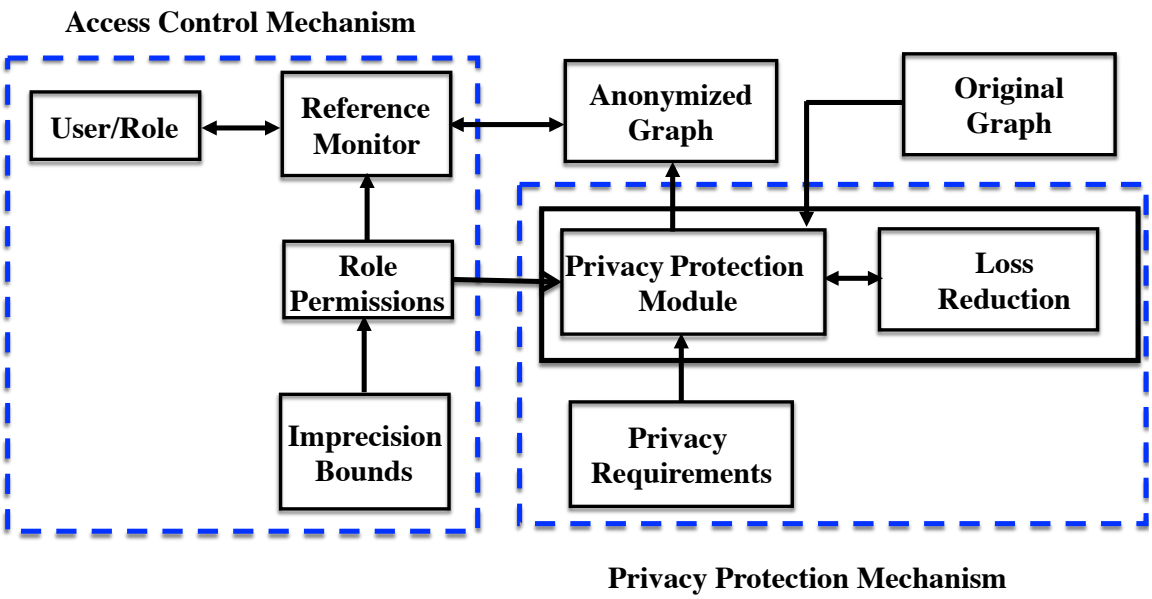


Fig. 3.3.: A framework for the proposed privacy-preserving access control mechanism for graph data.

available to *Access Control Mechanism* (ACM). The Loss Reduction module further minimizes the information loss while keeping the number of roles with satisfied bounds fixed. The permissions in an access control policy are based on selection predicates on the $\mathcal{QI}$ attributes. The policy administrator specifies the permissions along with the imprecision bounds for each permission/role, user-to-role assignments, and role-to-permission assignments [31]. The specification of the imprecision bound ensures that the authorized data has the desired level of accuracy. The imprecision bound information is not shared with the users because knowing the imprecision bound can result in violating the privacy requirement [38].

Access Control Enforcement

Before making the sensitive data available to the access control module, both the descriptive and and structural data of the graph are anonymized. Thus, we need to define the access control enforcement over the anonymized graph data. In this section,

we discuss the Relaxed and Strict access control enforcement policies (employed by the Reference Monitor in Fig. 3.3) over the anonymized graph.

1. *Relaxed:* Relaxed access control uses overlap semantics to allow access to all partitions that overlap a role/ permission.

2. *Strict.* Strict access control uses enclosed semantics to allow access to only those partitions that are fully enclosed by the role/permission.

In this paper, the focus is on relaxed enforcement. In particular, when partitions comprising the shared data between overlapping roles, say $R_i, R_j \in \mathcal{R}$, may contain some non-shared data that is exclusively privileged to an individual role, say R_i; In that case, the scope of the privilege set R_i is slightly increased resulting in relaxed access control mechanism. We refer the reader to [38] for a detailed discussion of these policies.

3.4 Heuristics for the k-BGP Problem

In this section, we present two algorithms based on greedy heuristics for graph anonymization with minimal information loss under a given role/query workload with their associated imprecision bounds. In the first stage, the vertices of the graph $\mathcal{G}$ are partitioned recursively using a kd-tree [39] until the resulting partition sizes are between k and $2k$. The leaf nodes of the kd-tree are the partitions that are mapped to super-nodes in the partitioned graph $\mathcal{G}_{\mathcal{P}}$. The second stage of the heuristics (Algorithm 3) further tries to minimize the information loss by rearranging the vertices across $\mathcal{P}$ partitions under the following constraints: i) the number of role bounds satisfied in first stage is not violated, and ii) each partition satisfies the k-anonymity constraint.

3.4.1 Two-Stage Heuristic 1 (TSH1)

Lemma 3.4.1 *The time complexity of TSH1 is* $\mathcal{O}(d|\mathcal{R}|^2n^2)$.

Algorithm 1: TSH1

Input: $\mathcal{G} = \langle V, E, \mathcal{T} \rangle$, k, $\mathcal{R}$, and B^{R_j}

Output: $\mathcal{G}_{\mathcal{P}} = \langle V_{\mathcal{P}} = \mathcal{P}, E_{\mathcal{P}}, \overline{\mathcal{T}} \rangle$

1 $\mathcal{CP} \leftarrow \mathcal{G}(V))$; `/* Initialize the set of Candidate Partitions. */`

2 **foreach** $CP_i \in \mathcal{CP}$ **do**

3 Find the set RO of roles that overlap CP_i such that $I_{CP_i}^{RO_j} > 0$;

4 Sort roles RO in increasing order of B^{R_j};

5 **while** *the feasible cut is not found* **do**

6 Select role from RO;

7 Create role cuts in each dimension;

8 Select dimension and cut having least overall imprecision for all roles in $\mathcal{R}$;

9 **if** *Feasible cut found* **then**

10 Create new partitions and add to $\mathcal{CP}$;

11 **else**

12 Split CP_i recursively along the median till the anonymity requirement is satisfied ;

13 Compact new partitions and add to $\mathcal{P}$;

14 $\mathcal{G}_{\mathcal{P}}$ = `ConstraintRepartitioning`($\mathcal{G}$, $\mathcal{P}$);

15 return $\mathcal{G}_{\mathcal{P}}$.

Proof The time complexity of the first stage of TSH1 is derived by multiplying the depth of the kd-tree by the amount of work performed at each level. The height of the kd-tree in the worst case is $\frac{n}{k}$, when each partition is exactly of size k. In the worst case, at each partition level, we may have to check all roles for a feasible cut, which leads to a $d|\mathcal{R}|^2 n$ complexity. Thus, the time complexity of the first stage is $\mathcal{O}(d|\mathcal{R}|^2 n^2)$. The time complexity of the second stage, procedure `ConstraintRepartitioning` (Algorithm 3), is $\mathcal{O}(d|\mathcal{R}|n)$. For each partition $P_a \in \mathcal{P}$ the algorithm considers $|\mathcal{P}_{kNN}(P_a)| = 2d$ nearest neighbor partitions[2], [3] as the candidate destination partition $P_b \in \mathcal{P}_{kNN}(P_a)$. This has a time complexity of $2d|\mathcal{P}| \log |\mathcal{P}|$ for all partitions $\mathcal{P}$. The time complexity of procedure `RoleBoundViolations` (Algorithm 4) is $\mathcal{O}(d|\mathcal{R}|n)$ as for each source partition P_a we consider only $|\mathcal{P}_{kNN}(P_a)| = 2d$ neighboring partition for imprecision calculation. Thus, the overall time complexity of the second stage is $\mathcal{O}(d\frac{n}{k} \log \frac{n}{k} + d|\mathcal{R}|\frac{n}{k})$ as $\log \frac{n}{k} << |\mathcal{R}|$, this simplifies to $\mathcal{O}(d|\mathcal{R}|n)$. Adding the time complexities of both stages, the overall complexity of TSH1 is $\mathcal{O}(d|\mathcal{R}|^2 n^2 + d|\mathcal{R}|n) \approx \mathcal{O}(d|\mathcal{R}|^2 n^2)$. ■

3.4.2 Two-Stage Heuristic 2 (TSH2): A Scalable Approach

In the Two-Stage Heuristic 2 algorithm (TSH2, for short), we modify TSH1 so that time complexity of $\mathcal{O}(d|\mathcal{R}|n \log n)$ can be achieved in contrast to the $\mathcal{O}(d|\mathcal{R}|^2 n^2)$ time complexity for TSH1. Because the complexity is subquadratic in network size n and number of roles $\mathcal{R}$, the TSH2 algorithm provides a scalable approach. This heuristic only considers a role with the lowest imprecision bound to check the role cuts for a given Partition, say P_i, and updates the role bounds as the partitions are added to the output. The update is carried out by subtracting the imprecision $I_{CP_i}^{RO_j} > 0$ from the imprecision bound B^{R_j} of each role, for a Partition, say P_i. For example, if a partition of size k has imprecision 10 and 15 for roles R_1 and R_2 with imprecision bound $B^{R_1} = 70$ and $B^{R_2} = 90$, then the bounds are updated to $B^{R_1} = 60$ and

[2]The complexity to find kNN using a Kd-tree is $\mathcal{O}(k \log N)$ [39]
[3]We consider only partition median points while finding the kNN partitions.

Algorithm 2: TSH2: A Scalable Approach

Input: $\mathcal{G} = \langle V, E, \mathcal{T} \rangle$, k, $\mathcal{R}$, B^{R_j}

Output: $\mathcal{G}_\mathcal{P} = \langle V_\mathcal{P} = \mathcal{P}, E_\mathcal{P}, \overline{\mathcal{T}} \rangle$

$\mathcal{CP} \leftarrow \mathcal{G}(V)$); `/* Initialize the set of Candidate Partitions. */`

foreach $CP_i \in \mathcal{CP}$ **do**

  `// Depth-first (preorder) traversal`

  Find the set of roles RO that overlap CP_i such that $I_{CP_i}^{RO_j} > 0$;

  Select role from RO with smallest B^{R_j};

  Create role cuts in each dimension;

  Reject cuts with skewed partitions;

  Select the dimension and the cut having the least overall imprecision for all roles in $\mathcal{R}$;

  if *Feasible cut found* **then**

    Create new partitions and add to $\mathcal{CP}$;

  else

    Split CP_i recursively along the median till anonymity requirement is satisfied ;

    Compact new partitions and add to $\mathcal{P}$;

    Update B^{R_j} according to $I^{R_j}, \forall R_j \in \mathcal{R}$

$\mathcal{G}_\mathcal{P}$ = `ConstraintRepartitioning`($\mathcal{G}$, $\mathcal{P}$);

return $\mathcal{G}_\mathcal{P}$.

Algorithm 3: `ConstraintRepartitioning`

Input: $\mathcal{G} = \langle V, E, \mathcal{T} \rangle$, $\mathcal{P}$

Output: $\mathcal{G}_{\mathcal{P}} = \langle V_{\mathcal{P}} = \mathcal{P}, E_{\mathcal{P}}, \overline{\mathcal{T}} \rangle$

foreach $P_a \in \mathcal{P}$ **do**
  if $|P_a| = k$ **then**
    continue;
  Compute $\mathcal{P}_{kNN}(P_a)$; `/* Determine` k `closest partitions. */`
  $rv_{old} =$ `RoleBoundViolations`$(\mathcal{P}_{kNN}(P_a))$; `/* Compute the number of role bound violations */`
  foreach $|P_b| \in \mathcal{P}_{kNN}(P_a)$ **do**
    if $|P_b| = k$ **then**
      continue;
    else
      $\forall v_a \in P_a$ compute the difference between the information loss, $\Delta^{v_a:a \to b}_{IL(\mathcal{P})}$, if v_n would move from P_a to P_b;
      Let P_c be the partition for which $\Delta^{v_a:a \to b}_{IL(\mathcal{P})}$ is minimal;
      `/* Check privacy constraint. */`
      if $|P_c| + 1 < 2k$ **then**
        Move v_n from P_a to P_c;
  $rv_{new} =$ `RoleBoundViolations`$(\mathcal{P}_{kNN}(P_a))$;
  if $rv_{new} > rv_{old}$ **then**
    Restore $\mathcal{P}_{kNN}(P_a)$;
Update the partition boundaries $\forall P_a \in \mathcal{P}$;
return $\mathcal{G}_{\mathcal{P}}$.

Algorithm 4: `RoleBoundViolations`

Input: $\mathcal{P}, \mathcal{R}$

Output: I_{new}

```
1 I_new = 0;
2 foreach r ∈ R do
3     foreach p ∈ P do
4         I_new = I_new + I_p^r; /* Compute imprecision of overlapping roles
              and partitions.                                              */
5 return I_new.
```

$B^{R_2} = 75$, respectively. Also, in TSH2, highly skewed partitions are rejected, i.e., role cuts are only feasible when the size ratio of the resulting partitions is not highly skewed. We use a skew ratio of 1:99 for TSH2 as a threshold. If a cut results in one partition having size greater than hundred times the other, then the cut is ignored.

Algorithm 2 (TSH2) has four differences compared to TSH1. First, the kd-tree traversal for the **foreach** loop in Lines 2-14 is based on preorder traversal. The preorder traversal ensures that a given partition is recursively split till the leaf nodes are reached. Then, the role bounds are updated. Second, in Line 14, the role bounds are updated as the partitions are being added to $\mathcal{P}$. Third, in Line 5 of Algorithm 2, we use only one role for the candidate cut and fourth in Line 7, the partition size ratio condition is checked to reject skewed partition cuts. If no feasible role cut is found, then the partition is split using the median cut approach as in Line 12.

Lemma 3.4.2 *The time complexity of TSH2 is* $\mathcal{O}(d|\mathcal{R}|n \log n)$.

Proof The depth of the kd-tree for TSH2 is $\log_{\frac{100}{99}} n$. The work performed at each level of the kd-tree is $\mathcal{O}(d|\mathcal{R}|n)$ as we consider only one role for a feasible cut. Then, the time complexity of the first stage is $\mathcal{O}(d|\mathcal{R}|n \lg n)$. As the time complexity of

Algorithm 3 is $\mathcal{O}(d|\mathcal{R}|n)$, the overall time complexity of TSH2 is $\mathcal{O}(d|\mathcal{R}|n \log n + d|\mathcal{R}|n) \approx \mathcal{O}(d|\mathcal{R}|n \log n)$. ■

3.5 Performance and Security Analysis

This section evaluates the proposed framework (Fig. 3.3) for system design performance as well as security analysis perspective. Section 3.5.1 presents performance evaluation for the proposed heuristics in terms of meeting the desired access control and privacy requirements with minimum information loss. Section 3.5.2 provides security analysis of the proposed framework from an attack perspective.

3.5.1 Performance Evaluation

This section presents a comparative assessment of the overall performance evaluation of the proposed heuristics TSH1 and TSH2 in terms of satisfying access control and privacy requirements and incurring minimum information loss.

Experiments have been conducted on a 2.4 GHz Intel Core $i5$ with 8 GB of 1600 MHz DDR3 SDRAM running Mac OS X operating system. All algorithms have been implemented using Java 1.7. We present two different sets of experimental results. In the first set of experimental results 'Number of Role Violations', we study the effect of anonymity parameter k on the number of role bound-violations, which is an access control requirement. In the second set of experimental results 'Information Loss Due to Anonymization', we study the changes in information loss value due to parameter k.

Datasets and RBAC Policy

In the experimental results, we use the following real graph topologies: `ego-Facebook`, `p2p-Gnutella04`, and `com-Youtube` available at Stanford Network Analysis Project

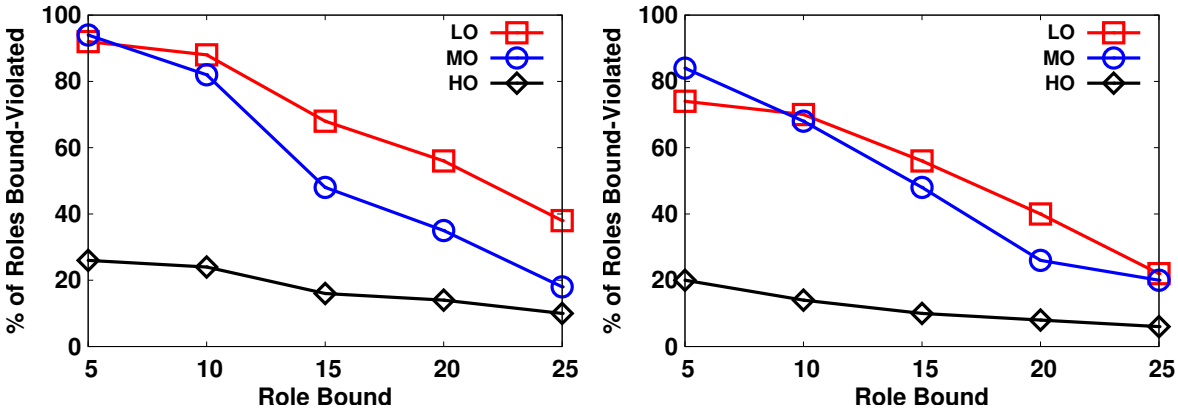
(SNAP)[4].We populate the vertices of these graphs (similar to [40]) with the Census dataset from IPUMS[5]. The dataset is extracted for the Year 2001 using the following attributes: Age, Gender, Marital status, Race, Birth place, Language, Occupation, and Income. The categorical data values have already been converted to numeric values. The first seven attributes are used as the QI attributes and are assigned to graph nodes while Income is considered as a sensitive attribute.

Role Workload Generation

We generate 50, 80, and 500 roles as the workload/permissions for the `ego-Facebook`, `p2p-Gnutella04`, and `com-Youtube` datasets, respectively. The roles are generated according to the approach of [38], which selects two attribute tuples randomly from the attribute tuple space and forms a role by making a bounding box of two tuples. The generated role workload may be overlapped. A highly overlapped workload means more sharing between roles, which signifies less data sensitivity and vice versa. We can further classify this workload into three classes: low-overlap (LO), medium-overlap (MO), and high-overlap (HO) and study the effect of degree of overlap between workloads on the proposed heuristics. If the overlap is between 10-20%, we consider this as LO; if the overlap is between 40-50%, we consider this as MO; and similarly, for an overlap in the range 80-90%, we classify this as HO. The average role size for the 50 roles under LO, MO, and HO is 81, 124, and 145, respectively. Similarly, for 80 roles, the corresponding roles sizes for LO, MO, and HO workload are 153, 201, and 263, respectively.

Number of Role Violations

[4]http://snap.stanford.edu/data/
[5]`https://usa.ipums.org/usa/`



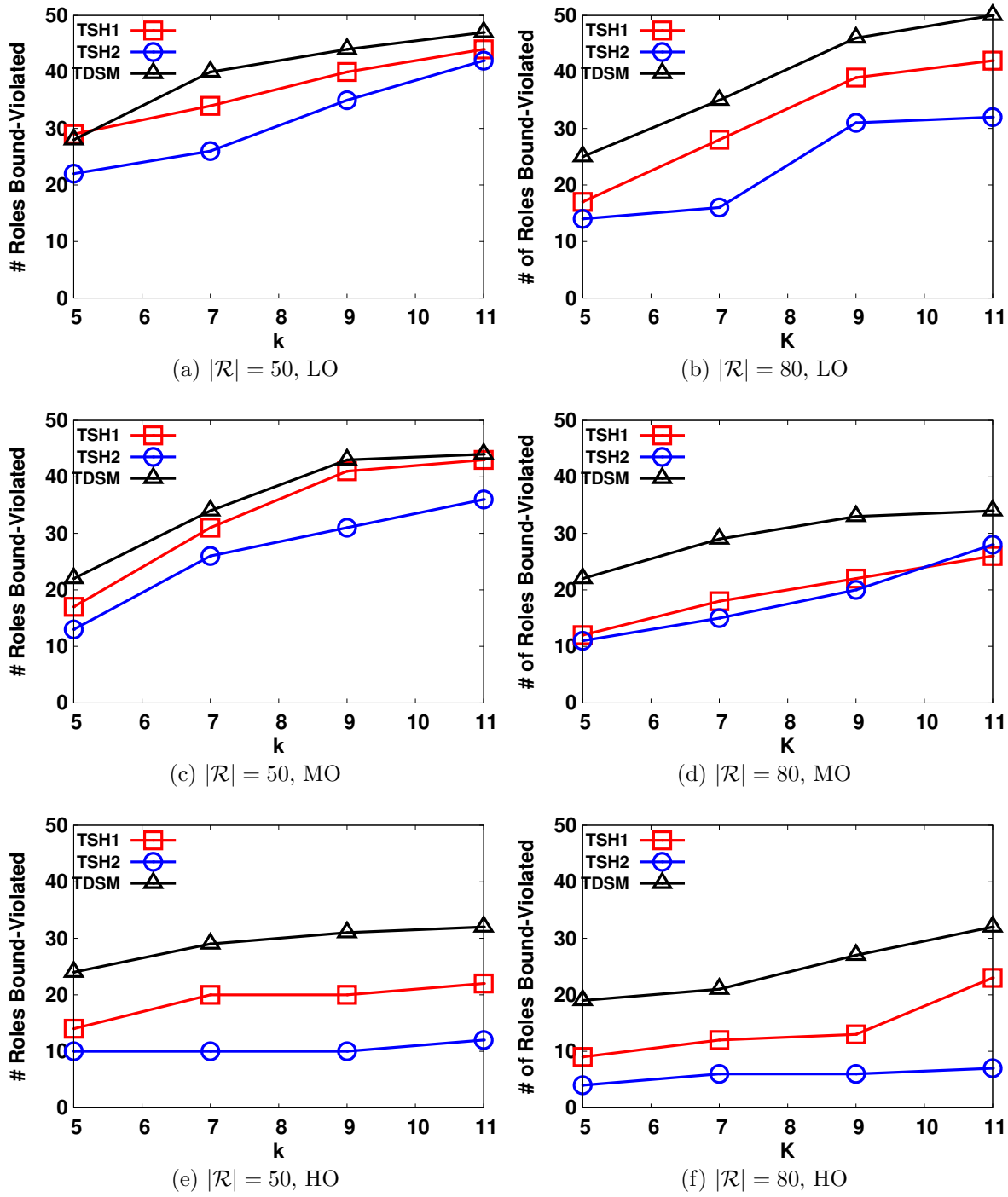
(a) TSH1, $|\mathcal{R}| = 50$ (b) TSH2, $|\mathcal{R}| = 50$

Fig. 3.4.: Effect of B^R on the % of role bound-violations for $k = 5$.

In this subsection, we evaluate the effect of anonymity parameter k on the number of role bound-violations for the two proposed heuristics and compare the results against TDSM [19] algorithm.

