Doctoral Thesis

A Study on Crypto-Assisted Differentially Private Graph Analysis

Shang Liu

June 2024

Department of Social Informatics
Graduate School of Informatics
Kyoto University

Doctoral Thesis
submitted to Department of Social Informatics,
Graduate School of Informatics,
Kyoto University
in partial fulfillment of the requirements for the degree of
DOCTOR of INFORMATICS

Thesis Committee: Takayuki Ito, Professor
Keishi Tajima, Professor
Hisashi Kashima, Professor

A Study on Crypto-Assisted Differentially Private Graph Analysis*

Shang Liu

Abstract

Graph data analysis has become highly popular across various domains, including social networks, transportation systems, and protein forecasting, due to its widespread applicability. Standard graph analytics encompass degree distribution, subgraph counting (e.g., k-star counting, triangle counting), and others. Nevertheless, most graph analytics are performed on sensitive data, posing a risk of data compromise through the analytical results. Thus, developing methods that enable these graph analytics while ensuring individual privacy is of paramount importance.

Differential privacy (DP) has been widely used to provide formal privacy protection. It safeguards individual privacy against adversaries with arbitrary background knowledge and has emerged as the gold standard for private graph analytics. However, DP ensures privacy by adding noise to sensitive information, which can impact overall utility. Conversely, cryptography has long been the foundation for secure communication in the presence of adversarial behavior. It ensures data confidentiality, integrity, and authenticity across various digital platforms and communications. Nevertheless, cryptography does not offer a formal privacy guarantee like DP. In the literature of private graph analysis, DP and cryptography have typically been studied separately. Combining these two approaches holds promise for improving the trade-off between utility and privacy in differentially private graph analytics.

In this dissertation, we present three works on exploiting crypto-assisted differentially private graph analytics. First, we introduce an approach that demonstrates how cryptography can enable high utility in publishing differentially private degree distribution under node-local differential privacy. Second, we present

*Doctoral Thesis, Department of Social Informatics, Graduate School of Informatics, Kyoto University, KU-I-DT6960-33-9715, June 2024.

CARGO, a crypto-assisted differentially private triangle counting system that achieves high-utility triangle counting of a central model without relying on a trusted server, akin to a local model. Finally, we introduce FEAT, a federated graph analytic framework that achieves an optimal tradeoff between utility and privacy by integrating cryptography into differential privacy. Specifically, we address the following three research topics:

- **Topic 1: Crypto-assisted differentially private degree distribution.** We propose an algorithm to publish the degree distribution with Node-LDP by exploring how to select the graph projection parameter in the local setting. Specifically, we design a crypto-assisted local projection method based on cryptographic primitives, achieving higher accuracy than our baseline pureLDP local projection method.

- **Topic 2: Crypto-assisted differentially private triangle counting.** We propose a crypto-assisted differentially private triangle counting system, named CARGO, leveraging cryptographic building blocks to improve the effectiveness of differentially private triangle counting without the assumption of trusted servers. It achieves high utility similar to the central model but without the need for a trusted server, akin to the local model.

- **Topic 3: Crypto-assisted differentially private federated graph analytics.** We propose a federated graph analytic framework, named FEAT, which enables arbitrary downstream common graph statistics while preserving individual privacy. We design a differentially private set union (DPSU) algorithm, which ensures that sensitive information is reported only once and the output global graph is protected under differential privacy.

Keywords: Differential Privacy, Cryptography, Graph Analysis

Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

Graph analysis is a powerful tool for uncovering meaningful insights in various types of graph data, such as social network analysis, transportation analysis, and epidemiological network analysis. For instance, the degree distribution [9] (e.g., the number of edges connected to a node) can reveal the connectivity in a social graph. Subgraph counts [10] (i.e., the number of triangles or stars) can measure centrality properties in a graph, such as the clustering coefficient [11], which represents the probability that two friends of an individual are also friends with one another. However, directly releasing graph statistics may leak sensitive information about individuals [12]. Most graph analytics are conducted on sensitive data, which could be compromised through the results of these graph statistics. Therefore, there is a need to develop solutions that can analyze these graph properties while preserving the privacy of individuals in a graph.

Recently, many research efforts have focused on the problem of publishing sensitive graph statistics under differential privacy (DP) [13, 14]. DP provides individual privacy against adversaries with arbitrary background knowledge and has emerged as the gold standard for private analytics. Compared with previous privacy models (e.g., k-anonymity, l-diversity, t-closeness), DP can resist most privacy attacks and provide a provable privacy guarantee. However, DP achieves

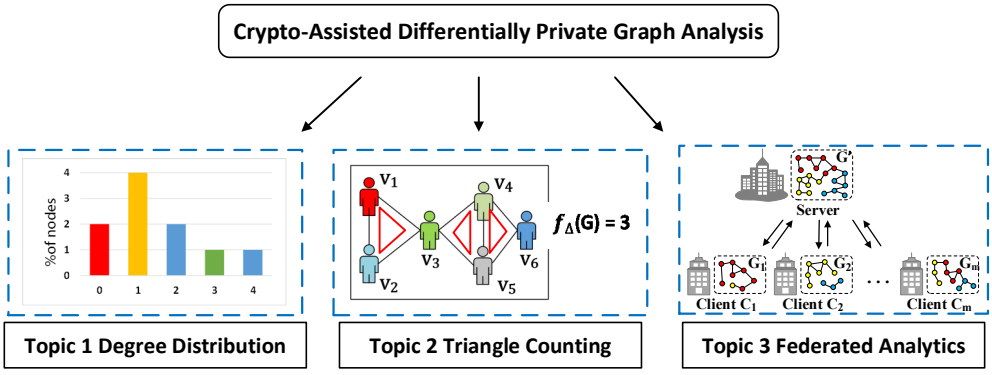


Figure 1.1. Overview of Doctoral Thesis

this formal privacy guarantee by adding noise to sensitive information, which can affect overall utility. Conversely, cryptography has long served as the foundation for secure communication in the presence of adversarial behavior. It ensures data confidentiality, integrity, and authenticity across various digital platforms and communications. However, cryptography does not offer a formal privacy guarantee like DP; adversaries can still infer sensitive information from graph analytic results. In the literature of private graph analysis, DP and cryptography have typically been studied separately. In this thesis, we explore how to improve the trade-off between privacy and utility in differentially private graph analysis by leveraging cryptographic techniques.

1.2 Overview of Our Studies

In this dissertation, we focus on three kinds of graph analytic tasks, with the objective of exploring how cryptography can improve the utility of differentially private graph analytics. Figure 1.1 provides an overview of this thesis, which includes three research topics. We propose different methods for three common graph tasks by leveraging cryptographic techniques to assist differential privacy in improving accuracy in graph analysis, including degree distribution, triangle counting, and federated graph analytics. As detailed motivations will be described in the following chapters, we briefly introduce the motivation, target, and approach of each research topic in this section:

1.2.1 Crypto-assisted differentially private degree distribution

Publishing graph statistics under node differential privacy [3,15–17] has attracted much attention since it provides a stronger privacy guarantee than edge differential privacy [18–21]. Existing works related to node differential privacy typically assume a trusted server that holds the entire graph. However, in many applications, a trusted curator is often not available due to privacy and security concerns.

In this chapter, for the first time, we investigate the problem of publishing graph statistics under Node Local Differential Privacy (Node-LDP), which does not rely on a trusted server. We propose an algorithm to publish the degree distribution with Node-LDP by exploring how to select the graph projection parameter in the local setting and how to execute the graph projection locally. To be specific, we propose a crypto-assisted local projection method based on cryptographic primitives, achieving higher accuracy than our baseline pureLDP local projection method. Furthermore, we enhance our baseline graph projection method from node-level to edge-level, preserving more neighboring information and thus providing better utility. Extensive experiments on real-world graphs demonstrate that crypto-assisted parameter selection offers better utility than pureLDP parameter selection, and edge-level local projection provides higher accuracy than node-level local projection, with improvements of up to 57.2% and 79.8%, respectively.

1.2.2 Crypto-assisted differentially private triangle counting

Differentially private triangle counting in graphs is essential for analyzing connection patterns and calculating clustering coefficients while protecting sensitive individual information. Previous works have relied on either central [2, 15, 22] or local models [5–7, 23] to enforce differential privacy. However, a significant utility gap exists between the central and local models of differentially private triangle counting, depending on whether a trusted server is needed. In particular, the central model provides high accuracy but necessitates a trusted server. Conversely, the local model does not require a trusted server but suffers from limited accuracy.

In this chapter, we introduce a crypto-assisted differentially private triangle counting system, named CARGO, leveraging cryptographic building blocks to improve the effectiveness of differentially private triangle counting without the assumption of trusted servers. It achieves high utility similar to the central model but without the need for a trusted server like the local model. CARGO consists of three main components. First, we introduce a similarity-based projection method that reduces the global sensitivity while preserving more triangles via triangle homogeneity. Second, we present a triangle counting scheme based on additive secret sharing that securely and accurately computes the triangles while protecting sensitive information. Third, we design a distributed perturbation algorithm that perturbs the triangle count with minimal but sufficient noise. We also provide a comprehensive theoretical and empirical analysis of our proposed methods. Extensive experiments demonstrate that our CARGO significantly outperforms the local model in terms of utility and achieves high-utility triangle counting comparable to the central model.

1.2.3 Crypto-assisted differentially private federated graph analytics

Collaborative graph analysis across multiple institutions is becoming increasingly popular. Realistic examples include social network analysis across various social platforms, financial transaction analysis across multiple banks, and analyzing the transmission of infectious diseases across multiple hospitals. We define the *federated graph analytics*, a new problem for collaborative graph analytics under differential privacy. Although differentially private graph analysis has been widely studied [1–8, 24, 25], it fails to achieve a good tradeoff between utility and privacy in federated scenarios, due to the *limited view* of local clients and *overlapping information* across multiple subgraphs.

Motivated by this, we first propose a **fe**derated gr**a**ph analy**t**ic framework, named FEAT, which enables arbitrary downstream common graph statistics while preserving individual privacy. FEAT leverages our proposed differentially private set union (DPSU) algorithm to aggregate the subgraph information, which ensures that the sensitive information is reported only once and the output global graph is protected under DP. Furthermore, we introduce an optimized framework based on our proposed degree-based partition algorithm, called FEAT+, which

improves the overall utility by leveraging the true local subgraphs. Finally, extensive experiments demonstrate that our FEAT and FEAT+ significantly outperform the baseline approach by approximately one and four orders of magnitude, respectively.

1.3 Thesis Structure

The structure of this thesis is as follows: Chapter 2 to Chapter 4 correspond to the three presented research topics. In Chapter 2, we demonstrate how cryptography can enable high utility in publishing differentially private degree distributions under node-local differential privacy. In Chapter 3, we introduce a crypto-assisted differentially private triangle counting system that achieves high-utility triangle counting comparable to a central model without requiring a trusted server, similar to a local model. In Chapter 4, we present a federated graph analytics framework that balances utility and privacy by integrating cryptography into differential privacy. Finally, Chapter 5 summarizes the thesis and discusses potential future research directions.

CHAPTER 2

Crypto-Assisted Differentially Private Degree Distribution

2.1 Introduction

Graph analysis has been receiving more and more attention on social networks, transportation, protein forecast, etc. However, directly publishing graph statistics may leak sensitive information about an individual [12]. Recently, many research works have studied the problem of publishing sensitive graph statistics under differential privacy (DP) [13, 14]. Compared with previous privacy models (e.g., k-anonymity, l-diversity, t-closeness), differential privacy can resist most private attacks and provide a provable privacy guarantee.