The imprecision bounds of all the roles are set based on the role size for the current experiment. Otherwise, the bounds of the roles can be set by the access control administrator. The intuition behind setting bounds as a factor of the role size is that the imprecision added to the role is proportional to the role size [38]. In our experimental results, we set the role imprecision bounds to 20% of the role size. Fig. 3.4 illustrates the effect of the role imprecision bound on the number of role bound-violations for a fixed value of the anonymity parameter k and three different role overlapping workloads: LO, MO, HO. It is quite intuitive that as we increase the role imprecision bound, the number of roles violating their imprecision bound decrease. This occurs because the tolerance value for a role being violated is increased.

From Figs. 3.5 and 3.7, observe that as we increase the value of k, the number of role bound-violations also increase, suggesting that the role violations are dependent on k. This occurs because as we increase the partition size (i.e., the k value



(a) $|\mathcal{R}| = 50$, LO

(b) $|\mathcal{R}| = 80$, LO

(c) $|\mathcal{R}| = 50$, MO

(d) $|\mathcal{R}| = 80$, MO

(e) $|\mathcal{R}| = 50$, HO

(f) $|\mathcal{R}| = 80$, HO

Fig. 3.5.: Effect of k on the # of role bound-violations for $B^R = 20\%$.

is increased), more roles are now overlapping the partitions, resulting in increased imprecision, and hence more role bound-violations. Results in Fig. 3.6 support the fact that as we increase the value of k, the role imprecision standard deviation (SD) also increase, resulting in more role bound-violations.

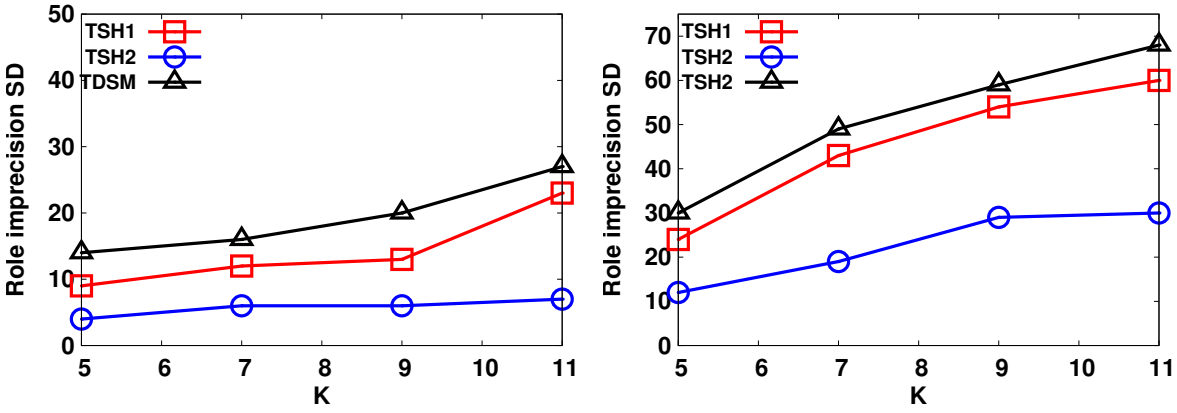
Also in Figs. 3.5 and 3.7, algorithm TSH2 performs consistently better than TSH1 in terms of role bound-violations across all different datasets and role workloads as the value of k is increased. The reason is that in TSH2, the role bounds of the overlapping roles are updated as new partitions are added to the final partition set. This results in generating compact partitions and thus results in fewer role bound-violations as more partitions fall within role boundaries resulting in reduced imprecision. Moreover, both the algorithm TSH1 and TSH2 perform consistently well compared with TDSM across various overlapping role workloads.

Furthermore, observe from Fig. 3.5 that as we increase the role overlap, a fewer number of roles are violated. This can be explained as follows: As we increase the role-overlap, the average role size increases; this means that more partitions fall within the role boundaries, and hence the role imprecision decreases, causing fewer roles to violate their bounds. Secondly, the role bound is based on the cardinality of role size; as we increase the role overlap, the average role size also increases and thus the role bound also increases, causing fewer number of roles to be violated.

Information Loss Due to Anonymization

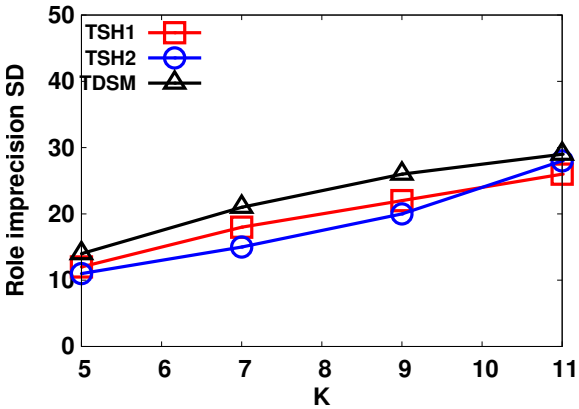
In this section, we study the effect of changing k value on the information loss. From Figs. 3.8, 3.9, and 3.10, we observe that as we increase the value of k, the information loss value also increases for all the different datasets and overlapping role workloads. However, this increase in loss value is non-linear.

Comparing algorithms TSH1 and TSH2, we observe that TSH2 has a higher information loss value when compared to TSH1 for all different datasets and role workloads as shown in Figs. 3.8, 3.9, and 3.10.

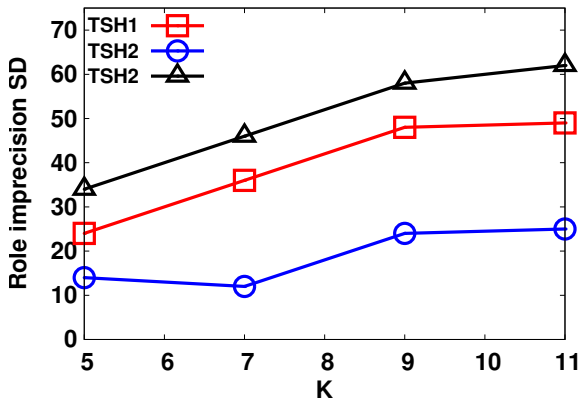


(a) $|\mathcal{R}| = 50$, LO

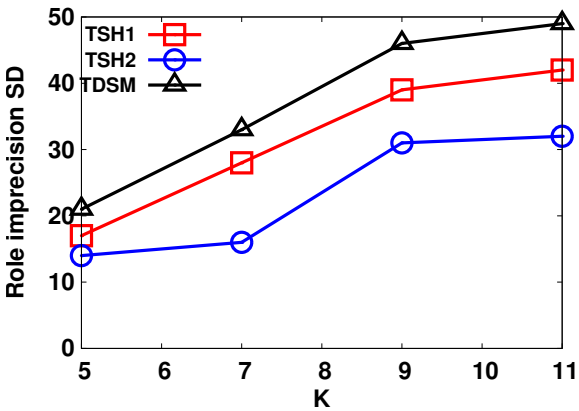
(b) $|\mathcal{R}| = 80$, LO



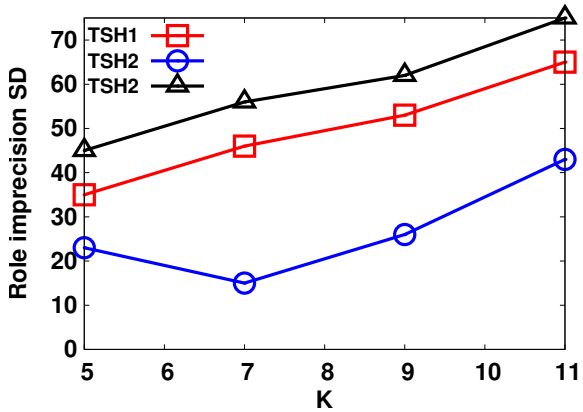
(c) $|\mathcal{R}| = 50$, MO



(d) $|\mathcal{R}| = 80$, MO



(e) $|\mathcal{R}| = 50$, HO



(f) $|\mathcal{R}| = 80$, HO

Fig. 3.6.: Effect of k on role imprecision SD.

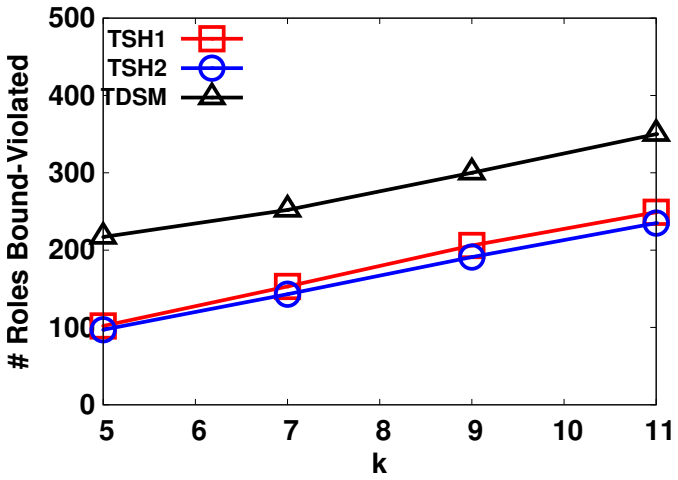
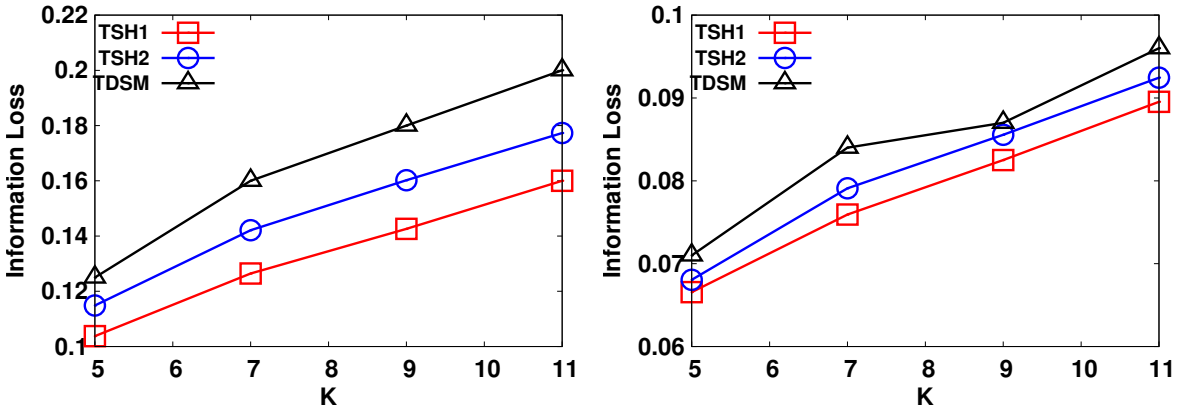


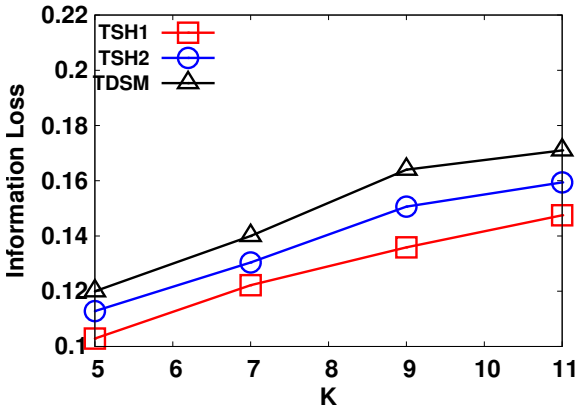
Fig. 3.7.: Effect of k on the # of role bound-violations for $B^R = 20\%$ and $\mathcal{R} = 500$.

In contrast to TSH1 (where we consider all roles overlapping a given candidate partition to find a feasible cut), in TSH2 we consider only one role with the least imprecision bound that is overlapping the candidate partition to find a feasible cut. TSH2 has a lower computational complexity compared to TSH1. Moreover, if the resulting split partitions are skewed, we reject the cut and choose to split the partitions using the median-cut approach. So, we reduce the role search space and have more median-cut-based partitions in TSH2 compared with TSH1. The median-cut approach aims to obtain a uniform occupancy; this technique causes high descriptive information loss when the data is skewed. More specifically, only one single attribute, instead of multiple attributes available in QI set, is used to split a partition, this causes the rest of the attributes in QI set to retain their least specific values, and thus causes a high penalty for those values [41]. Secondly, according to reasoning in [16], a higher structural information loss corresponds to clusters in which nodes have similar connectivity properties with one another. In other words, when the nodes in a cluster are either all connected (highly dense) or disconnected (highly sparse) among themselves and with the nodes in other clusters and vice versa, we can explain the reasoning for a smaller structural information loss.

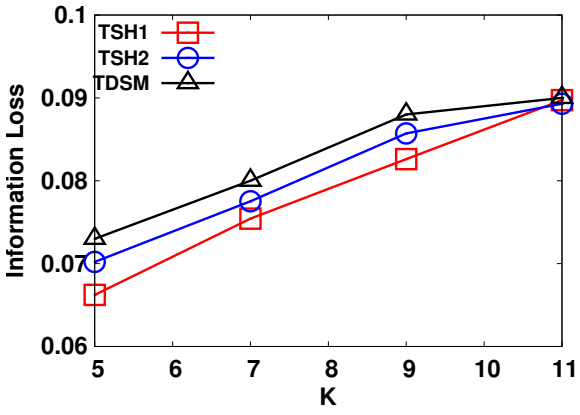


(a) $w = 0.5$, LO

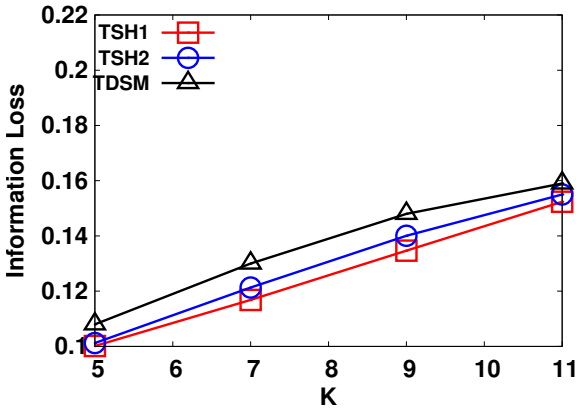
(b) $w = 0.2$, LO



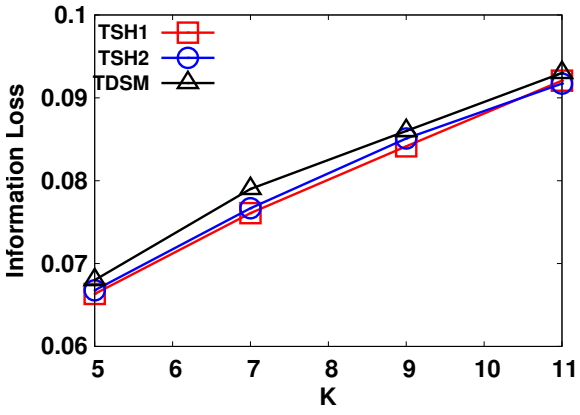
(c) $w = 0.5$, MO



(d) $w = 0.2$, MO



(e) $w = 0.5$, HO



(f) $w = 0.2$, HO

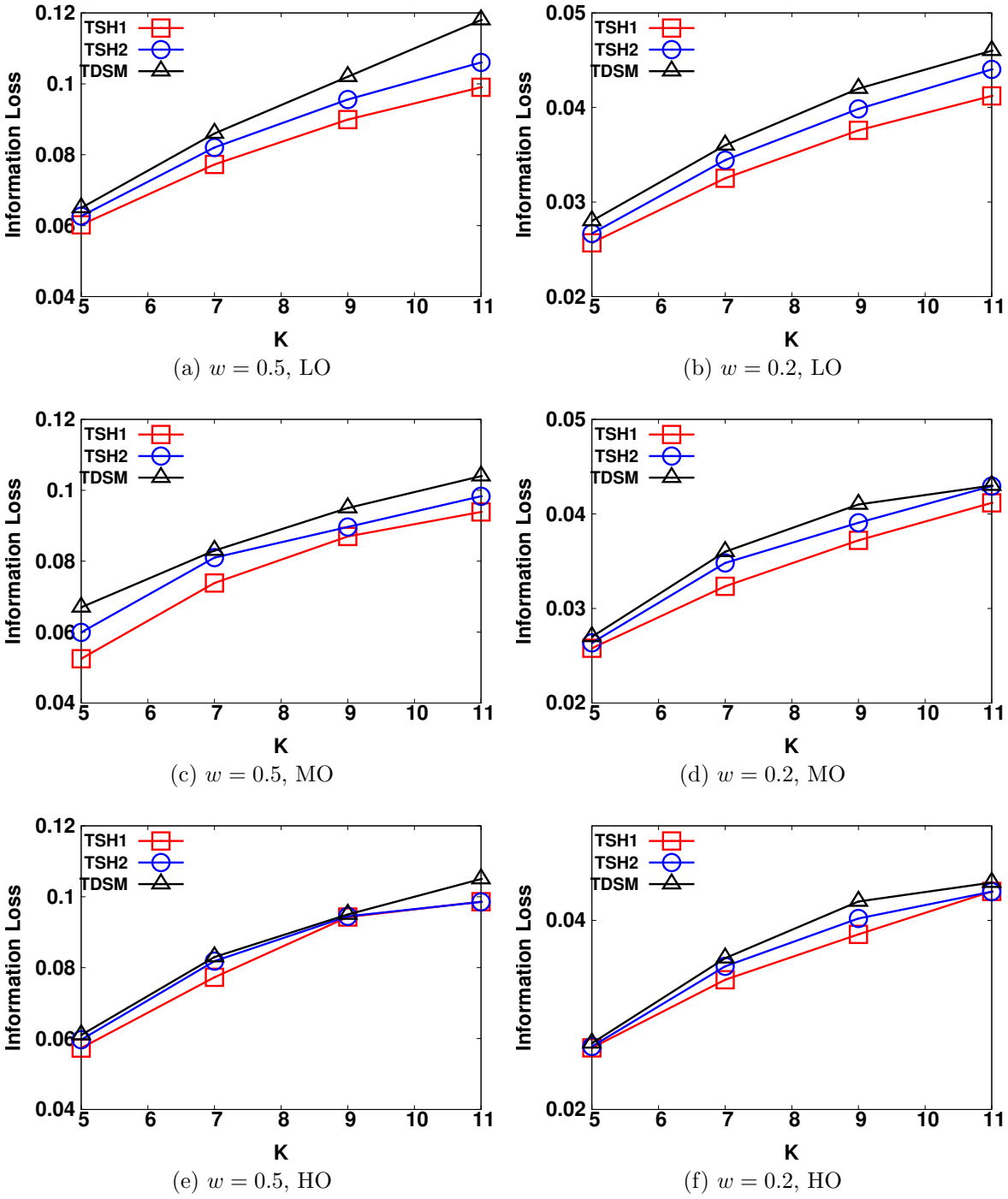
Fig. 3.8.: Effect of k on information loss for `ego-Facebook` and $|\mathcal{R}| = 50$.

Algorithm TSH1 (having the higher computational complexity) has the lower overall information loss value due to anonymization between the two proposed heuristics. TSH1 considers all the roles overlapping a given partition to determine a feasible cut with the least imprecision bound; thus, the technique does not aim to obtain a uniform occupancy, incurring a low information loss, when the data is skewed [41]. Secondly, as explained in [16], a smaller structural information loss corresponds to clusters in which nodes have similar connectivity properties with one another, or, in other words, when cluster nodes are either all connected (highly dense) or disconnected (highly sparse) among them and with the nodes in other clusters.

Furthermore, we observe that with the increase of role overlap, the performance gap between the two proposed heuristics in terms of information loss narrows down.

Therefore, we conclude that TSH2 offers a better performance in terms of satisfying more role bounds while it incurs more information loss. On the other hand, TSH1 performs better in terms of information loss, but it violates more role bounds. Both heuristics, however, perform well compared to TDSM in terms of number of role bound-violations and information loss. The performance gap between the two proposed heuristics, in terms of information loss, is between 5-15% for different workloads with varying degree of overlap. Table 3.2 summarizes the comparison between the proposed heuristics.

In Figure 3.11, we plot the data utility results as we vary the value of parameter k. The results show the data utility plots for both the descriptive data utility U_D and structural data utility U_S. We observe that the data utility value decreases as we increase the value of parameter k. The reason is larger partition sizes result in more information loss. The value of U_S is greater that 0.9 for all different cases. Moreover, the value of U_D is much lower than the value of U_S, which means information loss due to attribute generalization is higher than structure generalization.



(a) $w = 0.5$, LO (b) $w = 0.2$, LO

(c) $w = 0.5$, MO (d) $w = 0.2$, MO

(e) $w = 0.5$, HO (f) $w = 0.2$, HO

Fig. 3.9.: Effect of k on information loss for `P2PNutella` and $|\mathcal{R}| = 80$.

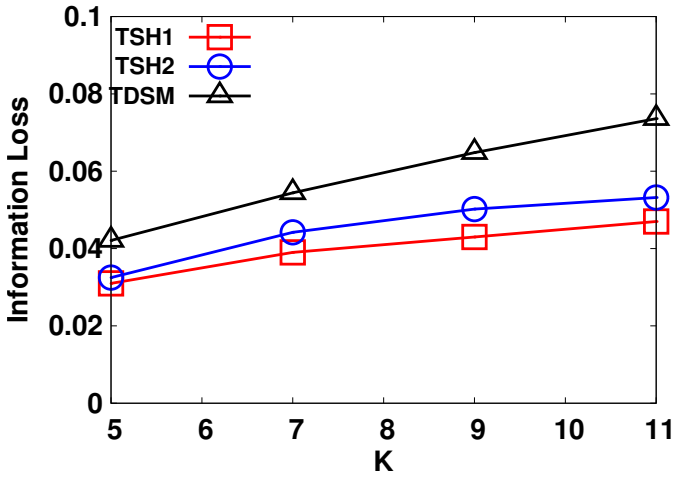
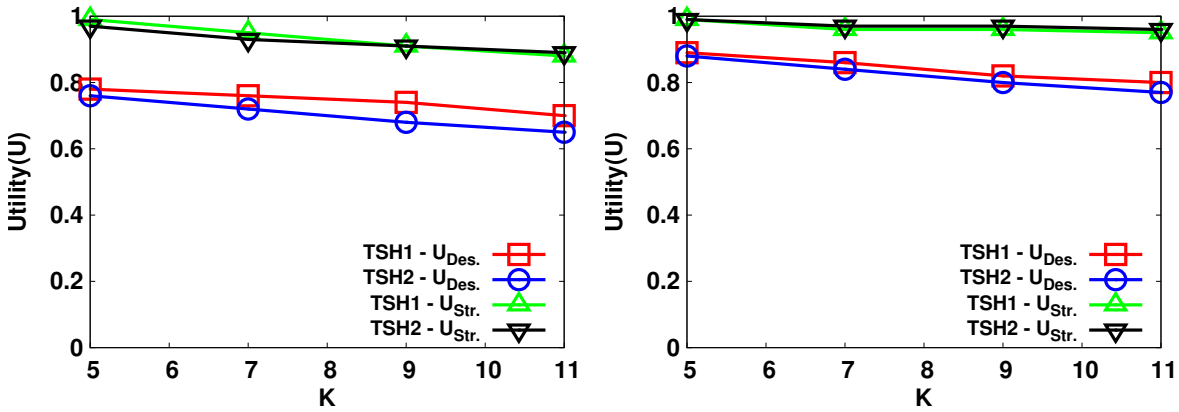


Fig. 3.10.: Effect of k on information loss for `com-Youtube` and $|\mathcal{R}| = 500$.



(a) `ego-Facebook` (b) `p2p-Gnutella04`

Fig. 3.11.: Effect of k on data utility.

Table 3.2.: Comparison of proposed heuristics

Heuristic	Role violation	Complexity	Information Loss
TSH1	High	$\mathcal{O}(d\|\mathcal{R}\|^2 n^2)$	Low
TSH2	Low	$\mathcal{O}(d\|\mathcal{R}\| n \log n)$	High

3.5.2 Security Analysis

This section presents a probabilistic analysis of the framework with respect to re-identification attack. We assume that the adversary $\mathcal{A}$ is assigned a role, say $R_{\mathcal{A}}$. Using some structural and attribute information about the target node x the adversary $\mathcal{A}$ can pose a query to initiate a re-identicaiton attack on graph.

De-anonymization Attack Under an Access Control Policy

Our access control mechanism does not allow an adversary $\mathcal{A}$ with an assigned role, say $R_{\mathcal{A}}$, to access data beyond his authorized privilege set as specified by RBAC policy administrator. Therefore, according to our proposed RBAC mechanism, complete graph de-anonymization is not possible by whatever background knowledge an adversary $\mathcal{A}$ may have. The is because the scope of adversary $\mathcal{A}$'s attack is assumed to be confined to authorized data set.

Theorem 3.5.1 (Re-identification risk) *Assume that the adversary $\mathcal{A}$ has some structural and attribute information as background knowledge for re-identification of a target node x. Given an anonymized graph $\mathcal{G}_{\mathcal{P}} = \langle V_{\mathcal{P}} = \mathcal{P}, E_{\mathcal{P}}, \overline{\mathcal{T}} \rangle$, where each partition $P_i \in \mathcal{P}$ forms a hyper-rectangle in a d-dimensional data space with volume $U_{vol} = \prod_{i=1}^{d}[U_i]$, $U_i = [U_i^{min}, U_i^{max}]$ is the domain range for dimension i, the adversary $\mathcal{A}$'s attribute set $Attr \subseteq \{QI_1, \ldots, QI_d\}$ can be considered as forming a subspace with dimension $|Attr|$ and having volume $\prod_{i=1}^{|Attr|}[L_i] \times \prod_{j=1}^{(d-|Attr|)}[U_j] \subseteq U_{vol}$, $L_i = [L_i^{min}, L_i^{max}]$. The probability of successfully re-identifying a target node x is given as:*

$$Pr(Re\text{-}id(x)) \leq \frac{1}{k} \times \max\left\{ Pr_{Q_A(Q_S)}(y), Pr_{Q_S(Q_A)}(y) \right\}, \tag{3.7}$$

where $Pr_{Q_A(Q_S)}(y)$ is the probability of $y \in cand(x)$ being a feasible candidate partition for x, and is computed by first applying the adversary $\mathcal{A}$'s structural information Q_S and then applying the attribute information Q_A. $Pr_{Q_A(Q_S)}(y) = \frac{1}{|cand_{Q_S}(x)| \times \prod_{j=1}^{|Attr|} \min(1, \frac{\overline{S}_j + L_j}{U_j})}$.

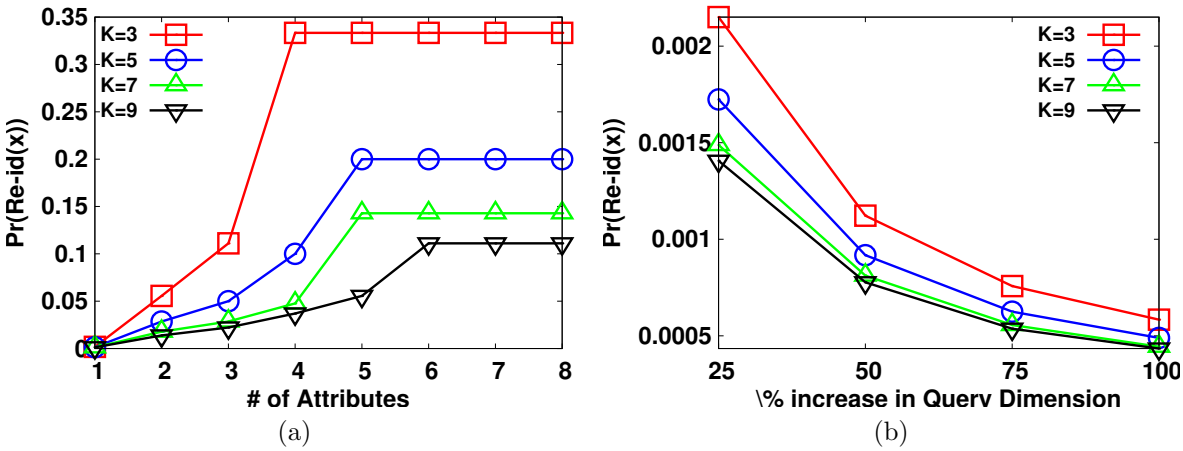


Fig. 3.12.: (a) The effect of $Q_A = \{[0-32], [0-0], [0-2], [0-3], [0-333], [0-32], [0-4], [0-331]\}$ on $Pr(Re\text{-}id(x))$. (b) The effect of % increase in query $Q_A = \{A_1, A_6\}$ dimension on $Pr(Re\text{-}id(x))$.

$|cand_{Q_S}(x)|$ is the number of feasible candidate partitions y in $\mathcal{P}$ returned as part of structural query $Q_S(x)$. $|cand_{Q_S}(x)| \times \prod_{j=1}^{|Attr|} \min(1, \frac{\overline{S}_j + L_j}{U_j})$ is the number of feasible candidate partitions y returned as part of query Q_A executed on $cand_{Q_S}(x)$ set, where $\overline{S}_j$ is the average of projection of all partitions along jth dimension, and L_j and U_j are the corresponding projections for query Q_A and total domain space U_{vol}. Similarly, $Pr_{Q_S(Q_A)}(y)$ is the probability of y computed by first applying the adversary $\mathcal{A}$'s attribute information and then structural information. Note: For those attributes that are not in the adversary $\mathcal{A}$'s background knowledge, their complete domain space is considered as part of $\mathcal{A}$'s background knowledge.

Proof Please refer to Appendix B. ■

Example 3 *We assume the adversary $\mathcal{A}$ has both attribute and structural information as background knowledge. For evaluation purpose, we use the following attribute domain ranges $\{[0-32], [0-0], [0-2], [0-3], [0-333], [0-32], [0-4], [0-331]\}$ from the IPUMA dataset.*

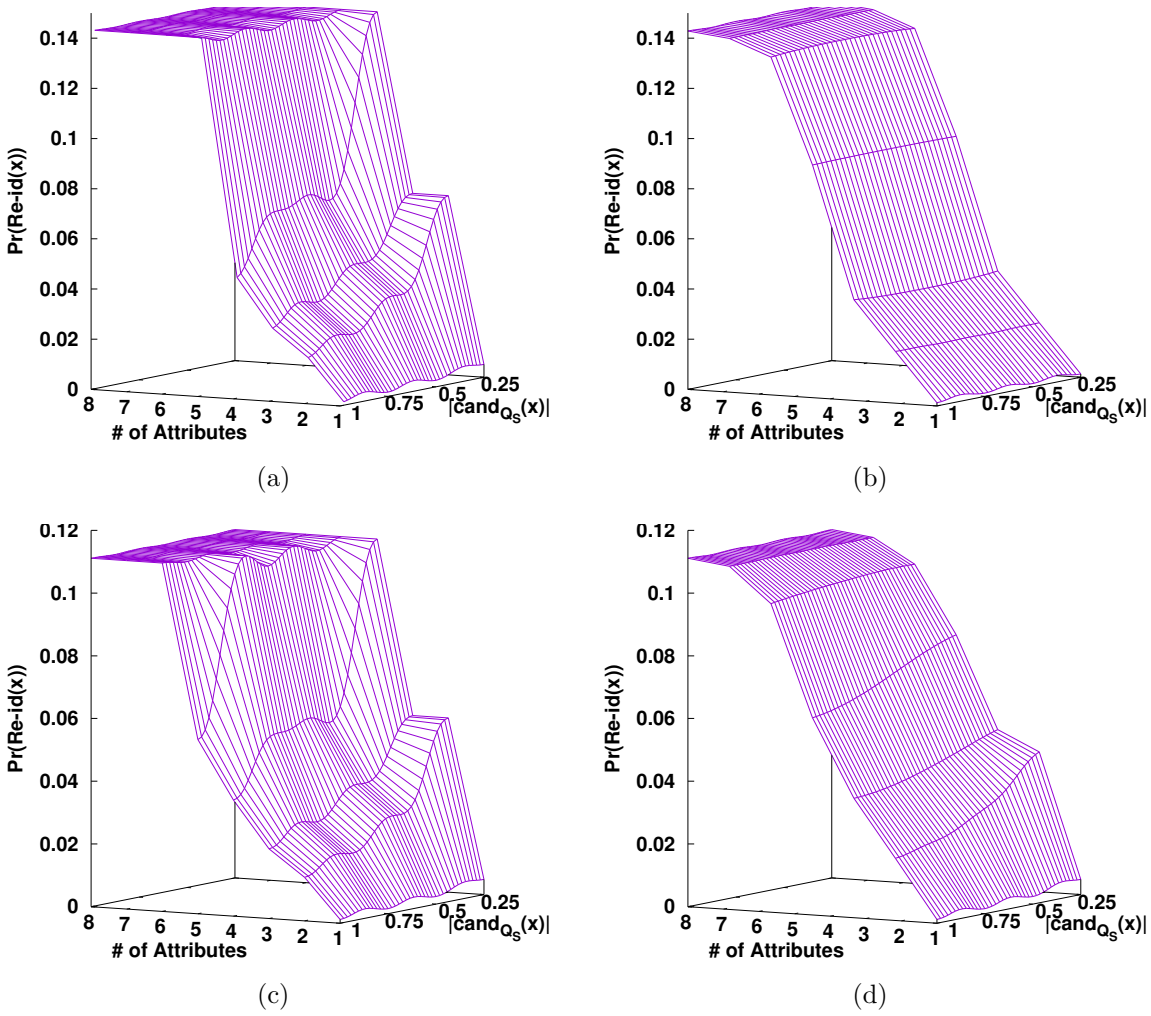


Fig. 3.13.: (a) $Pr(Re\text{-}id(x))$ vs query $Q_A(Q_S)$ for $k = 7$; (b) $Pr(Re\text{-}id(x))$ vs query $Q_S(Q_A)$ for $k = 7$; (c) $Pr(Re\text{-}id(x))$ vs query $Q_A(Q_S)$ for $k = 9$; (d) $Pr(Re\text{-}id(x))$ vs query $Q_S(Q_A)$ for $k = 9$;

We consider two cascaded query scenarios: i) $Q_A(Q_S)$, where the adversary $\mathcal{A}$ first applies the structure based query Q_S and then applies the attribute based query on the Q_A to determine the feasible candidate set $cand(x)$ for a target node x, and ii) $Q_S(Q_A)$, where first the attribute information is used, and then the structural information is applied to determine $cand(x)$ set.

Fig. 3.12(a) shows the effect of changing the number of attributes in query Q_A on re-identification probability. We observe that increasing the number of attributes results in higher $Pr(\text{Re-id}(x))$ value. This is because, more background attribute knowledge helps reduce $|cand_A(x)|$ size, thus resulting in a higher $Pr(\text{Re-id}(x))$ value. Fig. 3.12(b) studies the effect of increasing the range query dimension on the $Pr(\text{Re-id}(x))$ value. The number of attributes in adversary $\mathcal{A}$'s background knowledge are kept fixed to attribute $A_1 = [0 - 32]$ and $A_6 = [0 - 32]$. We observe that $Pr(\text{Re-id}(x))$ value reduces as the volume of range query Q_A increases for a fixed number of attributes. This is because, a large number of candidate partitions are returned resulting in lower $Pr(\text{Re-id}(x))$ value.

Fig. 3.13(a)-(d) show the effect of cascaded queries $Q_A(Q_S)$ and $Q_S(Q_A)$ on re-identification probability $Pr(\text{Re-id}(x))$. Increasing the value of k generates less number of $cand(x)$ partitions resulting in higher $\frac{1}{|cand(x)|}$ value and reduces $\frac{1}{k}$ value. Therefore, the overall value of $Pr(\text{Re-id}(x))$ is reduced. From Fig. 3.13(a), (d) we observe that cascaded query $Q_A(Q_S)$ results in higher $Pr(\text{Re-id}(x))$ value compared to cascaded query $Q_S(Q_A)$ value. The reason being complete graph $\mathcal{G}_{\mathcal{P}}$ has less structural discrepancy compared to graph obtained by first running attributed based query Q_A, thus resulting in better filtration of candidates. Therefore, plots in Figs. 3.13(a) and 3.13(c) give higher value $Pr(\text{Re-id}(x))$ compared to Figs. 3.13(b) and 3.13(d) for query $Q_S(Q_A)$.

However, it is intuitive that having both attribute and structure information improves the re-identification probability as the joint background knowledge results in reduced candidate partition size $cand(x)$ for a target node x.

3.6 Related Work

Although a number of anonymization schemes have been proposed for protecting users' privacy in published graph data (e.g., [42], [17], [43], [44], [45], [46], [47],

[48], [49]), they implicitly assume that there is no *authorization mechanism* in place for controlling access to data by group of users.

Almost all OSN services allow users to dictate sharing their profile information to viewers through fine-grained, customized policies. For example, PERSONA [32] hides private data with *attribute-based encryption* (ABE) schemes. XACCESS [33] presents an *automated* RBAC policy specification mechanism to capture the implicit privacy preference of social site users. Semantically interpretable functional "social roles" are extracted from static network structure based on identified social roles, confidentiality setting of personal data, and predefined user-permission assignments. Yuan et al. [42] introduce a framework which provides privacy-preserving services based on the user's personal privacy requirements. Specifically, the formulation combines the label generalization and the structure modification techniques by adding "noise" edges or nodes in a way that satisfy privacy protection requirements. We consider a data publishing framework where a centralized authority enforces the authorization constraints through RBAC policy while providing data privacy protection through anonymiation.

For the state-of-the art techniques in graph data anonymization and their classification, we refer the reader to recent survey papers [2], [3]. Our graph anonymization technique falls under the category of graph generalization/clustering based technique [16], [17], [47]. Under this scheme, the graph is first partitioned into subgraphs. Then, each subgraph is replaced by a super-node, which may be connected by super-edges. The number of nodes in each super-node, along with the density of edges that exist within and across super-nodes are published. Since the size of each cluster is at least k, the probability of re-identifying a user can be bounded to at most $\frac{1}{k}$. Hay et al. [17] propose an aggregation based graph anonymization algorithm, and study the extent of node re-identification based on structural information using three types of structural queries as an adversary background knowledge on anonymized graphs. Bhagat et al. [47] design a class-based anonymization algorithm, which groups the entities into classes and masks the mapping between entities and the nodes that rep-

resent them in the anonymized graph. Our work is perhaps closest in philosophy to [16], which proposes a greedy optimization solution that can be tuned to control information loss. Their clustering algorithm takes into consideration loss of both node labels and structure information. However, the interplay between access-control and privacy-protection mechanisms has been missing.

Differential Privacy (DP) [48] is another popular privacy protection approach, where noise is added to query results to satisfy privacy constraints. However, DP is viable for privacy-preserving data mining (PPDM) and it is still an open question if it can practically support privacy-preserving data publishing (PPDP) [50].

3.7 Summary

We present a framework for privacy-enhanced access-controlled graph data. The access control policies define the selection predicates available to roles/queries and their associated imprecision bounds. Only authorized role/query predicates on sensitive data are allowed by the access control mechanism. The privacy protection and loss reduction module anonymizes the graph data such that maximum number of roles satisfy their imprecision bounds and minimum information loss is incurred. For this, we formulate a k-anonymous Bi-objective Graph Partitioning (k-BGP) problem and give hardness results. Two heuristics TSH1 and TSH2 are developed to solve the constraint problem. We provide empirical evaluation of the proposed heuristics with a benchmark algorithm [35] from design perspective in terms of meeting privacy and access control requirements with minimum information loss. Within the context of k-BGP problem, we present an architecture framework elaborating how access control and privacy can be integrated. We evaluate the proposed framework from security perspective and present a probabilistic analysis for re-identification risk.

4. EFFICIENT AND SCALABLE INTEGRITY VERIFICATION OF DATA AND QUERY RESULTS FOR GRAPH DATABASES

4.1 Introduction

Graphs are used for representing and understanding objects and their relationships for numerous applications. Recent years have seen the emergence of many large graph datasets. The most well known example is the Web, which now contains more than 50 billion Web pages and more than one trillion unique URLs [1]. Other leading examples include social networks, biological networks, semantic Web, XML documents, and financial databases [51], [52].

To address this challenge, we propose two integrity verification schemes for graph data using message authentication codes (MACs). MACs have been a fundamental functionality for many recent developments in cryptography. These codes are used for establishing an SSL/TLS connection and for ensuring the integrity of shared data among multiple untrusted parties [22], [23]. Traditionally MAC functions handle messages as bit-strings and generate a MAC tag or "cryptographic checksum" to ensure the integrity of input messages. We propose a methodology based on MACs for graph data in order to verify the integrity of graph dataset. In the case of standard MAC-based schemes (e.g., MACs, HashMACs, and HMACs), a message is either shared completely or not shared at all with the user. In contrast, when graphs are used, a user may receive part(s) of a graph in form of a query result(s). A major advantage of using HMACs for graphs and redactable graphs is reduced computational requirements and processing time as compared to the digital signature-based mechanisms [53]. Digital signatures support not only integrity of data but also security

[1] `http://www.worldwidewebsize.com`

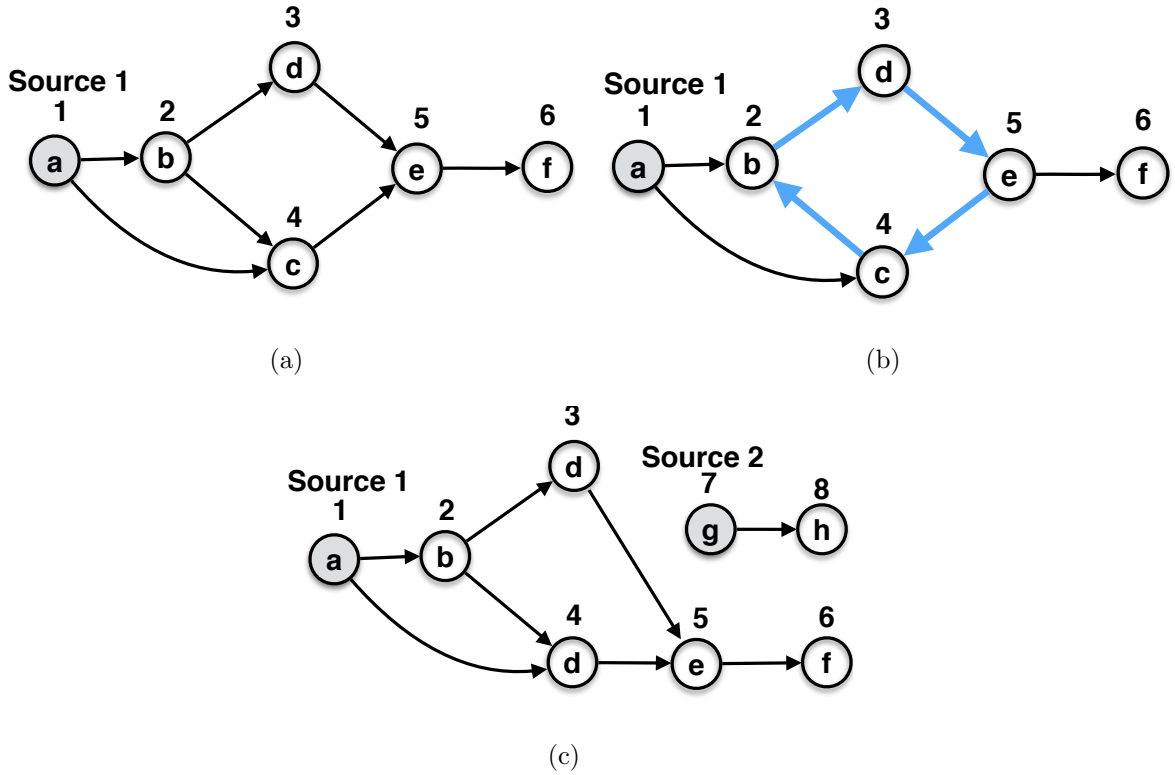


Fig. 4.1.: Graphs: (a) DAG, (b) Graph with cycle $v_2 \rightarrow v_3 \rightarrow v_5 \rightarrow v_4 \rightarrow v_2$, and (c) DAG with multiple sources $\{v_1, v_7\}$.

properties like authentication of data source and non-repudiation [26]. However, in many practical use cases, data protection requirements include only integrity verification of data as other properties of digital signatures incur additional costs. To that end, MACs have been developed for integrity verification. Freeman and Miller show that HMACs are 15-20 times faster than RSA digital signatures [54]. Further, no MAC-based technique has been proposed for graph data and graph query results. In this paper, our focus is primarily on integrity verification of graph data and graph query results using HMACs.