When DP is applied to graph analysis, there are two common variants of DP [26, 27]: Edge Differential Privacy [18–21] and Node Differential Privacy [3, 15–17]. Intuitively, Edge Differential Privacy guarantees that a query result does not significantly reveal sensitive information about a particular edge in a graph, while Node Differential Privacy protects the information about a node and all its adjacent edges. Obviously, Node Differential Privacy provides a much stronger privacy guarantee than Edge Differential Privacy. Existing works related to Node Differential Privacy are almost in the central (or global) model, where a trusted curator holds the entire graph data before data publishing. We refer to the above

two variants under a central server setting as Edge Central Differential Privacy (Edge-CDP) and Node Central Differential Privacy (Node-CDP), respectively. However, the assumption about a trusted server may not be practical in many applications (i.e., individual contact lists) due to security reasons, such as privacy leaks and breaches in recent years [28]. Local differential privacy (LDP) [29, 30] is a promising model that does not require a trusted server to collect user information. In LDP, each user perturbs its sensitive information by herself and sends perturbed messages to the untrusted server; hence it is difficult for the curator to infer sensitive information with high confidence. We refer to the above two variants of DP without a trusted server as Edge Local Differential Privacy (Edge-LDP) and Node Local Differential Privacy (Node-LDP), respectively.

Although there are many recent studies on publishing statistics under Edge-LDP [5, 31, 32], to the best of our knowledge, no existing work in literature attempts to investigate graph statistics release under Node-LDP. Basically, it is very challenging to publish graph statistics under Node-LDP due to the lack of global view and prior knowledge about the entire graph. Consider querying the node degree in a social graph, and if two graphs differ in one node, the results may differ at most $(n-1)$ edges in the worst case, where n is the number of users. Thus the sensitivity of Node Differential Privacy is $O(n)$ while that of Edge-DP is $O(1)$. Naively scaling the sensitivity of Edge-LDP for achieving Node-LDP suffers the prohibitive utility drop.

Graph projection [3, 15, 17] is the key technique to reduce the high sensitivity, but existing projections are only designed for the central model. When attempting to apply central graph projections into Node-LDP, it is difficult for each local user to project its neighboring information with a limited local view. In central models, with the global view, the server can determine optimal strategies of removing which edges or nodes to maximize the overall utility. However, in the local setting, each user can only see its own information but not other neighboring information. What's more, it is difficult for local users to obtain a graph projection parameter θ with high accuracy as they have little knowledge about the entire graph. In general, graph projection transforms a graph into a θ-bounded graph whose maximum degree is no more than θ. The parameter θ plays a vital role as it reduces the sensitivity from $O(n)$ to $O(\theta)$. If θ is too small, a large number of edges will be removed during the projection. If θ is too large, the sensitivity will become higher and more noise will be added during the protection.

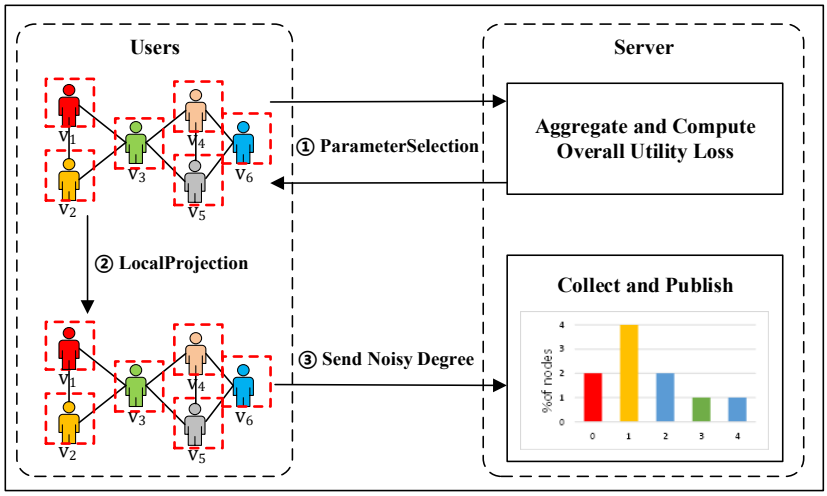


Figure 2.1. Framework of our methods

Graph projections in the central setting can easily opt for the desirable projection parameter θ with some prior knowledge of the whole graph, for instance, the maximum degree, the average degree, etc.; yet it is harder for local users to achieve it, since they have little prior knowledge about the entire graph.

In this paper, we introduce a novel local graph projection method for publishing the degree distribution under Node-LDP by addressing two main challenges: (1) How to obtain the graph projection parameter θ in the local setting; (2) How to execute the graph projection locally. The general framework is depicted in detail in Fig. 2.1, which includes three phases: (1) local users and server collaboratively select a projection parameter θ with minimum utility loss (Sec.2.4); (2) local users execute local graph projection based on selected parameters (Sec.2.5); (3) local users perturb individual information and send noisy degrees to the server.

First, to find the optimal projection parameter θ, we design a multiple-round protocol to find which parameter has the minimum utility loss. Specifically, for each round, each user calculates the potential utility loss with respect to a certain θ and sends to the server for computing the aggregated loss. The utility loss contains sensitive information since it is calculated based on each user's raw data. We design two methods to protect individual messages based on different privacy-enhancing techniques: pureLDP and crypto-assisted. The pureLDP method is a naive local graph projection method under Node-LDP. The obvious disadvantage is that multiple-round adding noise significantly deteriorates the utility. To improve it, we design a crypto-assisted parameter selection method that improves

the utility with cryptographic primitives. The key challenge is that aggregated utility loss is computed for the evaluation while individual utility loss can be protected. We first use the order-preserving encryption (OPE) scheme [33, 34] to encode individual information for comparing different utility loss values. Then, we mask the encrypted value with Secure Aggregation (SA) technique [35] to protect the order information of individual utility loss. The masks can be cancelled during the aggregation and the final aggregated utility loss is protected under OPE scheme.

Second, we propose two different local projection methods based on different granularity, including node-level method and edge-level method. In node-level method, each node is the minimal unit of a graph and correlations among neighboring users will be ignored coarsely. However, this approach loses too much neighboring information that significantly influences the overall utility (detailed analysis in Sec. 2.5.1). Then, we propose an improved approach, edge-level method, where each edge is the minimal unit that is more fine-granularity information. One main challenge is that privacy leakage may happen via communication messages among neighboring users. We represent this message as an operation vector and carefully design a randomized mechanism to perturb each bit of this vector while satisfying Node-LDP. As a result, it is difficult for neighboring users to distinguish whether the current node degree is larger than θ or smaller than θ.

Our contributions can be summarized as follows:

- We propose and study the problem of publishing the degree distribution under Node-LDP for the first time. We give a detailed description of the problem definition and conclude the research gap. We present an overview of publishing the degree distribution under Node-LDP.

- We design two methods to select the projection parameter θ in the local setting: pureLDP and crypto-assisted. Crypto-assisted method guarantees the security of individual utility loss with cryptographic primitives, which achieves a higher accuracy than the baseline pureLDP method.

- We design two local graph projection approaches based on different granularity: node-level and edge-level. The improved edge-level method preserves more information and provides better utility than the baseline node-level method.

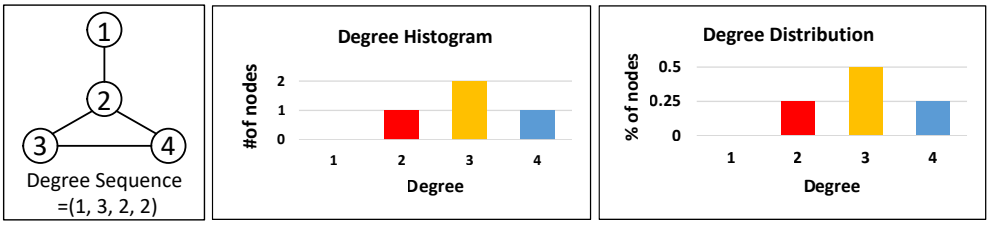


Figure 2.2. Example of publishing degree distribution

- Extensive experiments on real-world graph datasets validate the correctness of our theoretical analysis and the effectiveness of our proposed methods.

2.2 Problem Definition and Preliminaries

2.2.1 Problem Definition

In this paper, we consider an undirected graph with no additional attributes on nodes or edges. An input graph is defined as $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ is the set of nodes, where $|V| = n$, and $E \subseteq V \times V$ is the set of edges. For each user i, $B_i = \{b_{i1}, b_{i2}, ..., b_{in}\}$ is its adjacent bit vector, where $b_{ij} = 1$ if the edge $(v_i, v_j) \in E$ and $b_{ij} = 0$ otherwise. The number of adjacent edges for one node i is the node degree d_i, namely, $d_i = \sum_{j=1}^{n} b_{ij}$. The server collects a perturbed degree sequence $seq = \{d_1, d_2, ..., d_n\}$ from each local user and publishes the degree histogram $hist(G)$. The degree distribution $dist(G)$ can be easily obtained from $hist(G)$ by counting each degree frequency. Fig. 2.2 shows an example of degree sequence, degree histogram, and degree distribution, respectively.

We use two common measures to assess the accuracy of our algorithms. First, we use the mean squared error (MSE) [36] to estimate the error between noisy histogram $hist(G)'$ and original histogram $hist(G)$. Generally, the MSE can be computed as $\text{MSE}(hist(G), hist(G)') = \frac{1}{n}\sum_{i=1}^{n}(hist(G)_i - hist(G)_i{}')^2$, where n is the number of users in a graph. Also, we compute the mean absolute error (MAE) [37] which can be represented by $\text{MAE}(hist(G), hist(G)') = \frac{1}{n}\sum_{i=1}^{n}|hist(G)_i - hist(G)_i{}')|$.

2.2.2 Preliminaries

Since the trusted third party is impractical, LDP has become the de facto standard of privacy protection to protect individual information. As a graph consists of nodes and edges, there are two definitions when LDP is applied to either of them: edge local differential privacy (Edge-LDP) in Definition 1 and node local differential privacy (Node-LDP) in Definition 2.

Definition 1 (Edge-LDP) *A random algorithm M satisfies ϵ-Edge-LDP, iff for any $i \in [n]$, two adjacent bit vectors B_i and B_i' that differ only one bit, and any output $y \in range(M)$,*

$$Pr[M(B_i) = y] \leq e^{\epsilon} Pr[M(B_i') = y]$$

Definition 2 (Node-LDP) *A random algorithm M satisfies ε-Node-LDP, iff for any $i \in [n]$, two adjacent bit vectors B_i and B_i' that differ at most n bits, and any output $y \in range(M)$,*

$$Pr[M(B_i) = y] \leq e^{\varepsilon} Pr[M(B_i') = y]$$

Node-LDP is clearly a much stronger privacy guarantee than Edge-LDP since it requires hiding the existence of each node along with its incident edges. To our knowledge, however, there are few research works that release graph statistics under Node-LDP. Although Zhang *et al.* [38] consider Node-DP in the local setting where each node represents a software component and an edge represents control flow between components, the directed graphs on the control-flow behavior of different users are mutually independent. We consider a totally different setting where each node represents a user and each edge represents the correlation between neighboring users.