In case, data is stored on untrusted servers, generally integrity verification of graph data is triggered when: i) data is updated, ii) user issues a query, or iii) we need to

verify two graphs are identical[2]. Recently, hashing schemes have been proposed for integrity verification of directed graphs [56]. Hashing is used as a core function for integrity verification for all MAC-based schemes (e.g., HMACs and HashMACs) and hash-and-sign methodology. HMACs are shown to be much times faster than RSA digital signatures [54]. Therefore, in case integrity of graph data is to be assured without losing on efficiency, cost and simplicity of implementation, it is preferable to use schemes which do not employ digital signature-based techniques due to their higher computational and processing cost. Merkle hash technique (MHT) has been proposed as an approach for computing hashes for trees [57] and has been extended for directed acyclic graphs (DAGs) [58]. MACs for trees, DAGs, and cyclic graphs have not been well-studied in the literature. We present two efficient MAC-based schemes for integrity verification of graph data and query results. The schemes can be used by real-world graph database systems. In addition, we formally define MAC-based schemes for graph data, and analyze the security properties (Section 4.5) with respect to tampering of graph data in terms of its structure as well as data attributes.

4.1.1 Contributions

In summary, this chapter makes the following contributions:

- We have developed two schemes for graph data integrity verification
  - HMACs for graphs for two-party data sharing, and
  - Redactable HMACs for graphs for third-party data sharing.
- The proposed schemes can support "fail-stop" and "fail-warn" integrity assurance (Section 4.2.4) mechanisms, which can result in substantial saving in the cost incurred for integrity verification and data re-transfer of compromised graphs.

[2]Identical graphs are isomorphic graphs, but the reverse is not true [55].

- We provide formal definitions and constructions of HMACs for graphs and redactable HMACs for graphs.
- We prove that the proposed schemes are secure and protect the graphs and redacted graphs from being compromised.
- Experimental results on real-world graph datasets show that HMACs for graphs and redactable HMACs graphs are highly efficient compared to digital signature-based schemes for graphs.
- The computational complexity of the proposed schemes is linear in the number of vertices and edges in the graph. We compute one HMAC value and two other verification objects for redaction as part of the query results that are shared with the verifier. Therefore, our scheme is efficient both in processing time and in the transmission of result set $\mathcal{R}$ and verification objects $\mathcal{VO}$ to the client/user.

4.1.2 Organization

The rest of the chapter is organized as follows: Section 4.2 introduces background and desiderata of HMACs for graphs. Section 4.3 and 4.4 introduce the schemes HMAC for graphs (`gHMAC`) and redactable HMAC for graphs (`rgHMAC`), give their formal definitions, and describe the constructions. Section 4.5 gives the security analysis of the schemes. The complexity analysis and performance analysis are presented in Section 4.6. Section 4.7 overviews the related work and Section 4.8 contains concluding remarks.

4.2 Background and Desiderata of HMACs for Graphs

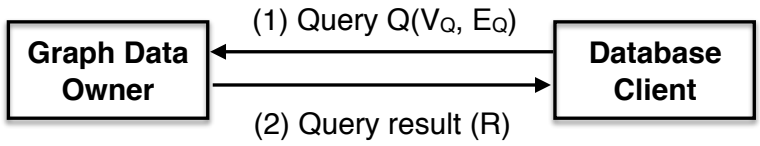
4.2.1 Data Model

We consider graph datasets which are modeled as a directed graph $G(V, E)$, where V is a set of nodes (or vertices) and E is a set of edges between these nodes; $e(x, y) \in E$ is an edge from x to y, $(x, y) \in V \times V$. Undirected graphs can be represented as directed graphs. Therefore, in what follows we consider only the case of directed graphs and we will use the term graph with the meaning of directed graph. A node x represents an atomic unit of data, which is always shared as a whole or is not shared at all. A source is a node that does not have any incoming edge. A node x is called an ancestor of a node y iff there exists a path consisting of one or more edges from x to y. Node x is an immediate ancestor, also called parent, of y in G iff there exists an edge $e(x, y)$ in E. Nodes having a common immediate ancestor are called siblings. Let $G(V, E)$ and $G_\delta(V_\delta, E_\delta)$ be two graphs. We say that $G_\delta(V_\delta, E_\delta)$ is a *redacted subgraph* of $G(V, E)$ if $G_\delta(V_\delta, E_\delta) \subseteq G(V, E)$. $G_\delta(V_\delta, E_\delta) \subseteq G(V, E)$ if and only if $V_\delta \subseteq V$ and $E_\delta \subseteq E$. Also $G_\delta(V_\delta, E_\delta) \subset G(V, E)$ if and only if $V_\delta \cup E_\delta \subset V \cup E$. A redacted subgraph $G_\delta(V_\delta, E_\delta)$ is derived from the graph $G(V, E)$ by redacting the set of nodes $V_\delta = V \setminus V'$ and the set of edges $E_\delta = E \setminus E'$ from G, where $G'(V', E')$ is the subgraph that is not part of the query result $\mathcal{R} = G_\delta(V_\delta, E_\delta)$.

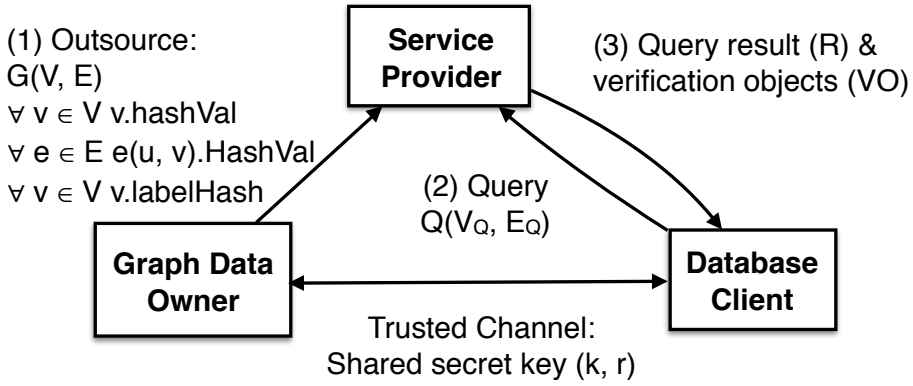
4.2.2 Graph Data Publishing and Query Model

In this paper, we consider two data publishing models: i) two-party data publishing model (Fig. 4.2 (a)), and ii) third-party data publishing model (Fig. 4.2 (b)).

In the two-party data publishing model, there are only client and $\mathcal{DO}$, which communicate with each other directly. In other words, the $\mathcal{DO}$ has also the additional responsibility of being a query front engine for processing the queries and generating the results $\mathcal{R} = G_\delta(V_\delta, E_\delta)$ as well. The third-party data publishing [9], [8], [59] framework, on the other hand, comprises the following three parties:



(a) Two-party data model



(b) Third-party data model

Fig. 4.2.: System model for publishing and querying graph data.

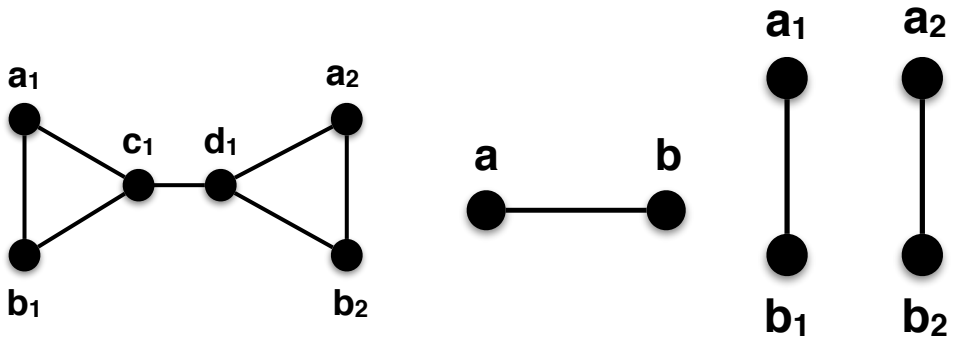
1. Graph data owner ($\mathcal{DO}$): The $\mathcal{DO}$ owns a graph database.

2. Database service provider ($\mathcal{SP}$): The $\mathcal{SP}$ stores the graph database and acts as a query front-end engine. The $\mathcal{SP}$ receives a graph query q from the database client, processes it on behalf of the $\mathcal{DO}$, and returns the graph query answer $\mathcal{R} = G_\delta(V_\delta, E_\delta)$ to the client. Since $\mathcal{SP}$ may not be trusted, it is required to return not only the graph query result, but also verification object $\mathcal{VO}$ and the $\mathcal{DO}$'s tag to the client.

3. Database client: We assume that the client has access to shared secret key (k, r) over a trusted secure channel. The client verifies the *soundness* (all the query result graphs are answers and they are not tempered) and *completeness* (there is no graph that is not in the query result but is an answer) of the query results.

In the third-party data model, the $\mathcal{SP}$ is trusted to "redact" the graph G to G_δ such that $V_\delta \subseteq V$, and $E_\delta \subseteq E$. The vertices and edges that are not included in the query result $\mathcal{R} = G_\delta(V_\delta, E_\delta)$ are the ones that are redacted from G to result in G_δ as part of the query processing. The $\mathcal{SP}$ is authorized to delete certain vertices and edges from G resulting in G_δ, which is then sent to the client as the query result $\mathcal{R} = G_\delta(V_\delta, E_\delta)$. The $\mathcal{SP}$ is not authorized to carry out any other operation(s) on the graph that modifies its structure or content in any manner.

Query Model: Let $Q(V_Q, E_Q, T_Q)$ be a query graph Q, where V_Q is the node set of Q, E_Q is the edge set of Q, and $T_Q = V \rightarrow \sum^*$ be a function that represents the label value for each vertex in V_Q. Our schemes assume a generic query model. As an example, consider *subgraph matching* [59] which is defined as follows: For a data graph G and a query graph Q, the goal of *subgraph matching* is to find every subgraph $g = (V_g, E_g) \in G$ such that there exists a bijection $f : V_Q \rightarrow V_g$ that satisfies $\forall v \in V_Q, T_Q(v) = T_G(f(v))$ and $\forall e \in E_Q, (f(u), f(v)) \in E_g$, where $T_G(f(v))$ represents the label of the vertex $f(v) \in G$.



(a) A graph G (b) A query Q (c) Result $\mathcal{R}_{G,Q}$

Fig. 4.3.: An example of subgraph query matching.

Example 4 *Consider the data graph G and query graph Q as given in Figs. 4.3(a) and 4.3(b), respectively. Repeated labels in the graph G are subscripted with number*

i. The query graph Q matches two instances on data graph G and the output result $\mathcal{R}_{G,Q} = \{(a_1, b_1), (a_2, b_2)\}$ is shown in Fig. 4.3(c).

As part of our scheme user does not receive any duplicate vertices or edges in the result $\mathcal{R}$.

4.2.3 Threat Model

For a two-party data model (Fig. 4.2 (a)), we assume a *trusted* data model in which both the $\mathcal{DO}$ and the client have access to the shared secret key (k, r). The owner of the graph data computes the gHMAC using the shared secret key (k, r), and the client/receiver of the graph data verifies the gHMAC using the key (k, r).

For a third-party data model (Fig. 4.2 (b)), we assume a *semi-trusted* model in which the $\mathcal{SP}$ may not be completely trusted and it does not have access to the gHMAC key k, but it can compute the query results. The $\mathcal{SP}$ may be the potential adversary $\mathcal{A}$ or another adversary $\mathcal{A}$ who may have attacked and hacked the $\mathcal{SP}$. Under this model, we assume that the client and $\mathcal{DO}$ can exchange the shared secret key k over a secure trusted channel.

4.2.4 Desiderata of MACs for Graphs

Challenges: The challenges in computing HMAC of a graph in contrast to computing HMAC of a monolithic message bit-string are as follows:

- The graph is a semi-structured data object and is more complex than a sequential string/chain of bits in a monolithic message. How we ensure integrity of the graph data depends on how we incorporate the edges and vertices in the graph and the order, if any, between the nodes computing the HMACs.

- How can a verifier re-compute the same HMAC tag value as the value computed by the $\mathcal{DO}$? Graphs can be traversed in many ways by different parties. One

needs to ensure that the verifier follows exactly the same traversal order as the HMAC computing entity, $\mathcal{DO}$, follows.

- How to ensure ensure integrity of each vertex and its structural position, i.e., the order between the sibling vertices?
- How to ensure integrity of each edge?
- The HMAC-based computation and verification schemes for graph data should be quite efficient especially in contrast to the digital signature-based schemes, because for monolithic messages, HMAC-based schemes are more efficient than digital signature-based schemes.

Fail-stop HMACs: Integrity assurance of graphs opens up new challenges which are not faced for the integrity assurance of monolithic messages – bit-strings. One of these challenges is fail-stop integrity verification. Since graphs have vertices and edges, the integrity verification process need to be terminated as soon as integrity of a vertex or an edge is found to be compromised; thus saving computational cost (such as cost of query processing in a cloud) and time, resulting in improved performance requirements. Such saving can be substantial for large graph datasets. We can develop HMACs that are fail-stop. Fail-stop HMACs should stop as soon as they determine a compromise.

Fail-warn HMACs: Another variant of this type of integrity assurance is fail-warn integrity verification. The integrity verification proceeds even after finding a compromised vertex or edge, and outputs a set of all such vertices and edges that are corrupted. The verifier/client may then request the data provider only those vertices and edges that are compromised instead of requesting the complete data again. That shall result in more efficient and cost-effective database services, network usage and quality of service.

Redactable HMACs: Given a graph, often part(s) of the graph are sent to a client as part of a query result. It thus may be necessary for the verifier to be able to

verify the integrity of such subgraph(s) using HMAC without the need to have access to the graph itself. It is a challenging problem especially for HMACs because, unlike in digital signatures, the key is symmetric, and thus no part of the key can be shared with the $\mathcal{SP}$, such as a cloud service. An HMAC-based scheme should be redactable in the sense that a verifier can verify the integrity of the redacted graphs, i.e., the subgraphs, with few integrity verification objects $\mathcal{VO}$ from the $\mathcal{SP}$. In other words, the $\mathcal{SP}$ does not have access to the secret key.

4.3 HMACs for Graphs (gHMAC)

4.3.1 Formal Definition

In this section, we provide the formal definition of hash message authentication code for graphs (gHMAC).

Definition 4.3.1 (HMAC for Graph (gHMAC)) *Given a graph $G(V, E)$ with vertex set V and edge set E, let $g\Pi$ be a gHMAC for graph G, and let $\Pi_H = (\mathtt{Gen_H}, \mathtt{H})$ be a hash function with output length ℓ. The scheme $g\Pi$ consists of three polynomial-time algorithms $g\Pi \equiv (\mathtt{gGen}, \mathtt{gHmac}, \mathtt{gVrfy})$ and is defined as follows:*

1. `gGen`*: On input 1^n, the algorithm chooses a uniform $k \in \{0,1\}^n$ and runs $\mathtt{Gen}_H(1^n)$ to obtain a random r to generate the key (k, r).*

2. `gHmac`*: The algorithm takes as input a graph $G(V, E)$ and a key (k, r) and outputs a graph tag t_G value and a source list* `SourceList` *of vertices. The* `SourceList` *is the list of source vertices for traversing different components of the graph, in case the components are disconnected. We write this as*

$$(t_G, \mathtt{SourceList}) \leftarrow \mathtt{gHmac}_{(k,r)}(G(V, E)).$$

3. `gVrfy`*: The algorithm takes as input a graph* $G(V, E)$*, a tag* t_G*, a key* (k, r)*, and a* `SourceList`*. The algorithm outputs a bit* b*, with* $b = 1$ *meaning* `valid`*, and* $b = 0$ *meaning* `invalid`*. We write this as*

$$b \leftarrow \texttt{gVrfy}_{(k,r)}(t_G, G(V, E), \texttt{SourceList}).$$

4.3.2 HMAC Scheme for Graphs

Overview of Our Scheme

HMAC scheme, `gHMAC`, takes a graph $G(V, E)$ and a shared secret key (k, r) as its input. The key (k, r) that has two parts – k which is the secret key used in HMACs and r which is a random bit-string ($\geq$ 128 bits). The `gHMAC` algorithm outputs an HMAC value referred to as tag t_G for the graph $G(V, E)$ and a list of source nodes `SourceList` generated during the graph traversal. The scheme traverses the graph in DFS order and computes the graph hash *ghash* value after visiting each node. During the traversal, it computes an integrity identifier, referred to as `xor-out`(u), for each node $u \in G$. The identifier is used to carry out "local integrity assurance" for that node. Also during the graph traversal a list u.`outList`() is maintained to track the order in which siblings of a node, say u, are visited. The scheme is highly efficient, as it performs a single graph traversal to compute tag t_G for graph G.

Detailed Description

The Algorithm 5 (`gHMAC`) initially marks all the vertices of G as unvisited and then employs a DFS traversal to recursively visit all the unvisited vertices of the graph. While visiting a particular vertex, say u, during a DFS traversal, the algorithm computes `out-xor`(u) value, which is exclusive-or $\oplus$ of the label hash values `labelHash` of all the descendant nodes of vertex u, and recursively calls the DFS algorithm. Once a node is finally visited, the algorithm computes the *ghash* value by taking a hash of

the concatenation of the the previous *ghash* value, a random value (r), and the vertex hash value `hashVal`. After the graph traversal is complete, the algorithm computes the HMAC value for G (Line 12) using the *ghash* value as input, where `opad` and `iPad` are parts of HMAC computation, as explained in Section **??**.

The graph verification scheme Algorithm 6 (`gVrfy`) computes a new tag t'_G value on user's behalf and compares it with the received tag t_G value for integrity verification. The graph verification algorithm `gVrfy` takes a graph $G(V, E)$, a tag t_G value for G, as well as a shared secret key (k, r), as input. The algorithm outputs a boolean 1, if verification passes, or 0, otherwise. Algorithm 6 is almost identical to Algorithm 5, except that it computes a new tag t'_G value and compares it against the received tag t_G value for verification purpose.

4.3.3 Illustration of How gHMAC Works

Example 5 *Graph HMAC computation*

Consider the graph G in Fig. 4.4(a), which represents a DAG. Each vertex contains a label, which is an alphabet and the `SourceList` $= \{v_1\}$. *The* `hashVal` *computation for a vertex, say u, requires first computing the integrity verifier* `out-xor`(u) *for that vertex. For example, consider Fig. 4.4(b) where vertex v_2 has two descendant vertices v_3 and v_4. The* `out-xor`(v_2) *value is computed as $H(c) \oplus H(d)$, which is the exclusive-or $\oplus$ of* `labelHash` *values of its descendants, i.e., vertices v_3 and v_4. The u.*`outList`() *for each vertex u contains the ordering information among the siblings of a parent vertex, say u, visited during the traversal. In this case, v_2.*`outList`$() = \{v_3, v_4\}$, *which shows that vertex v_3 is visited before the vertex v_4. For the above example, the vertex hash value* `hashVal` *and graph hash value ghash are computed in the following order of vertices $v_6 \rightarrow v_5 \rightarrow v_4 \rightarrow v_3 \rightarrow v_2 \rightarrow v_1$, which is G's post-order traversal.*

Algorithm 5: `gHMAC` (Compute graph HMAC for two-party data model.)

Input: $G(V, E)$, (k, r)

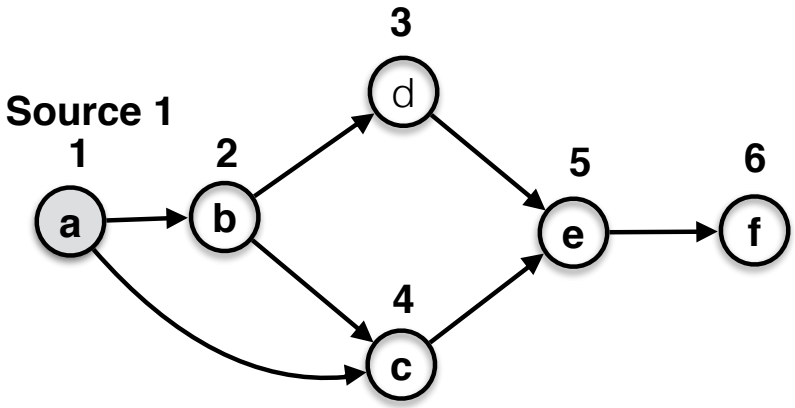
Output: t_G, `SourceList`

```
1  ghash = NULL.
2  SourceList = NULL.
3  foreach vertex u ∈ G.V do
4      u.color = WHITE.
5      u.outList = EMPTY.
6      u.labelHash = NULL.
7      u.hashVal = NULL.
8  foreach vertex u ∈ G.V do
9      if u.color == WHITE then
10         SourceList.add(u).
11         ghash = DFS-VISIT(G, u) .
12 t_G ← H((k ⊕ opad)||H((k ⊕ ipad)||r||ghash)).
13 return (t_G, SourceList).
14 DFS-VISIT(G, u)
15 begin
16     u.color = GRAY.
17     out-xor(u) = u.labelHash = H(u.label).
18     foreach v ∈ descendants(u) do
19         u.outList().add(v).
20         if v.labelHash == NULL then
21             v.labelHash = H(v.label).
22         out-xor(u) = out-xor(u) ⊕ v.labelHash.
23         if v.color == WHITE then
24             DFS-VISIT(G, v).
25     u.color = BLACK.
26     u.hashVal = H(r||out-xor(u)||u.label||u.content).
27     ghash = H(ghash||r||u.hashVal).
28     return ghash.
```

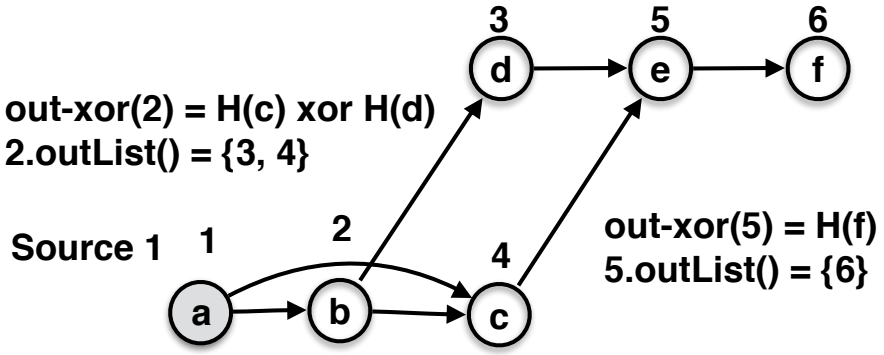
Algorithm 6: `gVrfy` (Verify graph integrity using HMAC tag for two-party data model.)

Input: $G(V, E)$, (k, r), t_G, `SourceList`
Output: Output: 0/1

```
1  ghash = NULL.
2  foreach vertex u ∈ G.V do
3      u.color = WHITE.
4      u.outList = EMPTY.
5      u.labelHash = NULL.
6      u.hashVal = NULL.
7  while SourceList ≠ EMPTY do
8      u = SourceList.getNext().
9      if u.color == WHITE then
10         ghash = DFS-VISIT(G, u).
11 t'_G ← H((k ⊕ opad)||H((k ⊕ ipad)||r||ghash)).
12 if t_G == t'_G then
13     return 1.
14 else
15     return 0.
16 DFS-VISIT(G, u)
17 begin
18     u.color = GRAY.
19     out-xor(u) = u.labelHash = H(u.label).
20     foreach v = u.outList().get() do
21         if v.labelHash == NULL then
22             v.labelHash = H(v.label).
23         out-xor(u) = out-xor(u) ⊕ v.labelHash.
24         if v.color == WHITE then
25             DFS-VISIT(G', v).
26     u.color = BLACK.
27     u.hashVal = H(r||out-xor(u)||u.label||u.content).
28     ghash = H(ghash||r||u.hashVal).
29     return ghash.
```



(a)



(b)

Fig. 4.4.: A DAG with source node id = 1, (b) Computation of `out-xor`(u), and u.`outList`() of vertices.

4.4 Redactable HMACs for Graphs (rgHMAC) and Query Processing

4.4.1 Formal Definition

In this section, we provide a formal definition for hash message authentication code for a redacted graph and query model described below.

Definition 4.4.1 (Redactable HMAC for graph) *Given a graph $G(V, E)$ with vertex set V and edge set E, a redactable HMAC scheme for graph $rg\Pi$ consists of four polynomial-time algorithms $rg\Pi \equiv$* (`rgGen`, `rgHmac`, `gRedact`, `rgVrfy`)*:*

1. `rgGen`: *On input* 1^n, *this algorithm chooses a uniform* $k \in \{0,1\}^n$ *and runs* $\texttt{Gen}_H(1^n)$ *to obtain a random* r *to generate the key is* (k, r).

2. `rgHmac`: *The tag-generation algorithm takes a graph* $G(V, E)$ *and the key* (k, r) *as its input, and outputs a pair* $(t_G, \texttt{SourceList})$, *where* t_G *is a tag for graph and* `SourceList` *is the list of source vertices for traversing different components of the graph, in case the components are disconnected. Since the algorithm may be randomized, we write this input/output behavior as*

$$(t_G, \texttt{SourceList}) \leftarrow \texttt{rgHmac}_{(k,r)}(G(V, E)).$$

3. `gRedact`: *The redaction algorithm takes a graph* $G(V, E)$, *set of nodes* $V'_\delta \subseteq V$, *set of edges* $E'_\delta \subseteq E$, *source list* `SourceList` *of vertices, a set of vertex hash values* $\mathcal{VH} = \{u.\texttt{hashVal} | v \in G.V\}$, *and a set of edge hash values* $\mathcal{EH} = \{e(u, v).\texttt{hashVal} | e(u, v) \in G.E\}$ *as its input. The algorithm outputs a redacted graph* $G_\delta(V_\delta, E_\delta)$[3], *where* $G_\delta(V_\delta, E_\delta)$ *is derived from* $G(V, E)$ *consisting of vertices in* $V_\delta = V \setminus V'_\delta$ *and* $E_\delta = E \setminus E'_\delta$, *a source list of vertices* $\texttt{SourceList}_\delta$, *and two verification objects: vo-g and vo-out. We represent the algorithm to generate the required query output as follow:*

$$(\underbrace{G_\delta(V_\delta, E_\delta)}_{\mathcal{R}}, \texttt{SourceList}_\delta, \underbrace{\textit{vo-g}, \textit{vo-out}}_{\mathcal{VO}}) \leftarrow$$
$$\texttt{gRedact}(G(V, E), \texttt{SourceList}, V'_\delta, E'_\delta, \mathcal{HV}, \mathcal{HE}).$$

4. `rgVrfy`: *The deterministic verification algorithm takes as input the query result* $G_\delta(V_\delta, E_\delta)$, *a tag* t_{G_δ}, *a key* (k, r), *a source list* $\texttt{SourceList}_\delta$ *of vertices, and two verification objects: vo-g and vo-out. The algorithm outputs a bit* b, *with*

[3]Note: As mentioned earlier $\mathcal{R} = G_\delta(V_\delta, E_\delta)$ is the result of user query. The query result also includes the required $\mathcal{VO}$.

$b = 1$ *meaning the result is* `valid`*, and* $b = 0$ *meaning the result is* `invalid`*. We write this validation of query result as*

$$b \leftarrow \texttt{rgVrfy}_{(k,r)}(t_{G_\delta}, G_\delta(V_\delta, E_\delta), \textit{vo-g}, \textit{vo-out}).$$

4.4.2 Redactable HMAC Scheme for Graphs

Overview of Our Scheme

In this section, we provide the construction for `rgHMAC` scheme. The third-party $\mathcal{SP}$ follows the redaction model discussed earlier in Section 4.2.4. As the third-party does not have access to the shared secret key (k, r), the challenge is to develop a scheme for generating a tag t_G value for redacted graph G_δ. In particular, we generate two different hash values: $e(u, v).\texttt{hashVal}$ for edges and $u.\texttt{hashVal}$ for vertices. Expensive operations for computing *ghash* value are not desirable. Therefore, the exclusive-or $\oplus$ operator, which is both commutative and associative can be utilized in combination with vertex and edge hash values. The exclusive-or $\oplus$ operator is purely used for confidentiality purpose and, in this case, as part of our scheme, it acts as a mechanism for integrity preservation.

Detailed Description

The proposed HMAC scheme `rgHMAC` (Algorithm 7) takes as input a graph $G(V, E)$, a shared secret key that has two parts (k, r) – k is the secret key used in HMACs and r is a random bit-string (≥ 128 bits). The algorithm outputs an rgHMAC value referred to as tag t_G for G, and a list of source nodes `SourceList` generated during the graph traversal. Initially, all the vertices are marked as unvisited. The algorithm carries out a DFS traversal on the unvisited vertices to compute the vertex hash values $v.\texttt{hashVal}$ for all $v \in V$ and the edge hash values $e(u, v).\texttt{hashVal}$ for all $e \in E$. In addition, during traversal, a list $u.\texttt{outList}$ is maintained to track the order in which

Algorithm 7: `rgHMAC` (Compute graph HMAC for third-party data model.)

Input: $G(V, E)$, (k, r)

Output: t_G, `SourceList`

```
1  ghash = r.
2  foreach vertex u ∈ G.V do
3      u.color = WHITE.
4      u.outList = EMPTY.
5      u.hashVal = NULL.
6  foreach vertex u ∈ G.V do
7      if u.color == WHITE then
8          SourceList.add(u).
9          ghash = DFS-VISIT(G, u).
10 t_G ← H((k ⊕ opad)||H((k ⊕ ipad)||r||ghash)).
11 return (t_G, SourceList).
12 DFS-VISIT(G, u)
13 begin
14     u.color = GRAY.
15     out-xor(u) = 1.
16     foreach v ∈ descendants(u) do
17         u.outList().add(v).
18         e(u, v).hashVal = H(r||u.label||v.label).
19         out-xor(u) = out-xor(u) ⊕ e(u, v).hashVal.
20         if v.color == WHITE then
21             DFS-VISIT(G, v).
22     u.color = BLACK.
23     u.hashVal = H(r||out-xor(u)||u.label||u.content).
24     ghash = ghash ⊕ u.hashVal.
25     return ghash.
```

Algorithm 8: `gRedact` (Generate query result $\mathcal{R} = G_\delta(V_\delta, E_\delta)$ and verification object $\mathcal{VO}$ for third-party data model.)

Input: $G(V, E)$, $Q(V_Q, E_Q)$, `SourceList`, $\mathcal{VH}$, $\mathcal{EH}$
Output: $\mathcal{R} = G_\delta(V_\delta, E_\delta)$, $\texttt{SourceList}_\delta$, $\mathcal{VO} = \{vo\text{-}g, vo\text{-}out\}$

```
1  vo-g = NULL.
2  vo-out = EMPTY.
3  SourceList_δ = EMPTY.
4  foreach vertex u ∈ G.V do
5      u.color = WHITE.
6      u.outList = EMPTY.
7      u.hashVal = NULL.
8  Compute R = G_δ(V_δ, E_δ) /* Query processing result.                    */
9  while SourceList ≠ EMPTY do
10     u = SourceList.getNext().
11     if u ∈ V_δ then
12         SourceList_δ.add(u).
13         DFS-VISIT(G, u, true).
14     else
15         DFS-VISIT(G, u, false).
16 DFS(G, u, flag)
17 begin
18     if u.color ≠ WHITE then
19         return.
20     u.color = GREY.
21     if flag == false and u ∈ V_δ then
22         SourceList_δ.add(u).
23         flag = true.
24     if u ∈ V_δ then
25         vo-out(u) = 1.
26         foreach v = u.outList.getNext() do
27             if (v ∈ V_δ) and (e(u, v) ∈ E_δ) then
28                 u.outList_δ.add(v).
29             else
30                 vo-out(u) = vo-out(u) ⊕ e(u, v).hashVal.
31         vo-out.add(vo-out(u)). /* Compute VO.                           */
32         if v.color == WHITE then
33             DFS-VISIT(G, v, flag).
34     else
35         if vo-g == NULL then
36             vo-g = u.hashVal.
37         else
38             vo-g = vo-g ⊕ u.hashVal. /* Compute VO.                     */
39     u.color = BLACK.
40     return (G_δ(V_δ, E_δ), SourceList_δ, vo-g, vo-out).
```

Algorithm 9: `rgVrfy` (Verify integrity of query result $\mathcal{R} = G_\delta(V_\delta, E_\delta)$ for third-party data model.)

Input: $\mathcal{R} = G_\delta(V_\delta, E_\delta)$, (k, r), $\texttt{SourceList}_\delta$, $\mathcal{VO} = \{vo\text{-}g, vo\text{-}out\}$, t_G
Output: Output: 0/1

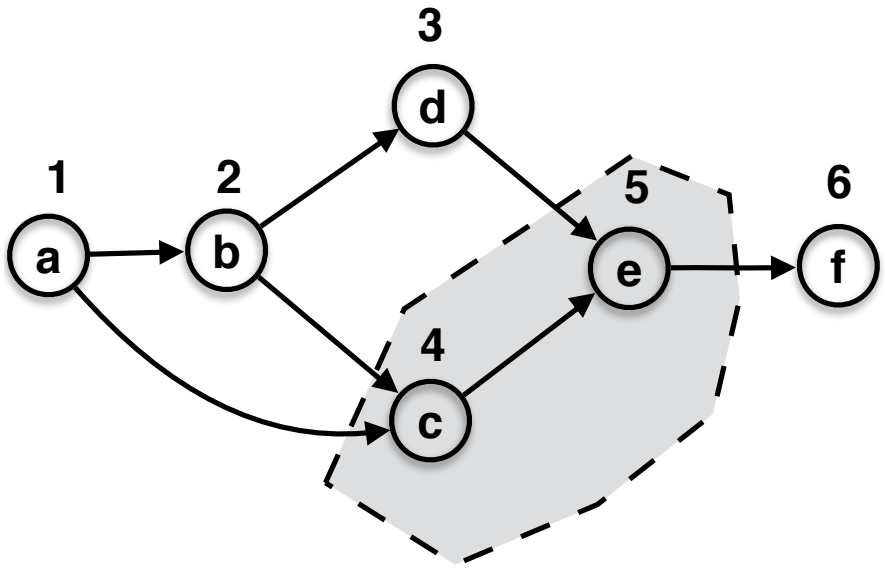
```
1  ghash = r.
2  foreach vertex u ∈ V_δ do
3      u.color = WHITE.
4      u.outList = EMPTY.
5      u.hashVal = NULL.
6  while SourceList_δ ≠ EMPTY do
7      u = SourceList_δ.getNext().
8      if u.color == WHITE then
9          ghash = ghash ⊕ DFS(G_δ, u).
10 ghash = ghash ⊕ vo-g.
11 t'_G ← H((k ⊕ opad)||H((k ⊕ ipad)||r||ghash)).
12 if t_G == t'_G then
13     return 1.
14 else
15     return 0.
16 DFS(G, u)
17 begin
18     u.color = GREY.
19     out-xor(u) = vo-out(u).
20     foreach v = u.outList_δ.getNext() do
21         e(u, v).hashVal = H(r||u.label||v.label).
22         out-xor(u) = out-xor(u) ⊕ e(u, v).hashVal.
23         if v.color == WHITE then
24             DFS-VISIT(G, v).
25     u.color = BLACK.
26     u.hashVal = H(r||out-xor(u)||u.label||u.content).
27     ghash = ghash ⊕ u.hashVal.
28     return ghash.
```

the siblings of a node, say u, are visited. The *ghash* value is computed recursively by taking the exclusive-or $\oplus$ of u.`hashVal` value with the previously computed *ghash* value.

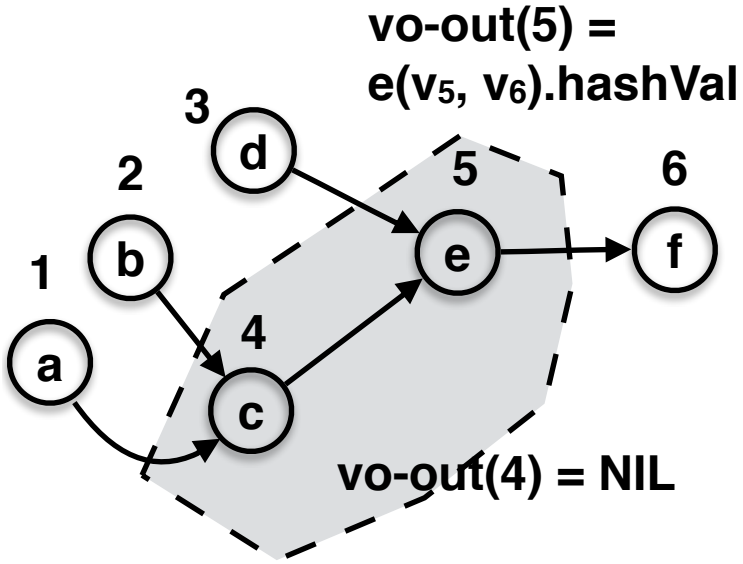
The algorithm `gRedact` (Algorithm 8) is executed by the third-party data distributor, which does not have access to shared secret key k, which poses a challenge to generate a *ghash* value. An approach is to use the $\oplus$ operator in combination with the `hashVal`. Algorithm 8 takes as input a graph $G(V, E)$, a vertex set V'_δ, an edge set E'_δ, a set of vertex hash values $\mathcal{VH} = \{\texttt{v.hashVal} | v \in G.V\}$ and a set of edge hash values $\mathcal{EH} = \{e(u, v).\texttt{hashVal} | e \in G.E\}$. The algorithm outputs a redacted graph $G_\delta(V_\delta, E_\delta)$, a list of source nodes $\texttt{SourceList}_\delta$, a verification object *vo-g* for graph , and the set of verification objects *vo-out*. The algorithm carries out a DFS traversal to compute the verification objects *vo-g* and *vo-out* during the graph traversal. During the graph traversal, for each vertex $u \in V_\delta$ a verification object `vo-out`(u) is computed, which contains the $\oplus$ of label hash values of those ancestors of u that are not present in the set V_δ. Similarly, for those vertices which are in $u \notin V_\delta$ the algorithm computes the verification object *vo-g*, by performing exclusive-or $\oplus$ operation on the vertex hash value `hashVal` of those vertices which are not in the set V_δ.

The algorithm `rgVrfy` (Algorithm 9) performs the graph verification. It takes as input a graph $G_\delta(V_\delta, E_\delta)$, a tag t_G, a shared secret key k, a random number r, a verification object *vo-g* for graph $G(V, E)$, and the sets of verification object *vo-out*. The algorithm outputs a boolean 1, in case of verification process success, or 0, otherwise. The main idea of the verification algorithm `rgVrfy` is to compute the value for $ghash_\delta$ for the redacted graph $G_\delta(V_\delta, E_\delta)$ first and then an exclusive-or $\oplus$ with verification object *vo-g* value to compute the *ghash* value of the complete graph G. The algorithm computes a new tag t'_G using this *ghash* value and compares it against t_G for integrity verification. The algorithm performs a DFS traversal on the graph G_δ, which is received as part of the query result, and computes the integrity verifier `out-xor`(u) for each vertex $v \in V_\delta$. Notice `out-xor`(u) is initialized with

`vo-out`(v), which contains $\oplus$ of label hash `labelHash` values of those descendant vertices of u, which are not present in V_δ.



(a)



(b)

Fig. 4.5.: (a) A DAG where the shadowed part with dotted boundary is the subgraph G_δ that the user receives as part of query result. (b) Verification object set *vo-out* computation for vertices in V_δ.

4.4.3 Illustration of How rgHMAC Works

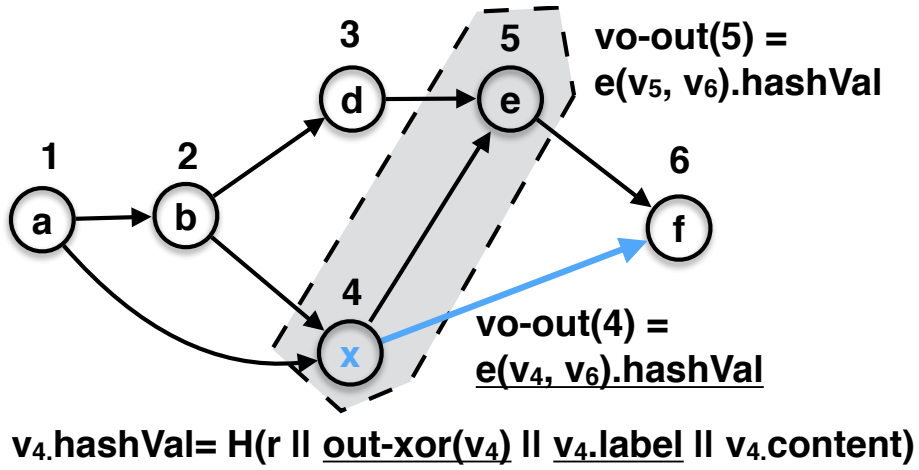
Example 6 (Generation of $\mathcal{VO}$ for query result $\mathcal{R} = G_\delta(V_\delta, E_\delta)$) *Consider the graph in Fig. 4.5(a), which is a DAG. Each vertex is labeled with an alphabet. Suppose the shaded part of the graph with dotted boundary is the subgraph $G_\delta(V_\delta, E_\delta)$, which the user receives as part of the query result $\mathcal{R}$. The algorithm* `gRedact` *determines the sets $V_\delta = \{v_4, v_5\}$ and $V \setminus V_\delta = \{v_1, v_2, v_3, v_6\}$ as part of the query processing step. The user receives the graph G_δ and two sets of verification objects, which are vo-g, and vo-out, as part of query result. For all the vertices that are not part of the query result $\mathcal{R}$ (i.e., $v \notin V_\delta$), the verification object vo-g computes exclusive-or $\oplus$ of the vertex hash values* `hashVal` *in a post-order fashion, vo-g = v_6.`hashVal` $\oplus$ v_3.`hashVal` $\oplus$ v_2.`hashVal` $\oplus$ v_1.`hashVal`. On the other hand, for the set of vertices that are in $V_\delta = \{v_4, v_5\}$, the algorithm computes* `vo-out`(u)*, which considers only those descendants of a vertex, say $u \in V_\delta$, which are not not present in V_δ. For example, from Fig. 4.5(b), we can notice that vertex v_4 has no descendants outside the set V_δ, while vertex v_5 has only one descendant vertex v_6 outside the set V_δ. Therefore, the verification object* `vo-out`$(v_4) =$ `NULL` *as this vertex has no descendant and* `vo-out`$(v_6) = e(v_5, v_6)$.`hashVal`.

4.4.4 Fail-stop and Fail-warn `gHMAC` and `rgHMAC`

One of the key advantages of the proposed HMAC-based schemes is that the HMAC verification process correctly identifies if a node or edge has been compromised. In particular,if a node is compromised with respect to its label or content, the node can be identified by computing the hash value of the label and content of the node along with the r. Similarly, if an edge $e(x, y)$ is tampered, the `out-xor`(x) will be incorrect, resulting in an incorrect `hashVal` computed for x. Furthermore, the end vertices of the edge are considered to be compromised.

In case of tampering, the following steps can be taken:

- **Fail-stop**: Stop the verification processes soon as the first compromise is found.



(a)

Fig. 4.6.: Fail-stop / Fail-warn integrity verification example.

- **Fail-warn**: Continue the verification process, however, warn the system or the user that a compromise has been detected. The process then maintains a list such compromised vertices and edges.

One of the actions of fail-stop or fail-warn system is to request the $\mathcal{DO}$ or $\mathcal{SP}$ to send those vertices and edges that have been compromised to the end user. Such requests can be in real-time, in batch, or after the completion of at the end of integrity assurance process. There is no need to request the complete data unlike in monolithic messages. Therefore, by avoiding the tampering of data, the proposed HMAC-based schemes can result in substantial cost saving time for re-sending data.

Example 7 (Fail-stop / Fail-warn scheme) *Consider the example given in Fig. 4.6, where the label of vertex v_4 is changed from 'c' to 'x' and a new edge $e(v_4, v_6)$ is added. As explained in Example 6 for the set of vertices in $\mathcal{R} = V_\delta = \{v_4, v_5\}$ the verification object* $\texttt{vo-out}(v_5)$ *is computed as* $e(v_5, v_6).\texttt{hashVal}$*; whereas the verification object* $\texttt{vo-out}(v_4)$ *for the new graph G' is computed as* $\texttt{vo-out}(v_4)$ *as* $e(v_4, v_6).\texttt{hashVal}$ *value. However, as the value* $e(v_4, v_6).\texttt{hashVal}$ *is not present with $\mathcal{SP}$, it can be determined that the edge $e(v_4, v_6)$ has been added externally and is not a part of*

the original graph. For the set of vertices $V \setminus V_\delta = \{v_1, v_2, v_3, v_6\}$ *the verification object vo-g is computed as exclusive-or* $\oplus$ *of the vertex hash values* `hashVal` *in post-order fashion, vo-g* $= v_6.\texttt{hashVal} \oplus v_3.\texttt{hashVal} \oplus v_2.\texttt{hashVal} \oplus v_1.\texttt{hashVal}$*. During the vo-g computation vertex* v_2*'s hash value* `hashVal` *is computed as* $v_2.\texttt{hashVal} = H(r||\texttt{out-xor}(v_2)||v_2.\texttt{label}||v_2.\texttt{content})$*, where* $\texttt{out-xor}(v_2) = e(v_2, v_3).\texttt{hashVal} \oplus e(v_2, v_4).\texttt{hashVal}$*.* $e(v_2, v_4).\texttt{hashVal}$ *will not evaluate to a correct value as* v_4 *has a tampered label 'x' resulting in an incorrect value for* $v_4.\texttt{hashVal}$ *as determined by* $\mathcal{SP}$*. Therefore,* $\mathcal{SP}$ *can continue the execution of algorithm and keep track of values that have been modified or stop the execution.*

4.5 Security Analysis

In this section, we review the security of gHMAC and rgHMAC schemes. We present two lemmas with their proofs.