There are two kinds of DP, namely, *bounded* DP and *unbounded* DP [14, 39]. In a bounded DP, two neighboring datasets D, D' have the same size n and $D\prime$ is obtained from D by changing or replacing one element. In unbounded DP, $D\prime$ can be derived from D by deleting or adding one element. Here, we use the bounded DP to publish the degree distribution. That is to say, the size of each adjacent bit vector is equal to n, where n is the number of users. Node-LDP satisfies the post-processing property (Theorem 1) and the composition property (Theorem 2) [13].

Theorem 1 (Post-Processing) *If a randomized algorithm R satisfies ε-DP, then for an arbitrary randomized algorithm S, $S \circ R$ also satisfies ε-DP.*

Theorem 2 (Composition Property) *$\forall \varepsilon \geq 0, k \in N$, the family of ε-DP mechanism satisfies $t\varepsilon - DP$ under t-fold adaptive composition.*

To satisfy DP, one way to add some noise into the query result. In the Laplace mechanism (Theorem 3) [13, 14], given the privacy budget ε and sensitivity $\triangle$, one publishes the result after adding Lap($\frac{\triangle}{\epsilon}$) noise.

Theorem 3 (Laplace Mechanism) *For any function f, the Laplace mechanism $A(D) = f(D) + Lap\ (\frac{\triangle f}{\varepsilon})$ satisfies ε-DP.*

2.3 Overview of Proposed Methods

We aim to design a method for publishing the degree distribution that approximates the original distribution as possible while satisfying the strict Node-LDP. Our proposed methods support the following functions: 1) obtaining the graph projection parameter θ in the local setting; 2) conducting the graph projection locally; 3) publishing the degree distribution under Node-LDP.

Algorithm 1 Publishing the degree distribution

Input: Adjacent bit vectors $\{B_1, ..., B_n\}$,
privacy budget $\varepsilon_1, \varepsilon_2, \varepsilon_3$
Output: A noisy degree distribution $dist(G)'$

1: $\theta \leftarrow$ `SelectParameter`$(\{B_1, ..., B_n\}, \varepsilon_1)$ // Sec. 2.4
/* User side. */
2: **for** each user $i \in \{1, 2, ..., n\}$ **do**
3: $\hat{d}_i \leftarrow$ `LocalProjection`$(B_i, \theta, \varepsilon_2)$ // Sec. 2.5
4: $d'_i \leftarrow \hat{d}_i$+Lap$(\frac{2\theta}{\varepsilon_3})$
5: User i sends d'_i to server
6: **end for**/* Curator side. */
7: Curator collects all noisy degree d'_i
8: **return** $dist(G)'$

We provide an overview of our solutions in Algorithm 1. First, a private parameter selection method is designed to select the projection parameter with minimum utility loss in the local setting (Section 2.4). The curator collects individual

utility loss from local users and evaluates each candidate projection parameter k by computing the aggregated utility loss. To protect sensitive individual utility loss during communications, we first propose one naive approach, pureLDP parameter selection, which adds noise into individual utility loss. However, this method adds too much noise to destroy the order information of different aggregated utility loss, significantly influencing the selection accuracy. Then, we propose an improved crypto-assisted parameter selection method using cryptographic primitives. Specifically, the individual utility loss is encrypted by order-preserving encryption (OPE) [40] scheme where the numerical order in the plaintext domain will be preserved in the ciphertext domain. To prevent leaking the order information of individual utility loss while preserving the order of the aggregated utility loss, we add one mask into encrypted values with Secure Aggregation technique [35]. The added masks are cancelled after the aggregation and the final aggregated utility loss is protected under OPE scheme.

Second, as soon as the projection parameter is decided, each user can execute the local projection (Section 2.5). Compared with the Node-CDP, it is more difficult for each user to execute the local projection due to the limited local view of the entire graph. We first give a baseline node-level approach that is motivated by graph projection [5] with Edge-LDP. In node-level local projection, the node is the minimal unit and correlations among users are ignored. It is easy to deploy but lose much information that significantly influences the utility. Then we design an improved edge-level local projection where each edge is the minimal unit during the projection. The key challenge is that information leakage may happen via mutual edges among neighboring users. For example, neighboring users may know that the current degree is larger than or less than θ during the local projection. We represent this sensitive message as each bit in an operation vector and design a randomized mechanism to perturb each bit. Thus neighboring users cannot distinguish the current node degree whether larger than θ or smaller than θ.

Third, after finishing the local projection, each user perturbs its projected degree using the Laplace mechanism. Here, the sensitivity is 2θ since any change of one edge will make an effect on two node degrees. Then, they send the noisy degree to the server. The curator collects the degree sequence and publishes the degree histogram and degree distribution.

2.4 Projection Parameter Selection

2.4.1 PureLDP Selection

Intuitively, the server can help local users select the parameter with the minimum utility loss from the candidate set {1,2,...,K} through multiple-round communications. We design a utility loss function to evaluate each candidate parameter k. Our utility loss function has two parts, as shown in Equation 2.1, which includes projection utility loss during the local projection and publishing utility loss from added Laplace noise. The publishing utility loss E_D is usually a constant value. For example, the publishing utility loss of degree distribution is equal to the variance, namely, $E_D = n.2(\frac{2k}{\varepsilon_3})^2 = \frac{8nk^2}{\varepsilon_3^2}$. The projection utility loss E_P is aggregated by all individual projection utility losss, i.e., $E_P = \sum_{i=1}^{n}\{d_i - k | v_i \in V, d_i > k\}$. But directly collecting each individual utility loss from local users may reveal personal information. In baseline method, we use the Laplace mechanism to provide the privacy guarantee and its sensitivity is $(n-1-k)$ in Node-LDP, as shown in Lemma 1.

$$F(k) = E_P + E_D, \tag{2.1}$$

$$E_P = \sum_{i=1}^{n} |\{d_i - k | v_i \in V, d_i > k\}|$$

$$E_D = n.2(\frac{2k}{\varepsilon_3})^2 = \frac{8nk^2}{\varepsilon_3^2}$$

Lemma 1 *For any projection loss $|d_i - \hat{d}_i|$ and $|d_i - \hat{d}_i|'$, we have*

$$||d_i - \hat{d}_i| - |d_i - \hat{d}_i|'|_1 \leq (n-1-k)$$

Proof of Lemma 1: Given the graph projection parameter is k, for each node degree d_i, if $d_i \leq k$, projected node degree $\hat{d}_i$ will remain the original value, namely, $\hat{d}_i = d_i$; otherwise, $\hat{d}_i = k$. Thus, we have

$$|d_i - \hat{d}_i| = \begin{cases} d_i - \theta, & d_i > k \\ 0, & d_i \leq k \end{cases}$$

Since the maximum node degree is $(n-1)$, the projection loss value is bounded by $(n-1-k)$.

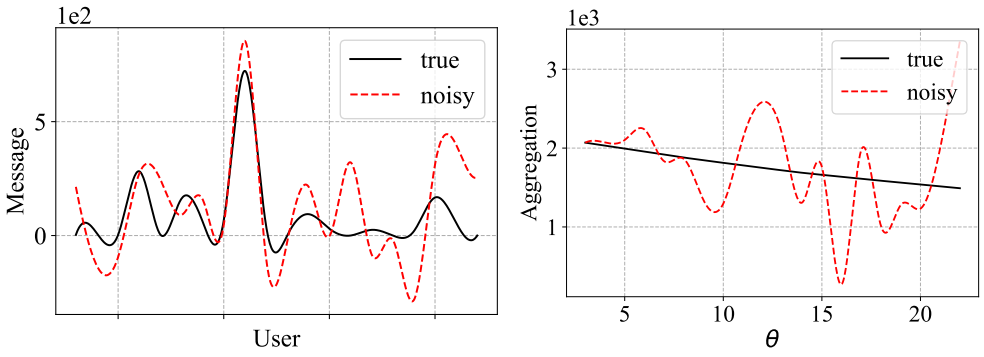
Algorithm 2 PureLDP parameter selection

Input: Adjacent bit vectors $\{B_1, ..., B_n\}$, privacy budget ε_1
Output: Projection parameter θ

for each integer $k \in \{1, 2, ..., K\}$ **do**
  /* User side. */
  for each user $i \in \{1, 2, ..., n\}$ **do**
    $\hat{d}_i \leftarrow$ `LocalProjection`(B_i, k) // Sec. 2.5
    $d_i \leftarrow \sum_{j=1}^{n} b_{i,j}$
    $E_{P_{k,i}} \leftarrow |d_i - \hat{d}_i| + \text{Lap}(\frac{n-1-k}{\varepsilon_1/K})$
    User i sends $E_{P_{k,i}}$ to server
  end for
  /* Curator side. */
  $E_{P_k} \leftarrow \sum_{i=1}^{n} E_{P_{k,i}}$
  $\theta \leftarrow k$ when $(E_{P_k} + E_D)$ is minimum
end for
return θ

Algorithm. Algorithm 2 presents the formal description of pureLDP parameter selection. It takes as input a graph G that is represented as bit vectors $\{B_1, ..., B_n\}$, the privacy budget ε_1, and the size of candidate parameter K. For each candidate parameter k, the original graph is first projected to k-bounded graph using the local graph projection method (in Section 2.5). Then, each user computes the projection utility loss and adds the Laplace noise into individual utility loss with the sensitivity $(n - 1 - k)$. After collecting all noisy individual projection utility loss, the server computes the sum of aggregated projection utility loss and publishing utility loss. Finally, the parameter θ is selected when the overall utility loss is the minimum and server sends this θ to each local user.

Limitation. Much noise is added into the true individual utility loss, which significantly destroys the order information of aggregated utility loss. To capture the impact of adding Laplace noise on the accuracy of pureLDP parameter selection method, we execute experiments on Wikipedia vote network from SNAP [41]. As shown in Fig. 2.3, the left figure presents the impact of added noise on the order of individual utility loss when $\theta = 20$, and the difference between true and noisy utility loss is up to 85%. The right one shows the influence on the order of aggregated utility loss under various θ and the difference is up to 90%. Finally, the accuracy of selecting projection parameter θ is influenced significantly.



(a) Under θ=20 (b) Under various θ

Figure 2.3. The impact of added noise on order information

2.4.2 Crypto-assisted Selection

Our goal is that each individual projection utility loss can be protected when the order of aggregated utility loss is preserved. Order-preserving encryption (OPE) scheme [42, 43] can achieve this idea that the i-th data in the plaintext domain is transformed to the i-th data in the ciphertext domain, so the numerical order among plaintexts is preserved among ciphertexts. Thus when individual utility loss are encrypted by OPE scheme, the numerical order of individual utility loss can be preserved and the order of aggregated utility loss is also preserved. But there is the other problem that the order of aggregated utility loss is preserved while the order of individual utility loss is revealed to server. Next, we use the secure aggregation [35] to mask encoded individual utility loss, and these masks can be cancelled during the aggregation.