4.5.1 Security of gHMAC

Lemma 4.5.1 *Let the hash function* H *be an implementation of a random oracle, and gh refer to the gHMAC of graph* $G(V, E)$*. Under the random oracle model, and under the assumption that the secret key is known only to the* $\mathcal{DO}$ *and the client, the gHMAC scheme is secure, i.e., a probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *cannot compute* gh' *as the gHMAC of* $G'(V', E')$ *such that* $gh = gh'$*.*

Proof Suppose the adversary $\mathcal{A}$ can compute gh' for a different graph G' such that $gh = gh'$. Consider the case that adversary $\mathcal{A}$ has modified G in an unauthorized manner that resulted in G'. Following unauthorized modifications of the graph may be carried out by adversary $\mathcal{A}$:

- **Label modification**: Suppose the label of a vertex x $x.\texttt{label}$ is modified to $x.\texttt{label}'$; then $gh = gh'$ if and only if (1) and (2) are true: (1) $\texttt{out-xor}(x) = \texttt{out-xor}(x)'$, which is true, iff $H(x.\texttt{label}) = H(x.\texttt{label}')$, and (2) $x.\texttt{hashVal} =$

$x.\texttt{hashVal}'$, which is true, iff $H(r||\texttt{out-xor}(x)||x.\texttt{label}||x.\texttt{content}) = H(r||\texttt{out-xor}(x)||x.\texttt{label}'||x.\texttt{content})$. In order for (1) and (2) to be true, adversary $\mathcal{A}$ has to carry out pre-image attacks on the hash function H, which, however, contradicts our assumption that H is a random oracle.

- **Content modification**: The argument follows the reasoning for label modification above. Suppose the content of a vertex x $x.\texttt{content}$ is modified to $x.\texttt{content}'$; then $gh = gh'$ if and only if the following is true: $x.\texttt{hashVal} = x.\texttt{hashVal}'$, which is true, iff $H(r||\texttt{out-xor}(x)||x.\texttt{label}||x.\texttt{content})$ $= H(r||\texttt{out-xor}(x)||x.\texttt{label}||x.\texttt{content}')$. In order for it to be true, adversary $\mathcal{A}$ has to carry out pre-image attacks on the hash function H, which, however, contradicts our assumption that H is a random oracle.

- **Vertex insertion**: Consider that a vertex z is added to the graph $G(V, E)$ by adversary $\mathcal{A}$ in order to result in a compromised graph G' where $z \in V'$ and $gh = gh'$. This implies that at the end of traversal of vertex z, the computed $ghash'$ value (that does include $z.\texttt{hashVal}$) is identical to that of $ghash$, which does not include $z.\texttt{hashVal}$ as per Algorithm 5 line 27. It is possible, however, if and only if, adversary $\mathcal{A}$ has been successful in carrying out pre-image attacks on the hash function H, which, however, contradicts our assumption that H is a random oracle.

- **Vertex deletion**: Consider that a vertex z is deleted from the graph $G(V, E)$ by adversary $\mathcal{A}$ in order to result in a compromised graph G' where $z \in V \wedge z \notin V'$ and $gh = gh'$. This implies that at the end of traversal of graph G', the computed $ghash'$ value (that does not include $z.\texttt{hashVal}$) is identical to that of $ghash$, which includes $z.\texttt{hashVal}$ as per Algorithm 5 line 27. It is possible, however, if and only if, adversary $\mathcal{A}$ has been successful in carrying out pre-image attacks on the hash function H, which, however, contradicts our assumption that H is a random oracle.

- **Edge insertion**: An edge $e(x, y)$ is added to the graph $G(V, E)$ by adversary $\mathcal{A}$ in order to result in a compromised graph G' where $e(x, y) \notin G$ and x, y are in both G and G'. If either of x, y are vertices inserted by adversary $\mathcal{A}$ in G' and are not in G, then it can be addressed as per the arguments for vertex insertion above. However, $\texttt{out-xor}(x) \neq \texttt{out-xor}(x)'$ because $\texttt{out-xor}(x)$ contains $y.\texttt{labelHash}'$. Thus, (1) $x.\texttt{hashVal} \neq x.\texttt{hashVal}'$, and (2) $ghash \neq ghash'$. (1) and (2) are true if and only if adversary $\mathcal{A}$ has been successful in carrying out pre-image attacks on the hash function H, which, however, contradicts our assumption that H is a random oracle.

- **Edge deletion**: An edge $e(x, y)$ is deleted by adversary $\mathcal{A}$ from G such that $e(x, y) \notin G'$ and x, y are in both G and G'. If either of x, y are vertices inserted by adversary $\mathcal{A}$ in G' and are not in G, then this can be addressed as per the arguments for vertex insertion above. However, $\texttt{out-xor}(x) \neq \texttt{out-xor}(x)'$ because $\texttt{out-xor}(x)'$ does not include $y.\texttt{labelHash}$. Thus, (1) $x.\texttt{hashVal} \neq x.\texttt{hashVal}'$, and (2) $ghash' \neq ghash$ of G. (1) and (2) are true if and only if adversary $\mathcal{A}$ has been successful in carrying out pre-image attacks on the hash function H, which, however, contradicts our assumption that H is a random oracle.

- **Edge modification**: An edge $e(x, y) \in G$ is modified by adversary $\mathcal{A}$ to $e(x, z) \in G'$ such that z is a node in G and G' and there is no edge from x to z in G. However, $\texttt{out-xor}(x) \neq \texttt{out-xor}(x)'$, because $\texttt{out-xor}(x)'$ does not contain $y.\texttt{labelHash}$, but contains $z.\texttt{labelHash}$. Thus, (1) $x.\texttt{hashVal} \neq x.\texttt{hashVal}'$, (2) $z.\texttt{hashVal} \neq z.\texttt{hashVal}'$, (3) $y.\texttt{hashVal} \neq y.\texttt{hashVal}'$, and (4) $ghash \neq ghash'$. (1), (2), (3), and (4) are true if and only if adversary $\mathcal{A}$ has been successful in carrying out pre-image attacks on the hash function H, which, however, contradicts our assumption that H is a random oracle. Any edge modification, such as $e(x, y)$ is modified to $e(y, x)$, can also be reasoned as

above. If more than one edges and vertices are modified, then the arguments above can be extrapolated to the ones above. Thus, the proof follows.

■

4.5.2 Security of rgHMAC

Lemma 4.5.2 *Let the hash function H be an implementation of a random oracle and gh refer to the* `rgHMAC` *of graph $G(V, E)$. Under the random oracle model, and under the assumption that the secret key is known only to the $\mathcal{DO}$ and the client, the* `rgHMAC` *scheme is secure, i.e., a PPT adversary $\mathcal{A}$ (assumption is adversary $\mathcal{A}$ is not the third-party data publisher that can redact G to G_δ) cannot compute gh' as the* `rgHMAC` *of $G'(V', E')$ such that $gh = gh'$.*

Proof For PPT adversary that is not in the third-party data publisher $\mathcal{SP}$, the proofs follow from Lemma 1.For PPT adversary that is the third-party data publisher $\mathcal{SP}$, the proofs are as follows. Suppose the adversary $\mathcal{A}$ can compute gh' for a different graph G' such that $gh = gh'$. Consider the case that adversary $\mathcal{A}$ modified G in an unauthorized manner that resulted in G'. The following unauthorized modification of the graph may be carried out by adversary $\mathcal{A}$:

- **Insertion of vertices**: One or more vertices $w_1, w_2, \ldots, w_m$ may be inserted to G_δ resulting in the updated graph G'_δ, where $w_i \notin V$, $1 \leq i \leq m$. Consider one vertex w_1 being added by adversary $\mathcal{A}$ in G_δ resulting in G'_δ. Adversary $\mathcal{A}$ updates all other values as needed. *vo-g'* is computed by the adversary $\mathcal{A}$ as *vo-g* $\oplus$ w_1.`hashVal`, so that computation of *ghash* (Line 10 of Algorithm 9) by the client/verifier cancels out the w_1.`hashVal` computed and $\oplus$-ed with *ghash* during the DFS traversal (Line 27 of the Algorithm 9). Note that w_1.`hashVal` $\oplus$ *vo-g'* results in *vo-g* by the property of exclusive-or $\oplus$ operator (i.e., $r_1 \oplus r_2 \oplus r_2 = r_1$). However, in order to be able to compute w_1.`hashVal`, the adversary $\mathcal{A}$ either (1) knows random r, which is part of secret key (k, r), or (2) can

assign w_1.`label` and/or w_1.`content` such that w_1.`hashVal` as computed by the adversary $\mathcal{A}$ matches with the value as computed by the verifier (Line 27 in Algorithm 9). (1) contradicts our assumption that the secret key is known only to the $\mathcal{DO}$ and client/verifier, and (2) contradicts our assumption that H is a random oracle. Without loss of generality, the argument can be extended to insertion of multiple vertices $w_1, w_2, \ldots, w_m$.

- **Label, content, or edge modification**: Label, content, or edges of G_δ are modified by adversary $\mathcal{A}$ that results in G'_δ. However, random r (part of secret key (k, r)), which is known only by the $\mathcal{DO}$ and client, is used for computing `hashVal` of the vertices and edges. Therefore, if adversary $\mathcal{A}$ can compute *vo-g'*, *vo-out'* so that G'_δ can be verified against gh, then either adversary $\mathcal{A}$ is aware of r or adversary $\mathcal{A}$ has been able to find a pre-image attack on H. The former contradicts our assumption that (k, r) is secret and known only to the $\mathcal{DO}$ and client and is not available to $\mathcal{SP}$ including any other entities. The latter contradicts our assumption that H is a random oracle, and thus proves the point that label, contents, or edges cannot be modified by adversary $\mathcal{A}$ including the semi-honest service provider $\mathcal{SP}$.

- **Deletion of vertices and edges**: Lemma 4.5.1 applies to deletion of vertices and edges by adversary $\mathcal{A}$ other than the $\mathcal{SP}$ that is authorized to carry out such operations[4]. Thus the proof follows.

■

Table 4.1.: Different graph datasets

Graph	**|V|**	**|E|**	**Description**
email-Enron	265,214	418,956	Email communication network from Enron
web-NotreDame	325,729	1,469,679	Web graph of Notre Dame
amazon0601	403,394	3,387,388	Amazon product co-purchasing network from June 1, 2003
web-Google	875,713	5,105,039	Web graph from Google
wiki-Talk	2,394,385	5,021,410	Wikipedia talk (communication) network

4.6 Performance Analysis

4.6.1 Complexity Analysis

The complexity of the HMAC scheme for graph datasets (Section 4.3) is linear with request to number of nodes and edges in the input graph dataset for both `gHMAC` (Algorithm 5) and `gVrfy` (Algorithm 6). Similarly, for redactable HMAC scheme for graph datasets (Section 4.4), the complexity is also linear with request to number of nodes and edges in the input graph dataset for `rgHMAC` (Algorithm 7) and `rgHMAC` (Algorithm 8).

Algorithms $1, 2, 3, 4$ perform a DFS traversal on the input graph, and at each visit to a vertex carry out a constant amount of computation. Thus, their complexity is $\mathcal{O}(|V| + |E|)$, where $|V|$ is the number of vertices, and $|E|$ is the number of edges for a graph $G(V, E)$. The complexity of query result verification `rgVrfy` (Algorithm 9) is $\mathcal{O}(|V_\delta| + |E_\delta|)$. The algorithm performs a DFS traversal on the redacted graph $G_\delta(V_\delta, E_\delta)$, and at each visit to a vertex, it performs a constant amount of computation.

In essence, these schemes are optimal in terms of their complexity; they are linear in the number of vertices and edges in the graph they process. It is apparent that a sub-linear algorithm cannot process the graph in less than $\mathcal{O}(|V| + |E|)$ and cannot

[4]Under the redaction model as defined earlier in the paper, $\mathcal{SP}$ is authorized to carry out only deletion of vertices and edges, and thus if the $\mathcal{SP}$ is an adversary $\mathcal{A}$, then such deletion operations on vertices and edges amount to be authorized operations.

carry out integrity assurance of all the vertices and edges in a graph. Thus, a $\mathcal{O}(|V|+|E|)$ complexity is the best complexity that an integrity assurance scheme for graphs can achieve.

4.6.2 Performance Evaluation

In this section, we present detailed evaluation of results for the performance of the proposed schemes. We provide two sets of results for two-party data publishing model: i) computation of graph HMAC tag (`gHMAC`), and ii) verification of graph integrity using HMAC tag (`gVrfy`) for graph HMAC-based scheme; and three different sets of results for third-party data publishing model: i) computation of graph HMAC tag (`rgHMAC`), ii) generation of query result $\mathcal{R} = G_\delta$ and verification object $\mathcal{VO}$ (`gRedact`), and iii) verifying the integrity of graph query result (`rgVrfy`) for redacted graph HMAC-based scheme.

Experimental Setup

Running Platform: We have conducted all our experiments on Intel(R) Xeon(R) CPU E5-2620 v3 2.40 GHz machine with 24 Cores and 188G byte RAM. All algorithms have been implemented using Java 1.7. We used JGraphT-0.9.1 (a free Java Graph Library)[5] API as graph processing library. We provided JVM with the following parameters: `−Xms4G` for min heap size and `−Xmx8G` for max heap size. Therefore, we do not use all the available RAM.

Datasets: We have used real-world datasets available at Stanford Large Network Dataset Collection[6] for our experiments. Table 4.1 shows the specification of datasets. The largest graph, wiki-Talk, has about 2M vertices and 5M edges.

Workload Generation: For our experiments, we use `JGraphT` library `Subgraph` class to generate subgraphs $G'(V', E')$ of varying sizes and determine the result set

[5] `http://jgrapht.org`
[6] `http://snap.stanford.edu`

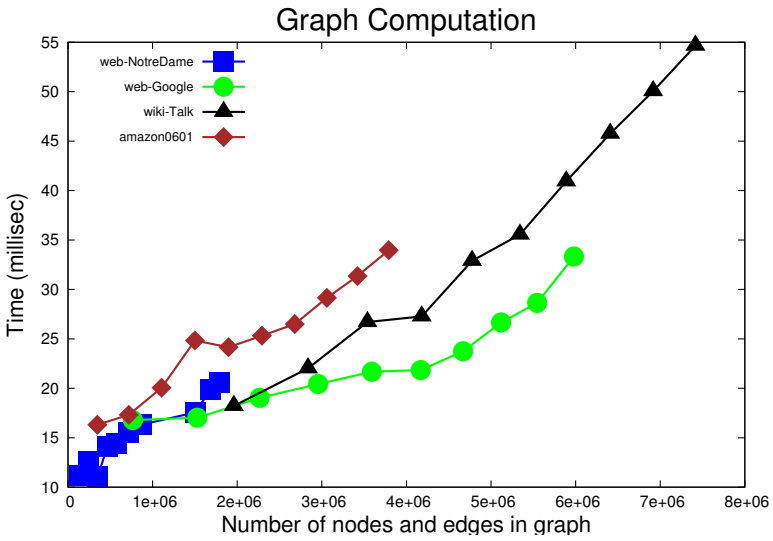


Fig. 4.7.: `gHMAC`: Time to compute graph HMAC tag.

$\mathcal{R} = G_\delta(V_\delta, E_\delta) = G(V, E) \setminus G'(V', E')$. The method `public Subgraph(G base, Set<V> vertexSubset, Set<E> edgeSubset)` generates an induced subgraph given a set of vertices and edges as input from a base graph represented as G.

Results for `gHMAC`

Computation time: Fig. 4.7 shows the time to compute HMAC tag as a function of number of nodes and edges in the input graph dataset. We observe that the computation time increases as the size of input graph dataset increase. The computation time involves determining the "local integrity verifier" `xor-out`(u) for all the vertices and computing the *ghash* value recursively (as a vertex is finally visited during a graph traversal) by concatenating the vertex hash value `hashVal` with the previously computed *ghash* value.

Verification time: Fig. 4.8 shows the time to verify HMAC tag as a function of number of nodes and edges in the input graph dataset. We observe that the verification time increases as we increase the size of input graph. Similar to the computation phase, the graph verification algorithms involves computing the HMAC

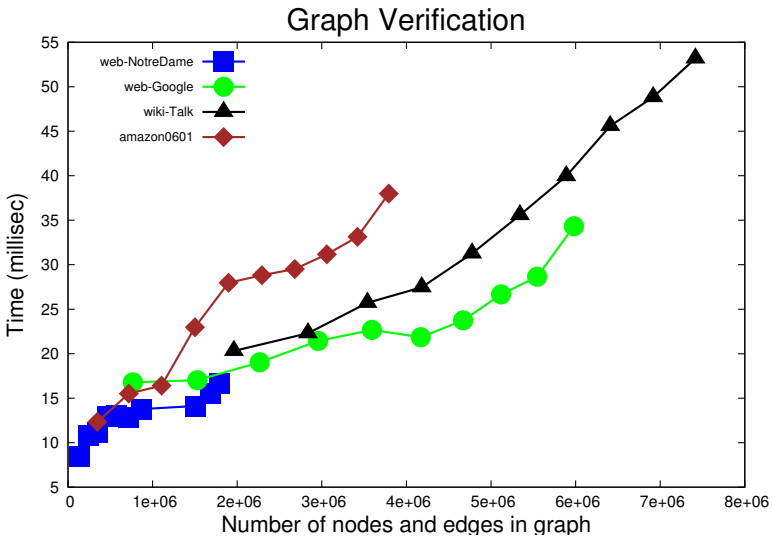


Fig. 4.8.: `gVrfy`: Time to verify graph integrity using HMAC tag.

tag t'_G. The computed tag value t'_G is compared to the input tag value t_G for integrity verification.

We observe that the time in both the above graphs increases almost linearly, as both the computation and verification algorithms use the function `DFS-Visit`(G, u) to traverse a graph, which has a linear time complexity of $\mathcal{O}(V + E)$. However, some spikes in the time curves can be explained as follow: The `JGraphT` library `Subgraph` method generates an induced subgraph given a set of vertices (which are selected randomly) as input from the original base graph G. The subgraphs can have a new topology each time the `Subgraph` method is called for a given input size of vertices. The spikes are the result of computations over subgraphs having more dense neighborhoods and thus requiring more computation for determining "local integrity verifiers" `xor-out`(u) for all the vertices and vice versa. For this reason, we average out our execution time results over a run of 10 iterations.

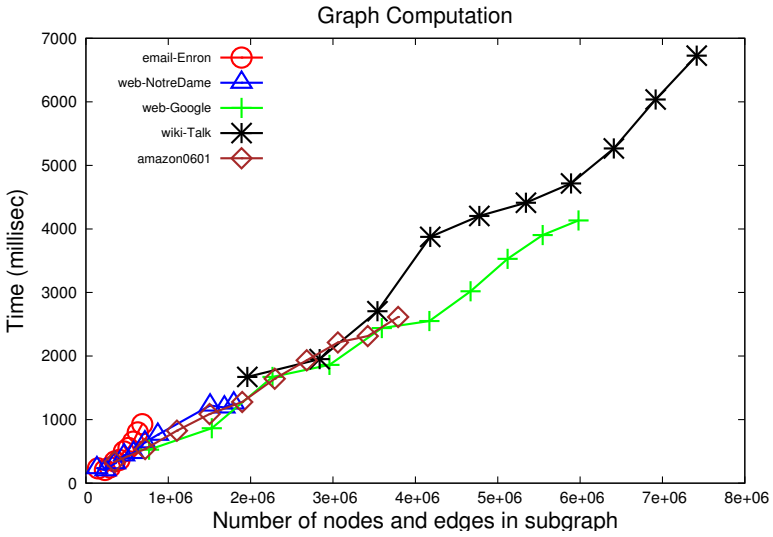


Fig. 4.9.: `rgHMAC`: Time to compute graph HMAC tag.

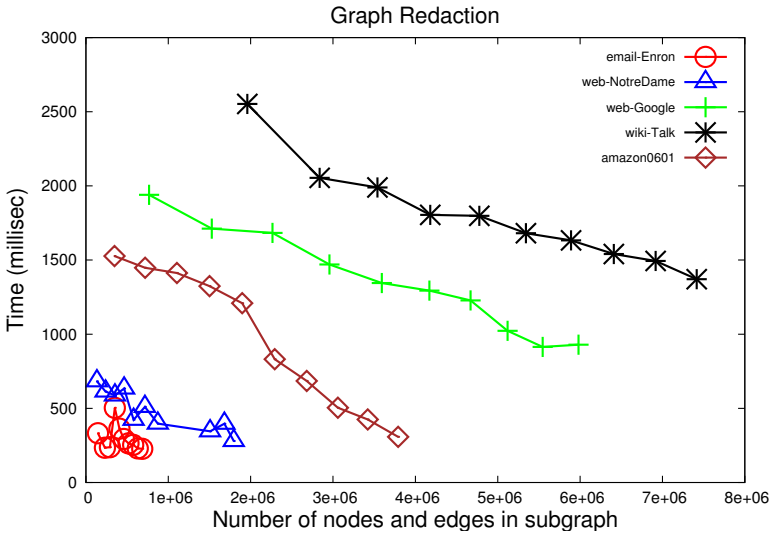


Fig. 4.10.: `gRedact`: Time to compute query result $\mathcal{R}$ and verification object $\mathcal{VO}$.