OPE Schemes. There are many existing works related to OPE scheme. For example, Popa *et al*. [40] proposed an interactive OPE scheme between the client and the server, which allows the encrypted state to update over time as the new values are inserted. The server organizes the encrypted values by maintaining a binary search tree, namely, OPE tree. To reduce the high cost of the encryption, Kerschbaum *et al*. [44] designed a more efficient OPE scheme that uses a dictionary to keep the state and thus does not need to store too much data. Roche *et al*. [45] proposed an alternative approach to optimize the heavy insertion of OPE schemes. It is very efficient at insertion and has a lower communication cost, but it provides only a partial order. Here, we choose a linear OPE scheme [46] to encode individual utility loss since it can be directly extended for the local

Algorithm 3 Crypto-assisted parameter selection

Input: Adjacent bit vectors $\{B_1, ...B_n\}$,
security parameters a, b
Output: Projection parameter θ

for each integer $k \in \{1, 2, ..., K\}$ **do**
  /* User side. */
  for each user $i \in \{1, 2, ..., n\}$ **do**
    $\hat{d}_i \leftarrow \texttt{LocalProjection}(B_i, k)$ // Sec. 2.5
    $d_i \leftarrow \sum_{j=1}^{n} b_{i,j}$
    $noise \leftarrow randint(0, a-1)$
    r $\leftarrow \texttt{PRG}(seed)$
    $mask = \sum_{j=i+1}^{n-1} r_{i,j} - \sum_{j=1}^{i-1} r_{i,j}$
    $Enc_{T_{k,i}} \leftarrow a * |d_i - \hat{d}_i| + b + noise + mask$
    User i sends $Enc_{T_{k,i}}$ to server
  end for
  /* Curator side. */
  $Enc_{T_k} \leftarrow \sum_{i=1}^{n} Enc_{T_{k,i}}$
  $\theta \leftarrow k$ when $(Enc_{T_k} + E_D)$ is minimum
end for
return θ

setting.

Discussion. It is important to note that order information can be sensitive and may compromise individual privacy. While the utility loss can be mitigated using order-preserving encryption techniques, they reveal ordering information to potential attackers. These attackers might then estimate the plaintext distribution based on the preserved order in the ciphertexts. To counteract this, we employ secure aggregation techniques as outlined in [35] to mask individual utility loss. It allows for the comparison of aggregated utility loss under different parameters without revealing individual data.

Secure Aggregation [35]. Consider a curator with n users where user $i \in [n]$ has its private local vector x_i. The objective of server is to compute the sum of models $\sum_{i \in n} x_i$ without getting any other information on private local data. Suppose each pair of users $(i, j), i < j$ agree on some random vector $s_{i,j}$. If user i adds $s_{i,j}$ to x_i and j subtracts it from x_j, then the mask $s_{i,j}$ will be canceled when their vectors are added, but their true inputs will be concealed without revealing. Formally, each masked value can be computed:

$$y_i = x_i + \sum_{j \in n: i<j} s_{i,j} - \sum_{j \in n: i>j} s_{i,j} \text{ (mod R)}$$

Then server collects y_i and computes:

$$\begin{aligned} z &= \sum_{i \in n} y_i \\ &= \sum_{i \in n} \left(x_i + \sum_{j \in n: i<j} s_{i,j} - \sum_{j \in n: i>j} s_{i,j} \right) \\ &= \sum_{i \in n} x_i \text{ (mod R)} \end{aligned}$$

Based on above two cryptographic primitives, we propose a crypto-assisted parameter selection method, as presented in Algorithm 3. First, we use the linear OPE scheme [46] to encode individual utility loss, namely, $f(x) = a * |d_i - \hat{d}_i| + b + noise$. Here security parameters a and b are kept secret from the server and the noise is randomly selected from $[0, a-1]$. Second, to hide the order of individual utility loss, we add one mask into the encoded values of the OPE scheme using SA. For each user i, it and the rest other $n-1$ users agree on common seeds. Then local users generate the random numbers r with the common seeds by the pseudorandom generator (PRG) [47] and add into the individual utility loss. Finally, the server collects all encrypted individual projection utility loss and computes the aggregated utility loss. The added masks can be cancelled with each other after aggregation and any information about individuals cannot be leaked. The final aggregated utility loss is still protected under OPE scheme.

2.5 Local Projection Methods

2.5.1 Node-level Local Projection

Local scenarios make projection operations challenging, since no party owns the entire graph and local users cannot easily add or remove any edges. We propose a node-level projection method where each node is the minimal unit. As presented in Algorithm 4, it inputs an adjacent bit vector and projection parameter θ. Each local user first counts the number of neighboring edges. If node degree d_i is larger than θ, projected degree $\hat{d}_i$ will be directly set as θ; otherwise, $\hat{d}_i$ remains the original value.

Algorithm 4 Node-level Local Projection

Input: Adjacent bit vector B_i=$\{b_{i1}, ..., b_{in}\}$,
projection parameter θ
Output: θ-bounded node degree $\hat{d}_i$

1: $d_i \leftarrow \sum_{j=1}^{n} b_{i,j}$
2: **if** $d_i > \theta$ **then**
3: $\quad \hat{d}_i = \theta$
4: **else**
5: $\quad \hat{d}_i \leftarrow d_i$
6: **end if**
7: **return** $\hat{d}_i$

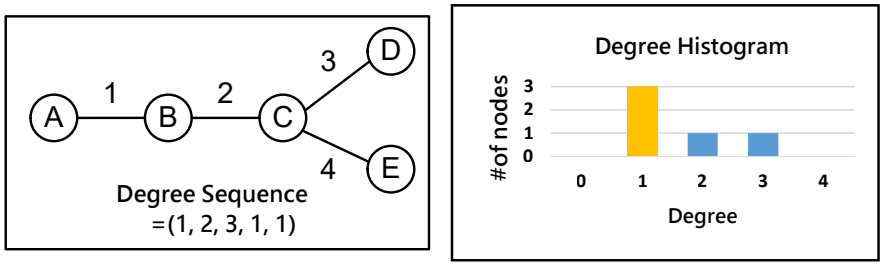


Figure 2.4. Example of degree histogram

Limitations. Although node-level projection is easy to implement, it omits correlations among neighboring users coarsely, influencing the accuracy significantly. For example, we have a simple graph with five nodes and some edges, as shown in Fig. 2.4. The original histogram can be represented as H_1 = (0, 3, 1, 1, 0). Assume that the projection parameter θ = 1, the projected degree sequence becomes Seq_1 = (1, 1, 1, 1, 1) and the current histogram is H_2= (0, 5, 0, 0, 0) after node-level projection. We can compute the projection loss: MSE(H_1, H_2)= $\frac{6}{5}$. If correlations are considered, any change in mutual edges will update two neighboring adjacent bit vectors. For example, if edge 2 and 3 are removed to bound all degrees, the degree sequence will become Seq_2= (1, 1, 1, 0, 1) and the degree histogram will be H_3= (1, 4, 0, 0, 0). The projection loss can be computed: MSE(H_1, H_3)= $\frac{4}{5}$. Obviously, node-level method loses more edge information, which significantly affects overall utility. What's more, the characteristic of degree distribution is destroyed by node-level projection. For instance, it is not easy to find a real-world graph that is represented by the sequence Seq_1.

Generally, we assume that the number of users in a graph is n, projection

parameter is θ, and original degree histogram is H_1=$(h_1, h_2, ..., h_n)$. If there are m nodes with degree larger than θ, we can get the projected histogram H_2=$(h_1, h_2, ..., h_\theta + m, 0, ..., 0)$ using node-level projection. On the other hand, if mutual edge information is considered during the projection, the new histogram will become H_3=$(h_1 + t_1, h_2 + t_2, ..., h_\theta + t_m, 0, ..., 0)$, where $t_i \in \mathbb{Z}$ $(i \in [1, m])$ is the variation of each bin in the histogram. We refer to this method as edge-level projection method. One mutual edge connects two nodes and there are two cases during the edge-level local projection: (1) two node degrees are both over θ. The final histogram of edge-level is same with that of node-level. (2) one node degree is larger than θ and the other one is smaller than θ. The change from the former one is same with the first case. The influence from the latter can be cancelled finally. Thus, we can easily achieve $m = t_1 + t_2 + ... + t_m$. Then we can compute their projection loss, namely, $MSE(H_1, H_2)$=$\frac{m^2}{n} = \frac{1}{n}(t_1 + t_2 + ... + t_m)^2$ and $MSE(H_1, H_3)$=$\frac{1}{n}(t_1^2 + t_2^2 + ... + t_m^2)$. Since $(t_1 + t_2 + ... + t_m)^2 \geq (t_1^2 + t_2^2 + ... + t_m^2)$, we can get $MSE(H_1, H_2) \geq MSE(H_1, H_3)$. Therefore, the result of node-level projection method is not desirable.

2.5.2 Edge-level Local Projection

Based on above discussions, if we consider the correlation among users, more edge information will be reserved after the projection. However, unlike Node-CDP where the trusted server can decide the optimal strategies of removing which edges or nodes to maximize the overall utility, it is difficult for a local user to update the mutual edges. The key challenge is that any change in the edges may leak individual sensitive information via mutual edges. For example, if one node degree d_i is larger than θ, it will delete some edges. At the same time, this user i will send messages to its neighboring users to update their adjacent bit vectors. The message itself reveals that the current node degree may be larger than θ. We design an edge-level method to protect this sensitive message.

Security Assumptions. We assume that 1) the communication between neighboring users is perfectly anonymous, that's to say, the third party (e.g., server or third user) cannot know the communication exists or not; 2) the user does not reveal sensitive neighboring information to other users, for example, B will not tell C that A is one of its friends or not. Based on above assumptions, one edge is only visible to two neighboring users and other edges are in a data-

Algorithm 5 Edge-level Local Projection

Input: Adjacent bit vector B_i=$\{b_{i1}, ..., b_{in}\}$,
projection parameter θ, privacy budget ε_2
Output: θ-bounded node degree $\hat{d}_i$

R_i=[0] $\times$ $\hat{d}_i$ // Record which edges will be deleted
$d_i \leftarrow \sum_{j=1}^{n} b_{i,j}$
if $d_i \geq \theta$ **then**
  Randomly select $(d_i - \theta)$ bits from R_i and set '1'
  for each $\mathrm{r}_{ij} \in \mathrm{R}_i$ **do**

$$r'_{ij} = \begin{cases} r_{ij} & w.p.\ \frac{\theta}{d_i} \\ 1 - r_{ij} & w.p.\ \frac{d_i - \theta}{d_i} \end{cases}$$

  end for
else
  for each $\mathrm{r}_{ij} \in \mathrm{R}_i$ **do**
    if $\frac{d_i - \theta}{d_i} \leq \frac{e^{\varepsilon_2} - 1}{e^{\varepsilon_2} - e^{-\varepsilon_2}}$ **then**

$$r'_{ij} = \begin{cases} r_{ij} & w.p.\ 1 - \frac{e^{-\varepsilon_2}(d_i - \theta)}{d_i} \\ 1 - r_{ij} & w.p.\ \frac{e^{-\varepsilon_2}(d_i - \theta)}{d_i} \end{cases}$$

    else

$$r'_{ij} = \begin{cases} r_{ij} & w.p.\ \frac{e^{\varepsilon_2}\theta}{d_i} \\ 1 - r_{ij} & w.p.\ \frac{d_i - e^{\varepsilon_2}\theta}{d_i} \end{cases}$$

    end if
  end for
end if
for each $\mathrm{r}_{ij} \in \mathrm{R}_i$ **do**
  if $r_{ij} = 1$ **then**
    $b_{ij} = 0$ and $b_{ji} = 0$
  end if
end for
return $\hat{d}_i$

invisible way. Thus, the communication message is just one bit and the sensitivity becomes $O(1)$.