Results for `rgHMAC`

Computation time: In Fig. 4.9, we report the results for redacted graph HMAC tag computation time versus the number of nodes and edges in the input graph dataset. We observe that as we increase the size of input graph, the computation

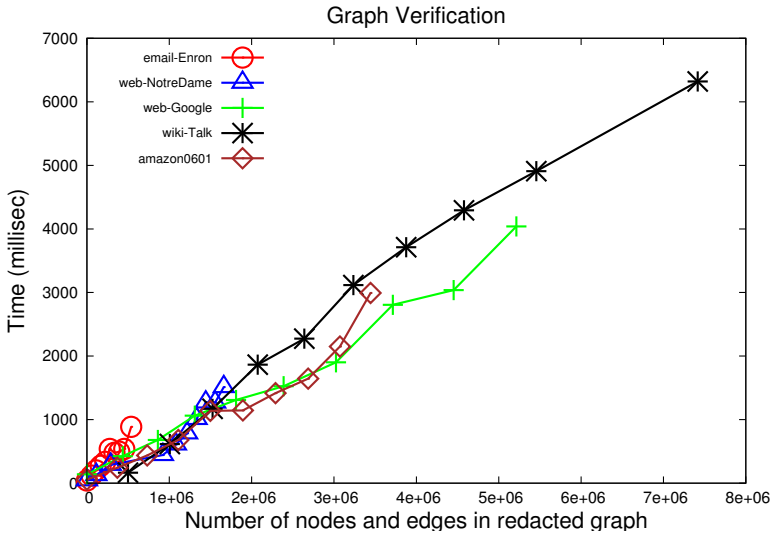


Fig. 4.11.: `rgVrfy`: Time to verify the integrity of query result $\mathcal{R}$.

time also increases. The algorithm `rgHMAC` computes the vertex and edge hash value `hashVal` for all the vertices and edges and provides this to the third-party $\mathcal{SP}$. Moreover, the algorithm computes the "local integrity verifier" `out-xor`(u) for all the vertices and recursively computes the *ghash* value. *ghash* value is computed by taking an exclusive-or $\oplus$ of the vertex hash value `hashVal` with the previously computed *ghash* value in post-order.

Redaction time: In Fig. 4.10, we report the results for graph redaction versus the size of graph G'_δ. We observe that, as we increase the graph G'_δ size, the time to compute the query result or redacted graph $\mathcal{R} = G_\delta(V_\delta, E_\delta)$ decreases. The reason being for a large input graph G'_δ size, the corresponding redacted graph $G_\delta = G \setminus G'_\delta$ is small. The redaction algorithm computes the vertex and edges hash value `hashVal` for all vertices and edges in the query result $\mathcal{R} = G_\delta$ and also determines two sets of verification objects: *vo-g* and *vo-out*. The *vo-g* computation entails performing exclusive-or $\oplus$ operation on the hash value `hashVal` of all the vertices in the set $V \setminus V_\delta$. These hash values need not be computed locally by algorithm `gRedact`, as they are provided by the third-party $\mathcal{SP}$ as part of input. As the size of graph G_δ reduces, so

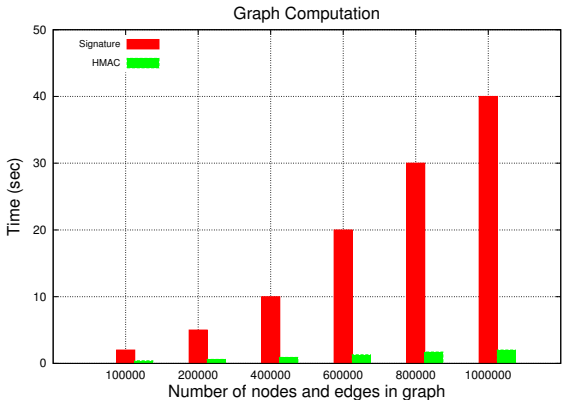
does the size of set *vo-out*. The reason being a smaller number of descendant nodes are present in $V \setminus V_\delta$ for the parent vertices in the set V_δ. This results in a reduced query processing time for the algorithm and hence yields progressively decreasing time-slope values. From Fig. 4.10, We notice that the relative difference between time curves for a given point on x-axis is in the order of largest to smallest graph dataset size for all x-axis values. The reason being for a fixed G_δ size the corresponding graph $G' = G \setminus G_\delta$ is larger for large graphs; thus, requiring the algorithm to perform more computation of determine the sets *vo-g* and *vo-out*.

Verification time: Fig. 4.11 shows the time performance results for query result integrity verification as a function of number of vertices and edges in the result graph G_δ. We observe that the verification time increases linearly as we increase the size G_δ. The verification time involves computing the hash value `hashVal` for all the vertices in V_δ and edges E_δ, which requires computing the "local integrity verifier" `out-xor`(u) and using verification object *vo-out* during the hash value computation. The complete *ghash* value is computed by taking an exclusive-or $\oplus$ of *vo-g* value with the *ghash* value computed for the redacted graph G_δ. Finally, for verification, the algorithm compares the computed tag value t'_G with the input tag t_G value.

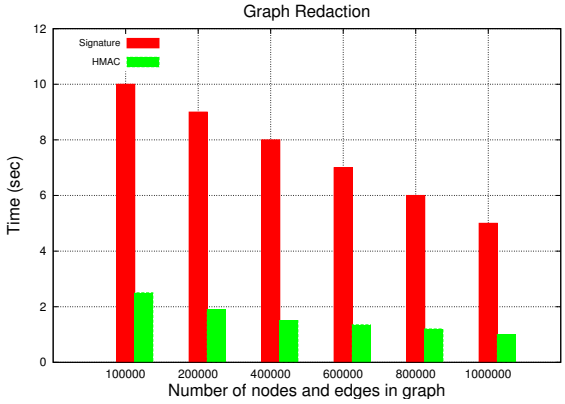
Comparison of rgHMAC with signature-based scheme

Figs. 4.12(a)-(c) show the comparison results for the proposed HMAC-based scheme with the signature-based scheme [53] in terms of time associated with computation, redaction (query result $\mathcal{R}$ and verification object $\mathcal{VO}$ generation), and query result integrity verification. Notice HMAC-based scheme outperforms the signature-based scheme by at least 4 times in terms of execution time for redaction and by at least 2 times in terms of execution time for integrity verification. Similarly, the HMAC-based scheme is more than an order of magnitude faster in terms of execution time during the computation stage. The reason being that the signature-based scheme uses i) computationally expensive modular exponentiation that is a dominant cost factor in

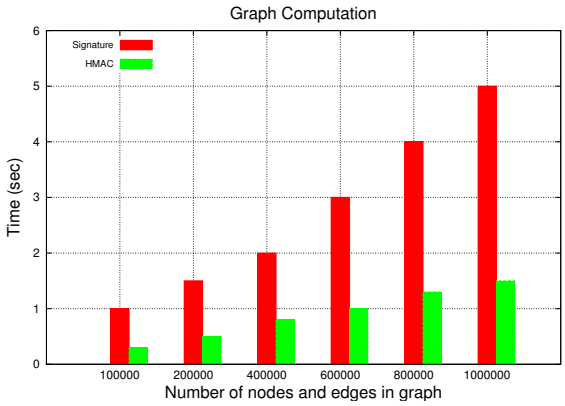
signing, redaction, and verification stages, and ii) computes secure names for keeping information about order among the siblings of parent nodes.



(a) Computation time.



(b) Redaction time.



(c) Verification time.

Fig. 4.12.: Comparison with [53].

4.7 Related Work

The problem of integrity verification of graph data and graph query result(s) is an important problem that has been fueled by the growing applications of graph datasets in a wide range of applications. However, integrity verification of graph data and graph query results using HMACs has been studied for the first time in the paper.

Merkle hash trees (MHT) is one of the most widely used techniques [60] for integrity assurance of DAGs [61] and for query result verification [8]. In [56], the authors define the formal security models of hashing schemes for graphs, and perfectly collision-resistant hashing schemes for graphs. They proposed the first constructions for general graphs that includes not only trees and graphs but also graphs with cycles and forests. The schemes uses pre- and post-order numbers of vertices to convert a cyclic graph into a two-level MHT (called *efficient-tree*). The proposed scheme is linear in the number of vertices and edges in the graph dataset. A structure-based routing scheme for XML-data dissemination to users under an access control policy is presented in [62]. The scheme proposes the notion of encrypted post-order numbers to support the integrity and confidentiality requirements of XML content. [63] proposes an integrity assurance technique referred to as 'structural signatures scheme' for trees in order to compute signatures of redactable subtrees. The scheme is based on the structure of the tree as defined by tree traversals (pre-order, post-order, in-order) and is defined using a randomized notion of such traversal numbers. With respect to MHT, it incurs comparable cost for signing the trees and incurs lower cost for user-side integrity verification. However, (1) a formal security model for the proposed scheme is not given, and (2) it is quite expensive in terms of the number of signatures computed, which are linear in the size of the tree/graph. [64] proposes a formal model for the notion of structural signatures for trees and have given a construction for the authentication of subtrees, which is linear in the number of nodes and quadratic in the number of siblings per node.

We have proposed two HMAC based data integrity verification schemes for graph and redacted graphs; we give formal definitions and show our schemes are linear in the number of vertices and edges in the graphs. [65] proposes two schemes on how to *authenticate* DAGs and directed cyclic graphs *without leaking*, which are the first such schemes in the literature. The schemes are based on the structure of graph as defined by DFS traversals and aggregate signatures (Computational Diffie-Hellman problem). The schemes minimize the number of authentication units that a user must receive in order to perform verification to $\mathcal{O}(1)$. The security of such schemes is based on the security of cryptographic hash function (random oracles) and aggregate signatures. [53] proposed a formal security model for leakage-free redactable signatures (LFRS) that is general enough to address authentication of not only trees but also graph and forests (disconnected trees/graphs). They also formally define the notion of secure names, which felicitate verification of ordering between sibling/nodes. Their proposed construction has linear complexity and outputs only one signature (optimal) that is stored, transmitted and used for authentication of a tree, graph and forest. [66] describe integrity and confidentiality preserving schemes for a DAG model of provenance database. Digital signature are used to sign the nodes and the relationships between them. An access control model based on paths on the provenance graph is proposed to preserve confidentiality of the nodes and edges in the provenance graph Fan et al. [8] present a framework for authentication of subgraph query services in outsourced graph databases. They propose an index Interaction-aware Feature-subgraph Tree (`IFTree`) to minimize I/O cost associated with *filtering-and-verification* requirement for processing subgraph query. In addition, `IFTree` is extended to authenticate subgraph query. The proposed extension is called `MIFTree`.

Digital signatures support not only integrity of data but also security properties like authentication of data source and non-repudiation [54]. However, in many practical contexts, data protection requirements include only integrity verification of data as other properties of digital signatures incur additional costs. To that end, MACs have been developed for integrity verification as they have been shown to be more

faster than RSA digital signatures [54]. There has been no wok on HMACs for graph data and graph query result(s). Therefore, the focus of this paper is to develop the notion of HMACs for graphs and graph query results. The proposed schemes exploit the security properties of XOR operator (especially when applied a key (k, r), one-way hash functions, and DFS traversal of graphs) in an efficient yet provable manner. We provide formal definitions and constructions for the proposed schemes, and our experimental results in Section 4.6 corroborate that our schemes are more efficient than previously proposed digital signature-based schemes.

4.8 Summary

Graphs are used for representing and understanding objects and their relationships for numerous applications. Graph databases are being used for managing several types of linked data. Some leading examples include social networks, biological networks, semantic Web, XML documents, and financial databases. However, the existing integrity assurance schemes for such data are neither scalable nor efficient – such schemes for graph data are based on digital signature schemes. In this paper, we have proposed two efficient integrity verification schemes – the first HMAC-based schemes for graphs and query results for graphs. The schemes exploit the security properties of $\oplus$ operator (especially when applied with a key (k, r), one-way hash functions, and depth-first traversal of graphs) in an efficient yet provable manner. The schemes rely on the local/global integrity verifiers of vertices and edges. The proposed schemes can prove to satisfy the security requirement in terms of structural/attribute modification to the graphs and redacted graphs. Our schemes are linear in the number of vertices and edges in the graphs, i.e., they have optimal complexity. We compute one HMAC value for `gHMAC` scheme and two other verification objects for redaction for `rgHMAC` scheme that are shared with the verifier. Therefore, our scheme is efficient both in processing time for query result and verification object transmission to the user. Our experiments show that HMAC-based schemes are efficient as compared to the digital

signature-based schemes for same size of graphs. Moreover, computing HMACs for graphs as large as 8 million vertices and edges takes as little as 55 milliseconds.

5. CONCLUSIONS

Section 5.1 summarizes the research contributions of this dissertation, Section 5.2 discusses the limitations of proposed methodologies, and and Section 5.3 proposes directions for the future research work.

5.1 Summary of Contributions

This dissertation addresses two notable challenges for graph databases; i) how to ensure users' privacy in published graph data under access control policy enforcement, and ii) how to verify the integrity and query results of graph datasets under both two-party and third-party data distribution environments.

To address the first challenge, a graph data publishing framework has been proposed that provides safeguard against data privacy breach through anonymization while enforcing access rules to satisfy the security protection requirements specified by the data publisher. We prove that the design of this framework poses a conflicting goal between privacy and access control. We formulate the privacy and access constraints on graph data as the k-anonymous bi-objective graph partitioning (k-BGP) problem. We show that the problem is NP-complete and provide new heuristic solutions. To the best of our knowledge, this is the first scheme that studies the interplay between RBAC policy constraints and privacy protection for graph data.

To address the second challenge, a cryptographic security model based on Hash Message Authentication Codes (HMAC) has been proposed. The model ensures integrity and completeness verification of data and query results under both two-party and third-party data distribution environments. Unique solutions based on HMACs for integrity verification of graph datasets are developed and detailed security analy-

sis of the proposed schemes is provided. Extensive experimental evaluation has been conducted to illustrate the performance of the proposed algorithms.

5.2 Limitations of the Proposed Methodologies

For the research presented in this dissertation, we have made several assumptions regarding the database environment which in practical life may need to be overcome in order to develop viable solutions for privacy, access control, and integrity for graph databases. Some of the limitations and assumptions include the following: We have made the assumption that the data graph model is static. However, this assumption may not hold for several application areas as mentioned in Chapter 1. For example, OSNs can change and evolve frequently over time. Such dynamic changes in graph databases are not addressed in our research methodology.

The privacy-enhanced access control mechanism presented in Chapter 3 assumes a relaxed access control policy. The interaction between privacy and strict access control policy has not been addressed in our work.

For the integrity verification schemes for graph databases and query results presented in Chapter 4, we do not assume any access control mechanism and primarily focus on integrity verification of problem. In reality access control mechanism is generally an added layer of security in addition to integrity verification system. The interaction between integrity and access control methods has not been discussed in this dissertation.

5.3 Future Work

Below we provide some directions for future work.

5.3.1 A Privacy Mechanism for Access-Controlled Dynamic Graph Databases

In a dynamic scenario, the graph structure may need to be updated because of the addition or removal of vertices and edges. Adding or removing new information while simultaneously meeting the privacy and access control requirements is a challenge that requires further research.

Employing a strict access control policy instead of a relaxed access control policy for a shared role workload requires the dataset to be partitioned into two types of groups: i) shared data region, and ii) non-shared data region. However, generating k-anonymous partitions for both these groups separately requires developing intelligent partitioning schemes to address corner cases due to non-uniform partitions being generated. This is a challenge that requires further research.

5.3.2 Integrity Verification for Dynamic Graph Databases

In a dynamic scenario, the graph structure may need to be updated because of addition or removal of vertices and edges. Adding or removing new information requires re-computing the integrity value for the updated graph database. Re-computing a new integrity value by just considering the added or removed vertices and edges is a challenge that requires further research. Moreover, the $\mathcal{VO}$ value may not remain valid and drastically impact the security guarantees for updated query results. Re-computing a new $\mathcal{VO}$ value by by just considering the added or removed vertices and edges is a challenge that requires further research

The $\mathcal{VO}$ computation may not be possible under an access control policy enforcement as a role is not able to see data beyond its privilege set.

5.3.3 Aggregation-based Schemes for Graph Data Integrity

The integrity verification schemes for graph databases and query results presented in Chapter 4 consider the whole graph for computing the integrity value. Computing the integrity of an aggregated graph is a challenge that requires further research.

Our proposed integrity verification schemes have linear time complexity $\mathcal{O}(V+E)$. Developing sub-linear time complexity algorithms for large graph databases that do not need to consider the whole graph for integrity verification is a challenge that requires further research.

REFERENCES

REFERENCES

[1] H. Zakerzadeh, C. C. Aggarwal, and K. Barker, "Big graph privacy," in *Proc. of the Workshops of the EDBT/ICDT*, 2015, pp. 255–262.

[2] J. Casas-Roma, J. Herrera-Joancomartí, and V. Torra, "A survey of graph-modification techniques for privacy-preserving on networks," *Artificial Intelligence Review*, pp. 1–26, 2016.

[3] X. Wu, X. Ying, K. Liu, and L. Chen, "A survey of privacy-preservation of graphs and social networks," in *Managing and Mining Graph Data*. Springer, 2010, pp. 421–453.

[4] L. Backstrom, C. Dwork, and J. Kleinberg, "Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography," in *Proc. of Int'l Conf. on World Wide Web (WWW)*, 2007, pp. 181–190.

[5] E. Bertino and R. Sandhu, "Database security-concepts, approaches, and challenges," *IEEE Trans. on Dependable and Secure Computing*, vol. 2, no. 1, pp. 2–19, 2005.

[6] A. Narayanan and V. Shmatikov, "De-anonymizing social networks," in *IEEE Symp. on S & P*, 2009, pp. 173–187.

[7] S. Ji, W. Li, P. Mittal, X. Hu, and R. Beyah, "Secgraph: A uniform and open-source evaluation system for graph data anonymization and de-anonymization," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 303–318.

[8] Z. Fan, Y. Peng, B. Choi, J. Xu, and S. S. Bhowmick, "Towards efficient authenticated subgraph query f in outsourced graph databases," *IEEE Trans. on Services Computing*, vol. 7, no. 4, pp. 696–713, 2014.

[9] M. J. Atallah, Y. Cho, and A. Kundu, "Efficient data authentication in an environment of untrusted third-party distributors," in *Proc. of Int'l Conf. on Data Eng. (ICDE)*, 2008, pp. 696–704.

[10] H. Hacigümüs, B. Iyer, and S. Mehrotra, "Providing database as a service," in *Proc. of Int'l Conf. on Data Eng. (ICDE)*, 2002, pp. 29–38.

[11] A. Amazon, "Amazon web services," *Available in: http:// aws.amazon.com/ es/ ec2/ (November 2012)*, 2010.

[12] A. J. Feldman, A. Blankstein, M. J. Freedman, and E. W. Felten, "Privacy and integrity are possible in the untrusted cloud." *IEEE Data Eng. Bull.*, vol. 35, no. 4, pp. 73–82, 2012.

[13] D. Menasce *et al.*, "Qos issues in web services," *IEEE Internet Computing*, vol. 6, no. 6, pp. 72–75, 2002.

[14] M. L. Yiu, E. Lo, and D. Yung, "Authentication of moving knn queries," in *Proc. of Int'l Conf. on Data Eng. (ICDE)*, 2011, pp. 565–576.

[15] L. Sweeney, "k-anonymity: A model for protecting privacy," *Int'l Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, 2002.

[16] A. Campan and T. M. Truta, "Data and structural k-anonymity in social networks," in *Privacy, Security, and Trust in KDD*. Springer, 2009, pp. 33–54.

[17] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis, "Resisting structural re-identification in anonymized social networks," *Proc. of the VLDB Endow. (PVLDB)*, vol. 1, no. 1, pp. 102–114, 2008.

[18] E. Zheleva and L. Getoor, "Preserving the privacy of sensitive relationships in graph data," in *Privacy, Security, and Trust in KDD*. Springer, 2008, pp. 153–171.

[19] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Mondrian multidimensional k-anonymity," in *Proc. of IEEE 22nd Int'l Conf. on Data Eng. (ICDE)*, 2006, pp. 25–25.

[20] S. Chaudhuri, T. Dutta, and S. Sudarshan, "Fine grained authorization through predicated grants," in *Proc. of IEEE 23rd Int'l Conf. on Data Eng. (ICDE)*, 2007, pp. 1174–1183.

[21] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy, "Extending query rewriting techniques for fine-grained access control," in *Proc. ACM Int'l Conf. Management of Data (SIGMOD)*, 2004, pp. 551–562.

[22] S. Contini and Y. L. Yin, "Forgery and partial key-recovery attacks on hmac and nmac using hash collisions," in *Advances in Cryptology–ASIACRYPT 2006*. Springer, 2006, pp. 37–53.

[23] T. Dierks and C. Allen, "RFC 2246: The TLS protocol version 1.0," 1999.

[24] D. Harkins, D. Carrel *et al.*, "The internet key exchange (IKE)," RFC 2409, november, Tech. Rep., 1998.

[25] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen, "Hotp: An hmac-based one-time password algorithm," Tech. Rep., 2005.

[26] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC Press, 2014.

[27] M. Bellare, R. Canetti, and H. Krawczyk, "Pseudorandom functions revisited: The cascade construction and its concrete security," in *Symp. on Foundations of Computer Science (FOCS)*, 1996, pp. 514–523.

[28] H. Krawczyk, M. Bellare, and R. Canetti, "RFC 2104: HMAC: Keyed-hashing for message authentication," 1997.

[29] S. Chaudhuri, R. Kaushik, and R. Ramamurthy, "Database access control and privacy: Is there a common ground?" in *CIDR*, 2011, pp. 96–103.

[30] R. Sayaf and D. Clarke, "Access control models for online social networks," *Social Network Engineering for Secure Web Data and Services*, pp. 32–65, 2012.

[31] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Trans. on Information and Syst. Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[32] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," in *Proc. ACM SIGCOMM*, 2009, pp. 135–146.

[33] T. Wang, M. Srivatsa, and L. Liu, "Fine-grained access control of personal data," in *Proc. of ACM Symp. on Access Control Models and Technologies (SACMAT)*, 2012, pp. 145–156.

[34] G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis, "Fast data anonymization with low information loss," in *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, 2007, pp. 758–769.

[35] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Workload-aware anonymization techniques for large-scale datasets," *ACM Trans. on Database Syst. (TODS)*, vol. 33, no. 3, p. 17, 2008.

[36] M. E. Nergiz and C. Clifton, "Thoughts on k-anonymization," *Data & Knowledge Engineering*, vol. 63, no. 3, pp. 622–645, 2007.

[37] V. S. Iyengar, "Transforming data to satisfy privacy constraints," in *Proc. ACM SIGKDD*, 2002, pp. 279–288.

[38] Z. Pervaiz, W. G. Aref, A. Ghafoor, and N. Prabhu, "Accuracy-constrained privacy-preserving access control mechanism for relational data," *IEEE Trans. on Knowledge and Data Eng.*, vol. 26, no. 4, pp. 795–807, 2014.

[39] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209–226, 1977.

[40] T. Tassa and D. J. Cohen, "Anonymization of centralized and distributed social networks by sequential clustering," *IEEE Trans. on Knowledge and Data Eng.*, vol. 25, no. 2, pp. 311–324, 2013.

[41] V. Ayala-Rivera, P. McDonagh, T. Cerqueus, and L. Murphy, "A systematic comparison and evaluation of k-anonymization algorithms for practitioners," *Transactions on Data Privacy*, vol. 7, no. 3, pp. 337–370, 2014.