Algorithm. We propose the edge-level projection method to improve node-level method and the edge is the minimal unit during the projection, as shown in Algorithm 5. Privacy leakage may occur when the local projection is performed since the sensitive messages are sent to neighboring users. We represent this

Table 2.1. Randomized projection vector

Pr	0	1
$<\theta$	1-x	x
$\geq \theta$	1-p	p

message as an operation vector $R_i = \{r_{i1}, ..., r_{id_i}\}$, and the size of R_i is d_i. If $r_{ij} = 1$, the corresponding edges in two neighbor lists will be removed; otherwise, they remain the same. We carefully perturb each bit of the operation vector to make two cases indistinguishable: node degree d_i is larger than θ or d_i is smaller than θ. Ideally, we want to flip each bit of the projection bit vector with probability in Table 2.1, where $p = \frac{d_i - \theta}{d_i}$ and $x = 0$. Obviously, when $x = 0$, our randomized mechanism cannot satisfy the Node-LDP. To satisfy the Node-LDP, we have the following inequation:

$$\begin{cases} e^{-\varepsilon_2} \leq \frac{x}{p} \leq e^{\varepsilon_2} \\ e^{-\varepsilon_2} \leq \frac{1-x}{1-p} \leq e^{\varepsilon_2} \end{cases}$$

Then, we can bound the scope of x as follows:

$$\begin{cases} pe^{-\varepsilon_2} \leq x \leq pe^{\varepsilon_2} \\ (p-1)e^{\varepsilon_2} + 1 \leq x \leq (p-1)e^{-\varepsilon_2} + 1 \end{cases}$$

When $d_i < \theta$, we want to preserve more edges during projection, that is to say, the number of '1' in projection bit vector is as small as possible. Thus we have

$$x = \begin{cases} pe^{-\varepsilon_2}, & pe^{-\varepsilon_2} \geq (p-1)e^{\varepsilon_2} + 1 \\ (p-1)e^{\varepsilon_2} + 1, & pe^{-\varepsilon_2} < (p-1)e^{\varepsilon_2} + 1 \end{cases}$$

After randomizing the bits of the projection bit vector, each user updates the adjacent bit vector according to randomized bit vector (Line 19). Then, local users count the number of edges and obtain the bounded degree $\hat{d}_i$.

2.6 Analysis and Discussions

Privacy Budget Allocation. As shown in Algorithm 1, there are three kinds of privacy budgets. Our goal is to find the optimal privacy allocation scheme with the best utility. Without loss of generality, we assume that the overall privacy

Table 2.2. optimal privacy allocation scheme α

ε	Ca-HepPh	Cit-HepPh	Twitter	Com-DBLP
0.5	0.895	0.927	0.945	0.945
1	0.944	0.937	0.949	0.947
1.5	0.901	0.940	0.944	0.948
2	0.948	0.946	0.947	0.937
2.5	0.944	0.922	0.948	0.943
3	0.944	0.948	0.941	0.940

Table 2.3. optimal parameter θ

ε	Ca-HepPh	Cit-HepPh	Twitter	Com-DBLP
0.5	3	4	18	13
1	9	7	31	17
1.5	15	10	41	20
2	19	12	43	23
2.5	24	15	45	25
3	26	18	48	27

budget is ε, $\varepsilon_3 = \alpha\varepsilon$, and $\varepsilon_1+\varepsilon_2 = (1-\alpha)\varepsilon$. For inner privacy budget allocation of local graph projection, we distribute the same privacy budget for the projection parameter selection and executing the local graph projection, namely, $\varepsilon_1 = \varepsilon_2$. We find the optimal α with the least utility loss by conducting many experiments for different cases, as shown in Table 2.2. And we use the optimal α for each case in the next experiments.

Selection of Parameter K. In Algorithm 2 and Algorithm 3, the parameter K, namely, the size of the candidate pool, plays a significant role in the trade-off between utility and privacy. When the size K is larger, more noise will be added by the pureLDP parameter selection and time overhead becomes higher. Similarly, the running time of crypto-assisted selection method will be higher. But if the K becomes smaller, the optimal projection parameter α is not covered possibly. We conduct extensive experiments and find the optimal parameter α for each case, as shown in Table 2.3. In our paper, we use $K = 50$ that is ample to cover the optimal parameter α of different cases.

Time Complexity. As shown in Table 2.4, we conclude the running time complexity of different combinations theoretically, $|V|$ and $|E|$ represent the number of nodes and edges respectively. Node-level local projection method transforms each node degree into θ-bounded degree directly, which takes time $O(|V|)$. In con-

trast, edge-level local projection method needs to traverse each edge for each node, resulting an $O(|V|.|E|)$ running time. PureLDP parameter selection method selects the optimal parameter θ from K candidates and for each candidate k, each user has to compute the projection loss, which takes time at most $O(K.|V|)$. By comparison, for each candidate parameter k of crypto-assisted selection method, each user has to communicate with the other $(|V|-1)$ users to determine the seed, resulting an $O(K.|V|^2)$ running time overhead.

Table 2.4. running time complexity

	pureLDP	crypto-assisted
Node-level	$O(\|V\| + K.\|V\|)$	$O(\|V\| + K.\|V\|^2)$
Edge-level	$O(\|V\|.\|E\| + K.\|V\|)$	$O(\|V\|.\|E\| + K.\|V\|^2)$

Security Analysis. Publishing the degree distribution in Algorithm 1 is under the following privacy guarantee.

Lemma 2 *Publishing the degree distribution satisfies* $(\varepsilon_1/K+\varepsilon_2+\varepsilon_3)$*-Node-LDP.*

Proof of Lemma 2: In Algorithm 1, SelectParameter(.) (Line 1) uses the Laplace with privacy budget ε_1/K, K is the number of candidate parameters. Executing the local projection (Line 3) uses our proposed mechanism and satisfies Node-LDP for ε_2. And publishing the distribution with Laplace Mechanism using ε_3. According to the post-processing theorem and composition property, Algorithm 1 satisfies $(\varepsilon_1/K + \varepsilon_2 + \varepsilon_3)$-Node-LDP.

2.7 Experimental Evaluation

In this section, we would like to answer the following questions:

- What is the tradeoff between utility and privacy of our proposed methods?
- What are results of different privacy budget allocation schemes?
- How much time do our proposed algorithms take?

2.7.1 Datasets and Setting

Our experiments run in python on a server with Intel Core i9-10920X CPU, 256GB RAM running Ubuntu 18.04 LTS. We use four real-world graph datasets

from SNAP [41], which are also used in [3, 36]. And we preprocess all graph datasets to be undirected and symmetric graphs. Table 2.5 presents more details about every graph dataset, including the number of nodes $|V|$, the number of edges $|E|$, and the number of edges after preprocessing $|E'|$ after preprocessing. In all experiments, we vary the privacy budget ε from 0.5 to 3. By default, we set hyper-parameter K=50 as we discussed above. All of our experimental results are the average values computed from 20 runs. We use 'PureLDP', CryptoAssisted', 'NodeProj' and 'EdgeProj' to represent pureLDP parameter selection, crypto-assisted parameter selection, node-level local graph projection and edge-level local graph projection respectively. Thus we have four different combinations to publish the degree distribution.

2.7.2 Relation between ε and MSE, MAE

As shown in Fig. 2.5, the utility of each combination method becomes better as the privacy budget ε increases. We can find that 'CryptoAssisted+EdgeProj' method always performs the best in most cases, while the results of 'PureLDP+ NodeProj' method are always the worst. To be specific, the MSE of 'CryptoAssisted+EdgeProj' method is less than that of 'PureLDP+NodeProj' by up to 87.2% on Twitter when $\varepsilon = 2.5$. The MAE of 'CryptoAssisted+NodeProj' method is larger than that of 'CryptoAssisted+EdgeProj' method by up to 66.4% in Twitter when $\varepsilon = 3$. The reason that 'CryptoAssisted+EdgeProj' method sometimes performs not the best in terms of MAE when $\varepsilon = 0.5$ is because our utility loss function uses the MSE as the evaluation metric, which makes a little influence on results of MAE, particularly when ε is very small. The results of pureLDP parameter projection are always worse than that of crypto-assisted parameter projection since the latter protects individual utility loss while preserving the order information of the aggregated utility loss accurately. Also, due to more information is preserved, edge-level local projection method performs much

Table 2.5. details of graph datasets

Graph	$\|V\|$	$\|E\|$	$\|E\|'$
Ca-HepPh	12,008	118,521	474,020
Cit-HepPh	34,546	421,578	843,156
Twitter	81,306	1,768,149	3,536,298
Com-DBLP	317,080	1,049,866	2,099,732

better than node-level local projection method. Overall, our proposed 'CryptoAssisted+EdgeProj' method improves our baseline 'PureLDP+NodeProj' approach for publishing the degree distribution under Node-LDP.

2.7.3 Impact of privacy budget allocation

To further estimate the influence of the privacy allocation scheme on the overall utility, we compare the best α with other three constant α, including 0.3, 0.6, and 0.9. We present the MSE results of different α on different graph datasets in Fig. 2.6. We can observe that the best α owns the lowest MSE against the other allocation schemes in most cases. On the other hand, with the increase of the overall privacy budget ε, the MSE value is decreasing. Thus most of privacy budget can be allocated to the final publishing the degree distribution, which is roughly consistent with our best α in Table 2.2, namely, ε_3 for publishing degree distribution is approximately equal to the overall privacy budget ε.

2.7.4 Analysis on running time

Finally, we compare the running time overhead of our proposed methods, as shown in Fig. 2.7. We can see that the running time of 'CryptoAssisted+EdgeProj' method is much larger than that of 'PureLDP+NodeProj' method. This is mainly because edge-level projection method needs to traverse each edge of every node and crypto-assisted parameter selection method has n users to communicate in pairs, which is in line with our theoretical analysis in Section 2.6. The difference between 'CryptoAssisted+EdgeProj' method and 'PureLDP+NodeProj' method is larger on Twitter. This is because Twitter has more edges than other graphs, as described in Table 2.5, which results in higher computation and communication overhead. In addition, incorporating cryptographic tools into Differential Privacy (DP) increases running time. For instance, the running time of 'CryptoAssisted+NodeProj' is greater than that of 'PureLDP+NodeProj'. The running time of 'PureLDP+EdgeProj' is shorter than 'CryptoAssisted+EdgeProj'.

2.8 Related Work

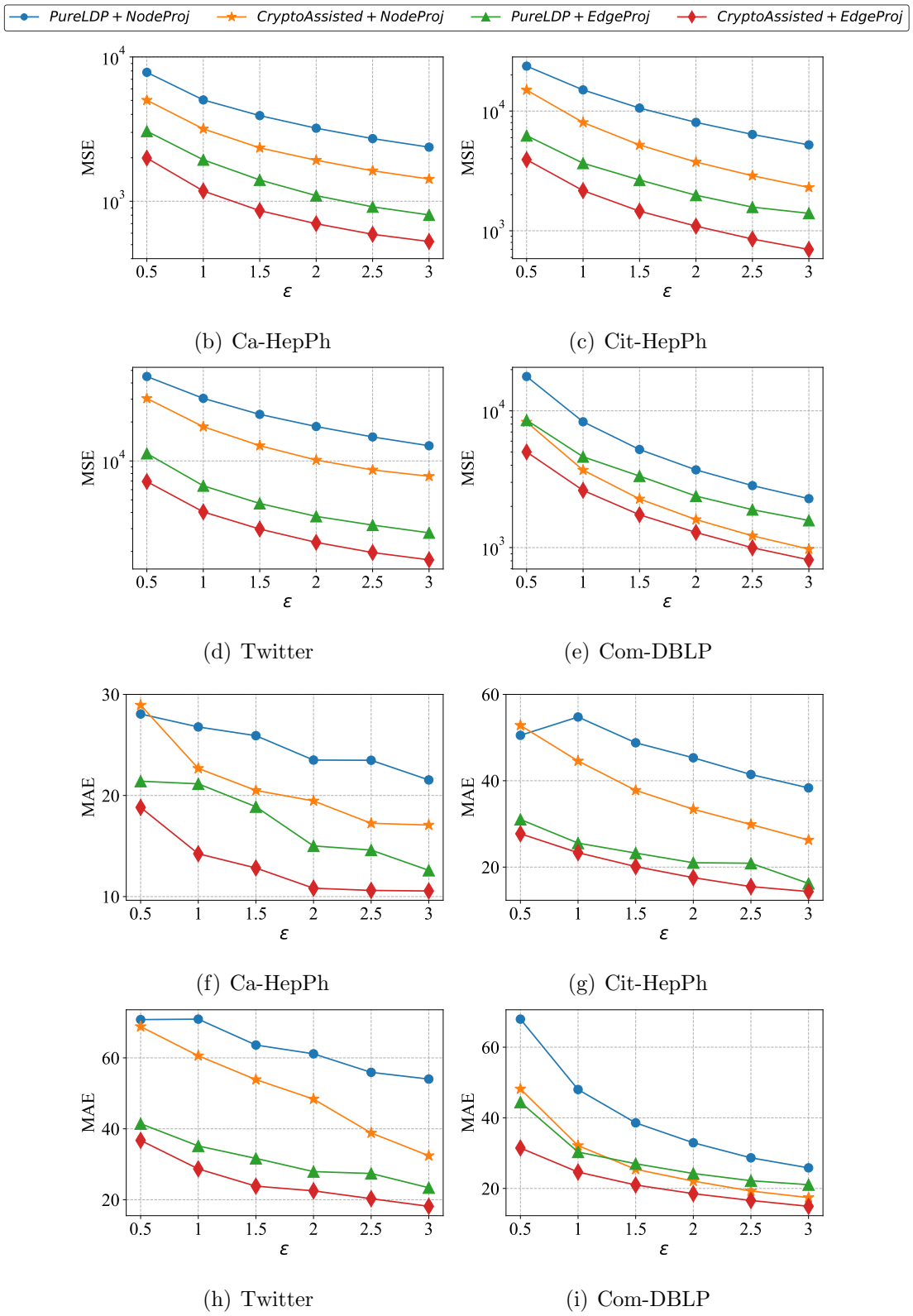
There are many existing works related to Node-CDP and Edge-LDP.

Node-CDP. There have been many prior research works related to Node differential privacy (Node-DP). For example, a handful of graph algorithms [3,15–17] have been designed for publishing the degree distribution by proposing different graph projection methods. For instance, the truncation method [15] removes all nodes with the degree over θ. Edge-removal approach [17] traverses all edges in an arbitrary order and removes each edge connected to a node with a degree more than θ. Edge-addition method [3] traverses the edges in a stable order and inserts each edge correlated to node with degree over θ. However, the existing projection methods are only designed for Node-CDP and are not viable in Node-LDP.

Edge-LDP. Since there is no need for a trusted server and a large amount of valuable information resides in a decentralized social network, LDP is becoming increasingly popular in privacy protection of graph analysis. Existing works focus on various graph statistics, such as degree distribution (or histogram) [36], subgraph counting (e.g., k-clique, k-star, k-triangle) [5,23], synthetic graph generation [32,48], publishing attributed graph [31,49], etc. For instance, Ye *et al.* [36] propose a LDP-enabled graph metric estimation framework for general graph analysis. In [5], subgraph counting is protected locally by a more sophisticated algorithm that uses an additional round of interaction between individuals and server. To strike a balance between noise added to satisfy LDP and information loss from a coarser granularity, Qin *et al.* [32] design a novel multi-phase approach to synthetic decentralized social graph generation. However, these existing works are all based on Edge-LDP which provides a weaker privacy guarantee than our work under Node-LDP.

2.9 Conclusion

To conclude, we first discuss the motivation for publishing the graph statistics under Node-LDP, and present the challenges of finishing the projection locally. We propose two methods for the projection parameter selection: pureLDP parameter selection and crypto-assisted parameter selection. Also, we design two methods for executing local graph projection: node-level local projection and edge-level local projection. Theoretical and experimental analysis verify the utility and privacy achieved by our proposed work.



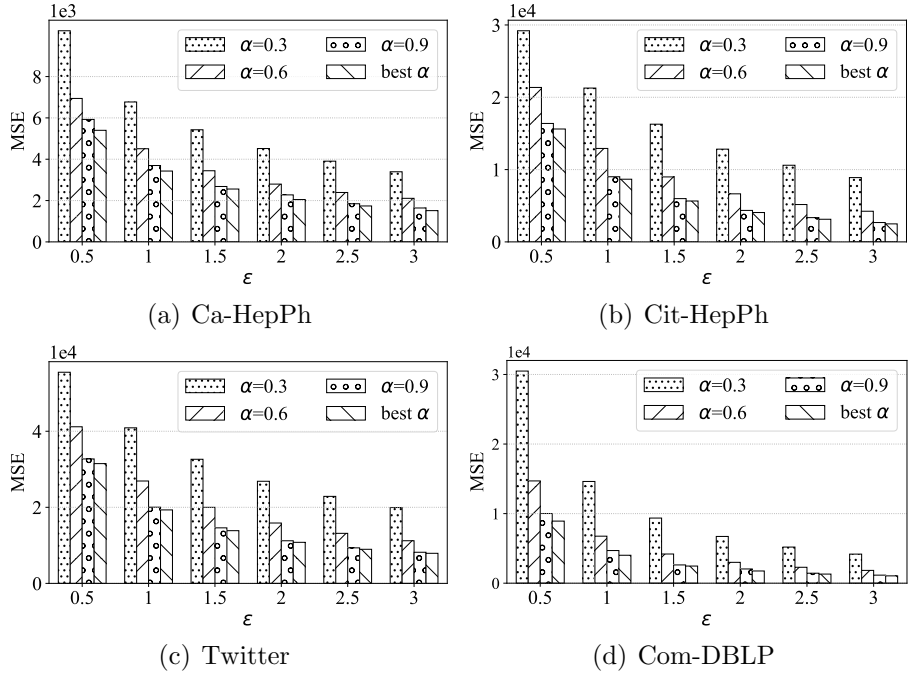
(b) Ca-HepPh (c) Cit-HepPh

(d) Twitter (e) Com-DBLP

(f) Ca-HepPh (g) Cit-HepPh

(h) Twitter (i) Com-DBLP

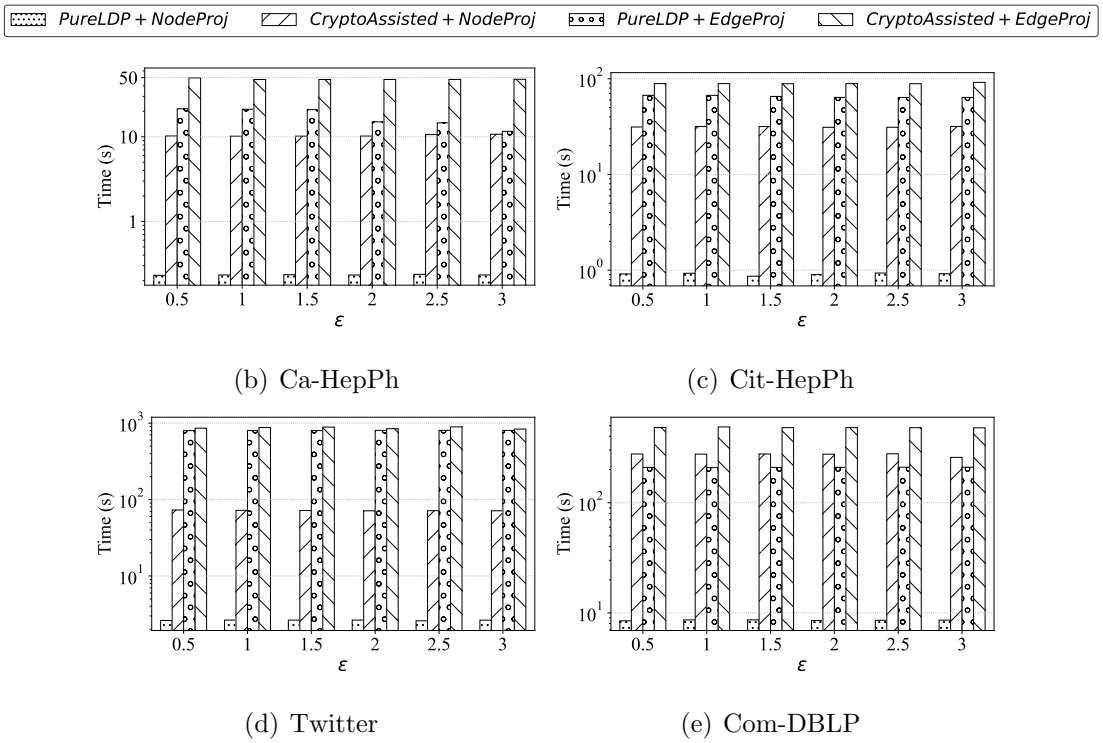
Figure 2.5. The MSE and MAE of algorithms on different graphs



(a) Ca-HepPh (b) Cit-HepPh

(c) Twitter (d) Com-DBLP

Figure 2.6. The MSE on different graphs, varying α



(b) Ca-HepPh (c) Cit-HepPh

(d) Twitter (e) Com-DBLP

Figure 2.7. The runtime on different graphs

CHAPTER 3

Crypto-Assisted Differentially Private Triangle Counting

3.1 Introduction

Graph data analysis is gaining popularity in various fields such as social networks, transportation systems, and protein forecasting due to its widespread presence. In graph analysis, triangle counting [50] is a crucial component for downstream tasks, including clustering coefficient [11], transitivity ratio [51], and structural similarity [52]. However, triangle counting involves sensitive individual information that could be leaked through the results of the process [53]. Differential privacy (DP) [14, 54] has been widely used to provide formal privacy protection. Existing works on differentially private triangle counting [2, 5, 7, 15, 22, 23] are mainly based on two DP models depending on the trust assumption of the server: *central differential privacy* (CDP), which requires a trusted server, and *local differential privacy* (LDP), which is preferable since it does not rely on a trusted server.