[42] M. Yuan, L. Chen, and P. S. Yu, "Personalized privacy protection in social networks," *Proc. of the VLDB Endow. (PVLDB)*, vol. 4, no. 2, pp. 141–150, 2010.

[43] B. Zhou and J. Pei, "Preserving privacy in social networks against neighborhood attacks," in *Proc. of IEEE 28th Int'l Conf. on Data Eng. (ICDE)*, 2008, pp. 506–515.

[44] K. Liu and E. Terzi, "Towards identity anonymization on graphs," in *Proc. ACM Int'l Conf. Management of Data (SIGMOD)*, 2008, pp. 93–106.

[45] L. Zou, L. Chen, and M. T. Özsu, "K-automorphism: A general framework for privacy preserving network publication," *Proc. of the VLDB Endow. (PVLDB)*, vol. 2, no. 1, pp. 946–957, 2009.

[46] J. Cheng, A. W.-c. Fu, and J. Liu, "K-isomorphism: privacy preserving network publication against structural attacks," in *Proc. ACM Int'l Conf. Management of Data (SIGMOD)*, 2010, pp. 459–470.

[47] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava, "Class-based graph anonymization for social network data," *Proc. of the VLDB Endow. (PVLDB)*, vol. 2, no. 1, pp. 766–777, 2009.

[48] C. Dwork, "Differential privacy: A survey of results," in *Int'l Conf. on Theory and Applications of Models of Computation*. Springer, 2008, pp. 1–19.

[49] P. Mittal, C. Papamanthou, and D. X. Song, "Preserving link privacy in social network based systems," in *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*, 2013.

[50] C. Clifton and T. Tassa, "On syntactic anonymity and differential privacy," *Trans. Data Privacy*, vol. 6, no. 2, pp. 161–183, 2013.

[51] H. V. Jagadish and F. Olken, "Database management for life sciences research," *SIGMOD Rec.*, vol. 33, no. 2, pp. 15–20, 2004.

[52] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," *Proc. of VLDB Endow. (PVLDB)*, vol. 2, no. 1, pp. 718–729, 2009.

[53] A. Kundu, M. J. Atallah, and E. Bertino, "Efficient leakage-free authentication of trees, graphs and forests." *IACR Cryptology ePrint Archive*, vol. 2012, p. 36, 2012.

[54] W. E. Freeman and E. L. Miller, "An experimental analysis of cryptographic overhead in performance-critical systems," in *Proc. of Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication (MASCOTS)*, 1999, pp. 348–357.

[55] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 3rd Ed.* MIT Press, 2009.

[56] M. U. Arshad, A. Kundu, E. Bertino, K. Madhavan, and A. Ghafoor, "Security of graph data: hashing schemes and definitions," in *Proc. of ACM Conf. on Data and Application Security and Privacy (CODASPY)*, 2014, pp. 223–234.

[57] R. C. Merkle, "A certified digital signature," in *CRYPTO*, 1989, pp. 218–238.

[58] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine, "A general model for authenticated data structures," *Algorithmica*, vol. 39, pp. 21–41, 2004.

[59] Z. Fan, B. Choi, Q. Chen, J. Xu, H. Hu, and S. S. Bhowmick, "Structure-preserving subgraph query services," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 8, pp. 2275–2290, 2015.

[60] R. C. Merkle, "A certified digital signature," in *CRYPTO*, 1989, pp. 218–238.

[61] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine, "A general model for authenticated data structures," *Algorithmica*, vol. 39, no. 1, pp. 21–41, 2004.

[62] A. Kundu and E. Bertino, "Secure dissemination of XML content using structure-based routing," in *Int'l Conf. on Enterprise Distributed Object Computing (EDOC)*, 2006, pp. 153–164.

[63] ——, "Structural signatures for tree data structures," *Proc. of the VLDB Endow. (PVLDB)*, vol. 1, no. 1, pp. 138–150, 2008.

[64] C. Brzuska, H. Busch, Ö. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder, "Redactable signatures for tree-structured data: Definitions and constructions," in *Int'l Conf. on Applied Cryptography and Network Security (ACNS)*, 2010, pp. 87–104.

[65] A. Kundu and E. Bertino, "How to authenticate graphs without leaking," in *Int'l Conf. on Extending Database Technology (EDBT)*, 2010, pp. 609–620.

[66] A. Syalim, T. Nishide, and K. Sakurai, "Preserving integrity and confidentiality of a directed acyclic graph model of provenance," in *Data and Applications Security and Privacy XXIV*. Springer, 2010, pp. 311–318.

[67] M. E. Dyer and A. M. Frieze, "Planar 3DM is NP-complete," *Journal of Algorithms*, vol. 7, no. 2, pp. 174–184, 1986.

[68] P. Rosenstiehl and R. E. Tarjan, "Rectilinear planar layouts and bipolar orientations of planar graphs," *Discrete & Computational Geometry*, vol. 1, no. 1, pp. 343–353, 1986.

[69] N. Mamoulis and D. Papadias, "Selectivity estimation of complex spatial queries," in *Int'l Symp. on Spatial and Temporal Databases*. Springer, 2001, pp. 155–174.

[70] Y. Tao, J. Sun, and D. Papadias, "Analysis of predictive spatio-temporal queries," *ACM Trans. on Database Syst. (TODS)*, vol. 28, no. 4, pp. 295–336, 2003.

APPENDICES

A: Proof of Theorem 3.3.1

Proof of Theorem 3.3.1: To prove this theorem, we use Theorem 3.1 [38] and need to first prove that Planer k-anonymous Graph Partitioning with Minimal $IL_S(\mathcal{P})$(Planar AGPMIL, for short) is NP-hard. The proof of Theorem 3.3.1 will follow as a direct consequence of above two proofs.

First, we state that the AGPMIL is NP-hard in the plane, and then generalize this result to the multidimensional Euclidean space.

PLANAR k-ANONYMOUS GRAPH PARTITIONING WITH MINIMUM $IL_S(\mathcal{P})$(Planar AGPMIL)

INSTANCE: A planar graph G with a set X of n points/vertices in the plane; an integer k, $n > k$.

QUESTION: Is there a clustering of X into a set of non-overlapping partitions $\mathcal{P} = \{P_1, \ldots, P_M\}$ such that $|P_i| \geq k$, and the structural information loss $IL_S(\mathcal{P})$ minimal?

Our NP-completeness proof of (Planar AGPMIL) is based on reduction from the following special version of the exact cover problem that is shown to be NP-complete by Dyer and Frieze [67].

PLANAR EXACT COVER BY 3-SETS (Planar X3C)

INSTANCE:

Consider a set Q of objects with $|Q| = 3r$. Let $T \subset Q \times Q \times Q$ be a set of triplets such that at most 3 triplets $t \in T$ can have a common object $q_i \in Q$. Such a mapping between $q_i \in Q$ and the triplet $t \in T$ to which it belongs to can form a bipartite association. An additional requirement of such a bipartite graph G is that it must be planer.

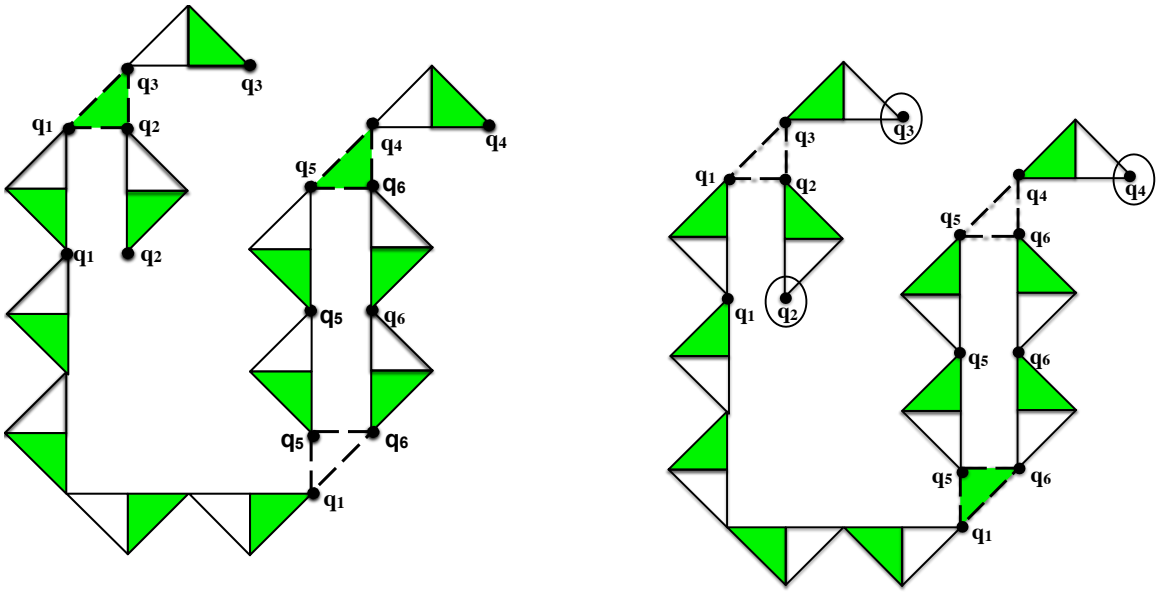
QUESTION: Does there exist such a bipartite association with a subset of r triples in T that contain all the elements of Q?

Let $Q = \{q_1, \ldots, q_{3r}\}$ and $T \subset Q \times Q \times Q$ be two sets of objects that constitute an instance of planar X3C. We construct a point set $X(Q, T)$ composed of sets Q and T such that the size of X is multiple of 3. This allows a grouping of points into a partition set $\mathcal{P} = \{P_1, \ldots, P_M\}$, where $|P_i| \geq k = 3$ and $IL_{S,1}(\mathcal{P})$ is minimum if and only if the planar X3C instance has a solution.

The reduction is based on calculating a *rectilinear planar layout* for bipartite graph G. Given a planar graph $G = (V, E)$, a rectilinear planar layout of G can be computed in time polynomial in the size of G as reported in Rosenstiehl and Tarjan [68].

In the set X of points each point $q_i \in Q$ is called an *element point*. A *triple triangle* is formed by three constituent points of a triplet $t = (t_1, t_2, t_3) \in T$ called *triple points*. We place the elements and the triple triangles somewhere at the corresponding line segments in the rectilinear layout. Our objective is establish a bipartite connection between *element points* and triple points by using equilateral right triangle Δ_0 with sides of lengths $1, 1, \sqrt{2}$. We assume that all points in $X(Q, T)$ are at integer coordinates.

A chain of diamonds is used to connect *element points* $q_i \in Q$ to *triple points*. A triangular diamond contains two triangles that are glued together by their sides of length 1. The chains of diamonds follow the line segments corresponding to the two vertices q_i and t_i and represent the connecting edge in the graph G. It can be



(a) (b)

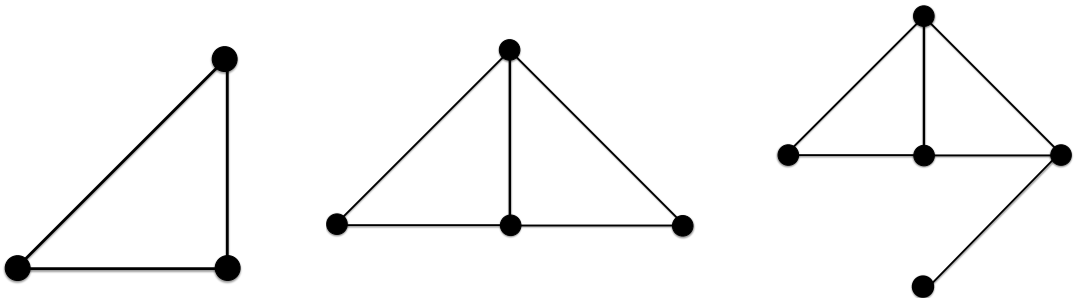
Fig. 1.: Partition into triangles different cases: (a) All graph vertices included in shaded partitions, solution to X3C exists; (b) Some graph vertices not included in shaded partitions, solution to X3C does not exist.

easily checked that this construction can be performed in polynomial time and $|X|$ is multiple of 3.

We assume that an *enlarging procedure* has been used on the rectilinear layout of graph G by multiplying all coordinates with a large positive integer. This ensures i) two distinct chains are generated sufficiently away from each other, and ii) an *element point* and its corresponding *triple points* can be connected exactly by a chain of diamonds.

We now prove that the set of points $X(Q,T)$ can be partitioned into $\mathcal{P} = \{P_1, \ldots, P_M\}$, such that $|P_i| \geq k$ and the $IL_{S,1}(\mathcal{P})$ is minimum if and only if Planar X3C problem has a solution.

Let us consider only partitions of size 3, 4 and 5 points. Fig. 2 shows different partition configurations generated with size 3, 4 and 5 points. Due to space limitation, we do not list any other possible groups. Fig. 2(a) states that no three points in $X(Q,T)$



(a) (b) (c)

Fig. 2.: Partitions of size 3, 4 and 5 points.

are at a distance less that 1 and $\sqrt{2}$ from each other, so the minimum $IL_{S,1}(\mathcal{P})$ for a partition with three vertices with three edges using $2|E_{P_i}|.\big(1 - \frac{2|E_{P_i}|}{P_i(|P_i|-1)}\big) = 0$; Similarly, for partitions of size 4, and 5 having 5 and 6 edges in Fig. 2(b) and Fig. 2(c), respectively, the minimum $IL_{S,1}(\mathcal{P})$ can be calculated as $\frac{5}{3}$ and $\frac{24}{5}$. Hence, it is easily observed that the partitioning of point set X has the minimal $IL_{S,1}(P_i) = 0 \times p = 0$ if all partitions are of size 3, where p is determined by reconstruction.

Finally, we claim that the partitioning of point set X into triangles with minimum intra-partition structural information loss $IL_{S,1}(\mathcal{P})$ is possible if and only if Planar X3C has a solution.

(If) We assume that there exists a partitioning of points $X(Q, T)$ into triangles with minimal $IL_{S,1}(\mathcal{P}) = 0 \times p = 0$. Consider some element point $q_i \in Q$, contained in exactly one partition (triangle) belonging to a chain of diamonds. Every other triangle must form a partition, and the corresponding point t_i on the other end of the chain cannot be covered by any triangle in this chain. Therefore, the corresponding three triple points must form another partition. However, the triangle(s) in the other chain(s) going away from q_i will cover the corresponding triple points, so these points cannot form a partition in the partitioning. This way, we may assign to each $q_i \in Q$ a unique triple in T. On the other hand, if we assign one q_i to some triple, the other two elements in this triple must be assigned to this triple also. For the other paths that

are excluded from q_i; may be they contain q_j that is not contained in any other triple. But this case cannot happen because we have a partition with minimal information loss, each q_i is in one partition (Fig. 1(a)). Obviously, this yields a solution to Planar X3C.

(Only if) We now assume that the Planar X3C has a solution $T' \subseteq T$. We construct a partitioning as follows. As a first step, each triple is mapped to a triple triangle as a partition in the partitioning set, while on the other hand all triples in $T \setminus T'$ are not. This uniquely determines which triangles in the chains are part of the partitioning. Since T' is a solution of Planar X3C, every element point in $X(Q,T)$ is exactly in one group.

Thus, it has been have shown that partitioning of X into partitions of three points with minimum $IL_{S,1}(\mathcal{P})$ is possible if and only if Planar X3C problem has a solution. Therefore, the partitioning of $\mathcal{P}$ with $|P_i| \geq 3$ has minimum $IL_{S,1}(\mathcal{P})$ if and only if Planar X3C problem allows a solution. This implies that the NP-complete problem (Planar X3C) is not harder than partitioning a set of points X into partitions P_i, such that $|P_i| \geq 3$ and $IL_{S,1}(\mathcal{P})$ is minimum. Since $IL_{S,1}(\mathcal{P})$ is due to a restricted version of the problem (i.e., where we do not consider the inter-partition edges, $|E_{P_i,P_j}| = \phi$) and this restricted version of problem contains a known NP-complete problem as a special case therefore, Planar AGPMIL is NP-hard in a plane.

We can now easily extend the results to show that a partitioning of $X(Q,T)$ points into partitions P_i with $|P_i| \geq 3$ with minimum $IL_S(\mathcal{P})$ is NP-hard in a the Euclidean space with dimensions ≥ 2.

The proof of Theorem 3.3.1 now follows directly from the above theorem and Theorem 3.1 [38]. Zahid et al. [38] prove that finding k-anonymous partitioning with imprecision bounds is NP-complete. We have shown that k-anonymous graph partitioning with minimum $IL_S(\mathcal{P})$ is NP-complete. Therefore, the decisional k-BGP problem is NP-complete with either or both constraints.

Example 8 *Suppose* $Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$ *and* $C = \{\{q_1, q_2, q_3\}, \{q_4, q_5, q_6\}, \{q_1, q_5, q_6\}\}$. *The solution to PX3C instance consists of* $\{\{q_1, q_2, q_3\}, \{q_4, q_5, q_6\}\}$ *since these two* 3-

element sets provide an exact cover for Q. It can be noticed from the construction in Fig. 1(a), the solution shown by shaded triangles is the only solution to the AGPMIL problem.

If we attempt to construct some other partition of the induced graph into disjoint triangles (e.g., Fig. 1(b)), we end up with one or more of the $q_i's \in \{q_1, q_2, q_3, q_4, q_5, q_6\}$ not being included in any of the shaded triangles. In other words, we do not generate a partition with desired properties. It can be noticed from Fig. 1(b), that vertices $\{q_2, q_3, q_4\}$ are not included in any partition.

B: Proof of Theorem 3.5.1

Proof We assume that the adversary $\mathcal{A}$ has both the structural and attribute information as background knowledge for re-identification of a target node x. Suppose the adversary $\mathcal{A}$ executes a structural query $Q_S(x)$ [69] to determine the set of feasible candidate partitions $cand_{Q_S}(x)$. With no other information, the adversary $\mathcal{A}$ then may choose any partition $y \in cand_{Q_S}(x)$ arbitrarily. The probability of a particular candidate partition y for node a x, therefore, can be given as:

$$Pr_{Q_S}(y) = \frac{1}{|cand_{Q_S}(x)|}. \tag{1}$$

In addition, as mentioned above, the adversary $\mathcal{A}$ may also have some attribute $Attr \subseteq \{QI_1, \ldots, QI_d\}$ information as background knowledge for the target node x. Given a set N of d-dimensional data points enclosed in a volume $U_{vol} = \prod_{i=1}^{d}[U_i]$, where $[U_i^{min}, U_i^{max}]$ is the range of domain in dimension i, the proposed algorithms partition the data space into $|\mathcal{P}|$ hyper-rectangles as a result of k-anonymous partitioning. The adversary $\mathcal{A}$ may then execute a query Q_A to determine the feasible candidate partitions $cand_A(x)$. The query Q_A can be considered as forming a subspace with dimension $|Attr|$ and having volume $\prod_{i=1}^{|Attr|}[L_i] \times \prod_{j=1}^{(d-|Attr|)}[U_i] \subseteq U_{vol}$, $L_i = [L_i^{min}, L_i^{max}]$. The size of the set $cand_{Q_A}(x)$ can be determined by computing the number of partitions that intersect a query Q_A and is estimated as: $|cand_{Q_S}(x)| \times \prod_{j=1}^{|Attr|} \min\left(1, \frac{\overline{S}_j + L_j}{U_j}\right)$ according to [70], where $\overline{S}_j$ is the average length of projections of all partitions $P_j \in \mathcal{P}$ along dimensions j and L_j and U_j are the corresponding projections for query Q_A and total domain space U_{vol} along dimension j. The probability that a partition, say P_i, intersects a query Q_A is given as: $\prod_{j=1}^{|Attr|} \min\left(1, \frac{\overline{S}_j + L_j}{U_j}\right)$, where the product term, also known as the Minkowski sum, determines the selectivity of the query Q_A. It is quite intuitive that increasing the

dimension or the number of attributes of query Q_A has the effect of reducing the size of $cand_A(x)$ set. The probability of a candidate $y \in cand(x)$ is given as follows:

$$Pr_{Q_A(Q_S)}(y) = \frac{1}{|cand_{Q_S}(x)| \times \prod_{j=1}^{|Attr|} \min\left(1, \frac{\overline{S}_j + L_j}{U_j}\right)}. \tag{2}$$

Similarly, the adversary $\mathcal{A}$ may first execute the query Q_A on $\mathcal{P}$ set to determine the number of feasible candidates as: $cand_{Q_A}(x) = |\mathcal{P}| \times \prod_{j=1}^{|Attr|} \min\left(1, \frac{\overline{S}_j + L_j}{U_j}\right)$ and then may apply the the structural query Q_S on $cand_{Q_A}(x)$ to further refine the candidate set.

$$Pr_{Q_S(Q_A)} = \frac{1}{cand_{Q_S}(x)}. \tag{3}$$

The disclosure-risk of a node x depends on the size of the set $|cand(x)|$ and the likelihood of candidates. The *exact* re-identification of a target node x by an adversary $\mathcal{A}$ requires the identification of a single candidate partition $y \in cand_Q(x)$.

Therefore, combining equations (2) and (3), the probability of a candidate $y \in cand_Q(x)$ for x is upper-bounded by:

$$Pr(y) \leq \max\left\{Pr_{Q_S(Q_A)}(y), Pr_{Q_A(Q_S)}(y)\right\}. \tag{4}$$

As the total number of candidate nodes per partition is k, the probability of successfully identifying a target node x given a candidate partition y is given as:

$$Pr(x|y) = \frac{1}{k}. \tag{5}$$

Therefore, combining Equations (4) and (5) the probability of re-identifying a node x is given as:

$$Pr(Re\text{-}id(x)) \leq \frac{1}{k} \times \min\left\{Pr_{Q_S(Q_A)}(y), Pr_{Q_A(Q_S)}(y)\right\}.$$

■

VITA

VITA

Muhammad Umer Arshad was born in Nizamabad, District Gujranwala, Pakistan, in 1980. He completed his B.Sc., and M.Sc., degrees in Electrical Engineering (EE) from University of Engineering and Technology (UET), Lahore, Pakistan, in 2002, and 2005, respectively. Before joining Purdue in Fall 2006, he was a Lecturer in the EE Department of UET, Lahore. While at Purdue, he completed an MS degree in Computer Engineering (CE) in Fall 2014. During his Ph.D., he has had the opportunity to work at VMWare, Yahoo!, and EMC2 as an intern in their cloud computing divisions. His broad research interests are in areas related to Data Infrastructure Systems, Cloud Computing, and Information Security & Privacy.