However, there is a significant utility gap between CDP-based [2, 15, 22] and LDP-based [5–7, 23] of differentially private triangle counting, which depends on whether or not there is a trusted server needed. In particular, CDP-based triangle counting models (as shown in Fig. 3.1(a)) need a trusted server to collect the

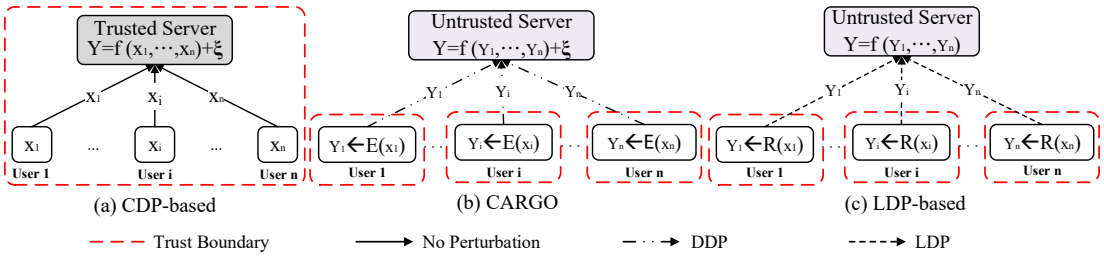


Figure 3.1. A comparison among CDP-based model, LDP-based model and CARGO. (a) CDP-based model relies on a trusted server and achieves a high accuracy, e.g., $O(\frac{d_{max}^2}{\varepsilon^2})$ error; (c) LDP-based model removes a trusted server but introduces more error, i.e., $O(\frac{e^\varepsilon}{(e^\varepsilon-1)^2}(d_{max}^3 n+\frac{e^\varepsilon}{\varepsilon^2}d_{max}^2 n))$; (b) Our CARGO achieves a good utility like CDP-based model, i.e., $O(\frac{d'^2_{max}}{\varepsilon^2})$, but without a trusted server like LDP-based model, where $\varepsilon, n, d_{max}, d'_{max}$ denote the privacy budget, number of users, true maximum degree, and noisy maximum degree, respectively.

whole graph before executing differentially private graph analysis. For a graph with n users and a privacy budget ε, the squared error of the central model is at most $O(\frac{d_{max}^2}{\varepsilon^2})$, where d_{max} is the maximum degree in a graph. In contrast, existing LDP-based triangle counting models (as shown in Fig. 3.1(c)) do not require a trusted server. Instead, each user perturbs its sensitive information using an LDP mechanism and sends noisy data to the untrusted server. The server then aggregates the data and releases a noisy triangle count. However, LDP-based triangle counting models incur more error of $O(\frac{e^\varepsilon}{(e^\varepsilon-1)^2}(d_{max}^3 n + \frac{e^\varepsilon}{\varepsilon^2}d_{max}^2 n))$ (refer to Table 2 in [5] as the state-of-the-art LDP-based triangle counting protocol), which is much larger than that of CDP-based models, especially, when n is large.

In this paper, we propose a **c**rypto-assisted differenti**a**lly p**r**ivate trian**g**le c**o**unting system (CARGO) that (1) achieves the high-utility triangle counting of the central model (2) without a trusted server like the local model. Our goal is to calculate the triangles in a graph, where each node represents a user and each edge denotes the relationship between users, while protecting each user's neighboring information (i.e., edges). Our system is inspired by recent studies [55–62] that employ cryptographic techniques to bridge the utility gap between LDP and CDP models. However, these systems are specifically designed to process tabular data [55–57] or gradients in federated learning [58–60] but not graph data. Designing secure and private methods for counting triangles requires new principles due to the high sensitivity of triangle counting and the limited view of local users for a global graph. As shown in Fig. 3.1(b), CARGO establishes a trust boundary

for local data by leveraging cryptographic primitives and distributed differential privacy instead of injecting LDP noise, allowing for high-utility triangle counting comparable to that of CDP-based models (see more details in section 3.3.1). We now elaborate on our key contributions:

Similarity-based projection. We propose a novel *local* projection method to reduce the sensitivity of the triangle counting while preserving more triangles. A significant obstacle in achieving differential privacy in counting triangles is the high sensitivity of triangle queries, leading to more noises needed for differentially private results. The basic idea to address this in a central setting [3, 15] is to project (i.e., truncate) the original graph into a bounded graph. The previous local graph projection method via randomly deleting edges [5] tries to reduce the sensitivity but results in much projection loss. Our similarity-based projection method relies on the significant fact that node degrees of a triangle are pretty similar to each other [63]. We prioritize deleting the edges with the least possibility of constructing triangles, which results in preserving more triangles. It is worth noting that this simple yet efficient local projection algorithm can also be used to improve the utility of locally private triangle counting (details in Section 3.3.3).

ASS-based triangle counting. We introduce a novel triangle counting algorithm based on an additive secret sharing (ASS) technique [64]. Local users often face difficulties in calculating triangle counts due to their limited view of the global graph. This limitation prevents them from seeing the third edges between others. The state-of-the-art triangle counting method [5] attempts to address this problem in untrusted settings by including an additional round of interaction. However, there is still a significant gap in utility compared to the central model. We propose a secure triangle counting method based on the ASS method with high accuracy. A triangle exists in a graph if three edges of a triple exist simultaneously. Namely, the multiplication of three bits in a matrix is equal to 1. We introduce a protocol for multiplying three secret values, which allows us to securely and accurately compute the triangle counts while protecting sensitive neighboring information (details in Section 3.3.4).

Distributed perturbation. We present a distributed perturbation method by combining additive secret sharing [64] and distributed noise generation [65–67]. The previous state-of-the-art crypto-assisted differential privacy (crypto-assisted DP) method [56] randomizes the private value by adding two instances of Laplace

noise, which leads to significant loss of utility. Our distributed perturbation adds minimal but sufficient noise to the local user data. This partial noise is insufficient to provide an LDP guarantee but the aggregated noise is enough to provide a CDP protection. Furthermore, secret sharing ensures that two untrusted servers only see encoded values beyond other information. And the final aggregated noise can provide ε-Edge Distributed Differential Privacy (DDP) guarantee (details in Section 3.3.5).

Comprehensive theoretical and empirical analysis. We provide a comprehensive theoretical analysis of our proposed protocols, including utility, privacy, and time complexity. In particular, we provide the upper-bounds on the estimation error for triangle counting and find that CARGO can significantly reduce the estimation error of local models. Additionally, we prove that our proposed CARGO satisfies ε-Edge DDP (details in Section 3.4). Finally, several experiments have been conducted to demonstrate that our CARGO achieves high-utility triangle counts comparable to central models, and significantly outperforms local models by at least an order of 5 in utility (details in Section 3.5).

3.2 Preliminaries

3.2.1 Problem Statement

Graphs and Triangle Counting

In our work, we consider an undirected graph with no additional attributes on nodes or edges, which can also be represented as $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ is the set of nodes, and $E \subseteq V \times V$ is the set of edges. Each local user v_i owns one adjacent bit vector $A_i = \{a_{i1}, ..., a_{in}\}$ that records the neighboring information, where $a_{ij} = 1, j \in [n]$ if and only if edge $\langle v_i, v_j \rangle \in E$. The adjacent bit vectors of all local users compose a symmetric adjacency matrix $A = \{A_1, A_2, ..., A_n\}$. A triangle in a graph G consists of three nodes where each node connects to the other two nodes. Table 3.1 summarizes the major notations used in this paper.

Table 3.1. Summary of Notations.

Notation	Definition
$G = (V, E)$	Graph with nodes V and edges E
n	Number of users
v_i	i-th node in V
d_i	Node degree of v_i
A	Adjacent matrix
A_i	Adjacent bit vector of v_i
D	True degree set
D'	Noisy degree set
d_{max}	True maximum degree
d'_{max}	Noisy maximum degree
$\triangle$	Sensitivity of triangle counting
T	True number of triangles
T'	Noisy number of triangles

Trust Assumptions

Our system includes n users and two servers, as illustrated in Fig. 3.2. We assume that two servers are *semi-honest* and *non-colluding*. This is a common assumption in cryptographic systems, such as [56, 58, 68, 69], and can be enforced via strict legal bindings. Semi-honest implies that they follow the protocol instructions honestly but may be curious about additional information. Non-colluding means that they do not disclose any information to each other beyond what is allowed by the defined protocol. Furthermore, we assume that there are no corrupt users, and each user has a private channel with each server to share sensitive information confidentially. We also assume that any parties beyond the system, such as servers, analysts, or other individuals, are adversaries who are computationally constrained.

Utility Metrics

We use two common utility metrics to evaluate our methods, including l_2 loss (e.g., squared error) like [70,71], and relative error as with [72,73]. To be specific, let T' be a private estimation of the true triangles T. The l_2 loss function maps the true number of triangles T and the private estimation T' to the l_2 loss, which can be denoted by: $l_2(T, T') = (T - T')^2$. When T is large, the l_2 loss may also be large. Thus, we also compute the relative error in our experiments. The relative

error is defined as: $re(T, T') = \frac{|T-T'|}{T}$, where $T \neq 0$.

3.2.2 Differential Privacy on Graphs

Differential privacy (DP) [14, 54] has become a standard for privacy protection, which can formalized in Definition 3. Based on different trusted assumptions, DP can be divided into two types: *central differential privacy* (CDP) and *local differential privacy* (LDP).

Definition 3 (Differential Privacy [54]) *Let n be the number of users. Let $\varepsilon > 0$ be the privacy budget. Let $\mathcal{X}$ be the set of input data for each user. A randomized algorithm $\mathcal{M}$ with domain $\mathcal{X}^n$ satisfies ε-DP, iff for any neighboring databases $D, D' \in \mathcal{X}^n$ that differ in a single datum and any subset $S \subseteq Range(\mathcal{M})$,*

$$Pr[\mathcal{M}(D) \in S] \leq e^{\epsilon} Pr[\mathcal{M}(D') \in S]$$

In this work, we use the global sensitivity [54] to achieve the DP. The global sensitivity considers the maximum difference between query results on two neighboring databases.

Edge DP. As a graph consists of nodes and edges, there are two definitions when DP is applied to either of them: *edge differential privacy* (Edge DP) [19] and *node differential privacy* (Node DP) [19]. Edge DP guarantees the output of a randomized mechanism does not reveal whether any friendship information (i.e., edge) exists in a graph G; whereas Node DP hides the existence of one user (e.g., node) along with her adjacent edges. Node DP provides a stronger privacy guarantee since it protects not only edge information but also node information. But Node DP brings much more error than Edge DP. Considering our privacy goal (protecting the neighboring information) and higher accuracy requirement, we choose to use Edge DP in our work like [2, 5–7, 23].

Definition 4 and Definition 5 give the formal definition of Edge CDP and Edge LDP respectively. Edge LDP assumes that two edges $\langle v_i, v_j \rangle$ and $\langle v_j, v_i \rangle$ between user v_i and v_j are different secrets [5].

Definition 4 (Edge CDP [74]) *Let $\varepsilon > 0$ be the privacy budget. A randomized algorithm $\mathcal{M}$ with domain $\mathcal{G}$ satisfies ε-Edge CDP, iff for any two neighboring graphs $G, G' \in \mathcal{G}$ that differ in one edge and any subset $S \subseteq Range(\mathcal{M})$,*

$$Pr[\mathcal{M}(G) \in S] \leq e^{\epsilon} Pr[\mathcal{M}(G') \in S]$$

Definition 5 (Edge LDP [32]) *Let $\varepsilon > 0$ be the privacy budget. For any $i \in [n]$, let $\mathcal{M}_i$ be a randomized algorithm of user v_i. $\mathcal{M}_i$ satisfies ε-Edge LDP, iff for any two neighboring adjacent bit vectors A_i and A_i' that differ in one edge and any subset $S \subseteq Range(\mathcal{M}_i)$,*

$$Pr[\mathcal{M}_i(A_i) \in S] \leq e^{\epsilon} Pr[\mathcal{M}_i(A_i') \in S]$$

Edge DDP. Combining DDP [65–67] with standard Edge CDP (Definition 4), we introduce an Edge Distributed Differential Privacy (Edge DDP) for protecting triangle counting in CARGO. The formal definition is as follows:

Definition 6 (Edge DDP) *Let $\varepsilon > 0$ be the privacy budget. Let r_i be a distributed noise generated by user v_i. A randomized algorithm M with randomness over the joint distribution of $\boldsymbol{r}$:=$(r_1, ..., r_n)$ satisfies ε-Edge DDP, iff for any neighboring graphs G and G' that differ in one edge, for any output $y \in range(\mathcal{M})$,*

$$Pr[\mathcal{M}(G) = y] \leq e^{\varepsilon} Pr[\mathcal{M}(G') = y]$$

Note that both ε-Edge LDP (Definition 5) and ε-Edge DDP (Definition 6) protect one edge with privacy budget ε. The difference is that Edge LDP is for the local model, whereas Edge DDP is for the crypto-assisted DP model. We use Edge LDP to prove Edge DDP for the entire process of our CARGO.

3.2.3 Additive Secret Sharing

In two-party additive secret sharing (ASS) [64], a private value is split into two secret shares that can be used to construct the true value, like [75–77]. Each value is represented as an l-bit integer in the ring $\mathbb{Z}_{2^l}$, and cannot reveal any information about the private value. In this paper, we denote a secret share of x by $\langle x \rangle$. To additively share a secret value x, a random number $r \in \mathbb{Z}_{2^l}$ is generated. Then the shares for parties S_1 and S_2 can be represented as $\langle x \rangle_1 = r \bmod 2^l$, $\langle x \rangle_2 = (x - r) \bmod 2^l$, respectively, where $\langle x \rangle = \langle x \rangle_1 + \langle x \rangle_2$. The basic operations naturally supported in the ASS domain include addition and multiplication. Given two shared values $\langle x \rangle$ and $\langle y \rangle$, each party $S_{i \in \{1,2\}}$ receives $\langle x \rangle_i$ and $\langle y \rangle_i$. Each party S_i locally computes $\langle u \rangle_i = \langle x \rangle_i + \langle y \rangle_i$. Then, the addition can be securely computed

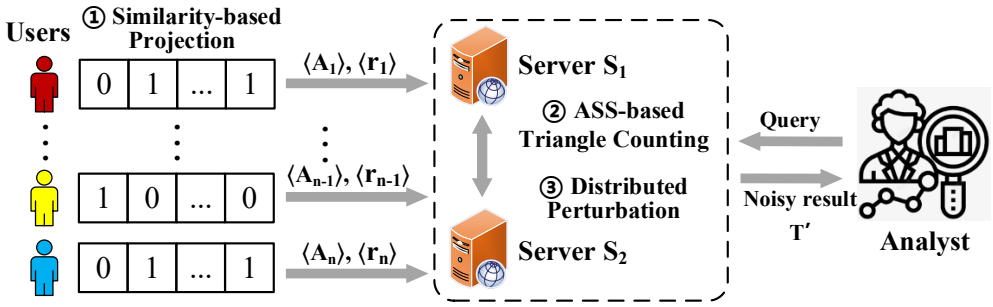


Figure 3.2. CARGO system.

by aggregating $\langle u \rangle_1$ and $\langle u \rangle_2$, i.e., $u = \langle u \rangle_1 + \langle u \rangle_2 = \langle x \rangle_1 + \langle x \rangle_2 + \langle y \rangle_1 + \langle y \rangle_2 = x + y$. The multiplication of two shared values may be complex. It needs one-round communication of the Beaver triple, which can be prepared offline.

3.3 CARGO System

3.3.1 Design principle

Our main idea is to let the users and two servers *collaboratively* compute secret shares of the true triangle counts securely and add distributed differentially private noise without requiring any trusted servers. The previous crypto-assisted differentially private data analysis protocols [55–60] are designed for the general tabular data. The high sensitivity of triangle counting and the limited view of local users for a global graph make these protocols less effective for private graph data analysis. Furthermore, Cryptε [56], a state-of-the-art crypto-assisted DP protocol, employs two non-colluding and untrusted servers to independently add Laplace noise twice, resulting in twice the utility loss compared to CDP models. To this end, we first design a novel local projection based on triangle homogeneity to reduce the high sensitivity from $O(n)$ to $O(d'_{max})$ while preserving more triangles as much as possible. Then, we propose a secure protocol for counting triangles using additive secret sharing techniques, which enables users and two servers to collaboratively calculate the secret shares of true triangle counts T, i.e., $\langle T \rangle_1$ and $\langle T \rangle_2$. We then introduce a distributed perturbation algorithm that adds minimal yet sufficient noise to triangles for privacy preservation. To safeguard the privacy of the partial noise, users do not directly transmit it to the server. Each user encodes the noise using additive secret sharing and then distributes it

Algorithm 6 Overall protocol of CARGO system

Input: G represented as adjacent lists $A = \{A_1, ..., A_n\}$,
True degree set $D = \{d_1, ..., d_n\}$,
Privacy budget $\varepsilon = \varepsilon_1 + \varepsilon_2$
Output: Noisy triangle count T'

1: Initialize: $T = \varnothing$
Step 1: Similarity-based Projection
2: $(D', d'_{max}) \leftarrow \mathsf{Max}(D, \varepsilon_1)$ ▷ Algorithm 7
3: $\hat{A} \leftarrow \mathsf{Project}(A, D, D', d'_{max})$ ▷ Algorithm 8
Step 2: ASS-based Triangle Counting
4: $\langle T \rangle \leftarrow \mathsf{Count}(\hat{A})$ ▷ Algorithm 9
Step 3: Distributed Perturbation
5: $T' \leftarrow \mathsf{Perturb}(\langle T \rangle, d'_{max}, \varepsilon_2)$ ▷ Algorithm 10
6: **return** T'

to the servers. The servers integrate this encoded noise into their secret shares of the triangle counts. By aggregating these shares, the servers can compute the differentially private triangle counts accurately.

3.3.2 Framework

Fig. 3.2 shows CARGO's system architecture. CARGO involves two kinds of entities: local users and two non-colluding servers. The local user v_i, $i \in [n]$ (n = number of users in a graph), owns the sensitive friendship information which is represented as an adjacent bit vector. At the beginning, local users interact with one of the servers for the private estimation of maximum degree d'_{max}. The local user then projects the original adjacent bit vector into d'_{max}-bounded adjacent bit vector (step ①). Next, the local user secretly shares her adjacent bit vector to two servers and each server computes the secret share of the true triangle counts (step ②). Subsequently, the local user generates a distributed noise and secretly shares it with two servers. Two servers sum up their own shares of the noise, and add aggregated noise into the share of the triangle counts, respectively. In other words, each server obtains the secret share of the noisy triangle counts personally. The final aggregation of two shares is equal to the noisy triangle counts of the entire graph, which satisfies ε-Edge Distributed DP (step ③).

Main Steps. Algorithm 6 presents the overall protocol of CARGO system, which consists of three main steps:

Step 1: *Similarity-based Projection.* Graph projection is the key technique to reduce the global sensitivity from $O(n)$ to $O(\theta)$, where θ is the projection parameter. Here, we set θ as the maximum degree d_{max} to avoid removing edges from an adjacent bit vector (e.g., to avoid the loss of utility). However, there is no prior knowledge about the maximum degree. CARGO recalls a $\mathsf{Max}(.)$ function (Algorithm 7) to privately compute a noisy maximum degree d'_{max} that is approximately equal to d_{max}, as shown in Table 3.5. Next, each user transforms the original adjacent bit vector into a d'_{max}-bounded adjacent bit vector using a $\mathsf{Project}()$ function (Algorithm 8). Previous local projection method via randomly deleting edges [5] results in much projection loss. We propose a similarity-based projection method by leveraging the triangle homogeneity (Observation 1). During the projection, the user deletes the edges with the least possibility of constructing triangles, and thus more triangles are preserved (see more details in Section 3.3.3).

Step 2: *ASS-based Triangle Counting.* Next, the true triangle count can be computed based on the projected adjacent bit vector. The main challenge in triangle counting is that each user has a limited view of a global graph. In other words, local users cannot see the third edge between others. The previous state-of-the-art two-round triangle counting method [5] in untrusted settings leads to more errors. We propose an ASS-based triangle counting method for securely and accurately calculating the triangle count. Each user encodes each bit in its adjacent bit vector via additive secret sharing and sends it to two sever. Each server obtains the secret share of the triangle count and knows nothing about the true result. CARGO recalls a $\mathsf{Count}()$ function (Algorithm 9) to calculate the number of triangle counts securely and accurately via Additive Secret Sharing (ASS) (refer to Section 3.3.4 for more details).

Step 3: *Distributed Perturbation.* After accurately calculating the shares of triangle counts, CARGO employs a $\mathsf{Perturb}()$ function (Algorithm 10) to privately estimate the triangle count of a graph. The previous state-of-the-art crypto-assisted DP method [56] guarantees differential privacy by incorporating two instances of Laplace noise, which results in a significant loss of utility. We propose a distributed perturbation method by combining the additive secret sharing and distributed noise. Each user first generates a minimal but sufficient noise. Since such a small amount of noise is unable to provide enough privacy guarantees compared with LDP, we employ ASS to let each user split the generated noise

Algorithm 7 Max: private estimation of d_{max}

Input: True degree set $D = \{d_1, ..., d_n\}$,
Privacy budget ε_1
Output: Noisy maximum degree d'_{max}

Initialize: $D' = \varnothing$
for $i = 1$ **to** n **do**
  $d'_i \leftarrow d_i + \mathsf{Lap}(\frac{1}{\varepsilon_1})$
  $D' \leftarrow D' \cup \{d'_i\}$
  Send d'_i to untrusted server
end for
Server: $d'_{max} \leftarrow \mathsf{max}(d'_1, ..., d'_n)$
return (D', d'_{max})

into two secrets and share them with two servers, respectively. The servers, in turn, are unable to decipher any information about the noise independently. By merging the shared noise with the shared triangle count, each server acquires a secret share of the noisy triangle count. The final computation of the noisy triangle count, safeguarded under ε-Edge Distributed DP by aggregating two shared noisy results (details in Section 3.3.5).

Extension to Node DP. CARGO can be extended to Node DP by revising Algorithm 7 and Algorithm 10. The main change is from the sensitivity updates. To be specific, given the number of nodes n, in Algorithm 7, any change of one node will influence the other $(n-1)$ node degrees in the worst case. Thus, the sensitivity of Max is $O(n)$ when we use Node DP. Similarly, the sensitivity of Perturb in Algorithm 10 becomes $O\binom{d'_{max}}{2}$. Although our algorithm Project can reduce the high sensitivity from $O\binom{n}{2}$ to $O\binom{d'_{max}}{2}$, there are still much more utility loss than Edge DP. Therefore, how to reduce the high sensitivity of Node DP while preserving more triangles is the focus of future work.

3.3.3 Similarity-based Projection

Private Estimation of the Maximum Degree

In this work, we assign the maximum degree d_{max} as the projection parameter θ just like [3, 5], primarily for two main reasons. On the one hand, as shown in Table 3.4, d_{max} is much smaller than the number n of users in real-world graphs, which can significantly reduce the sensitivity. On the other hand, it

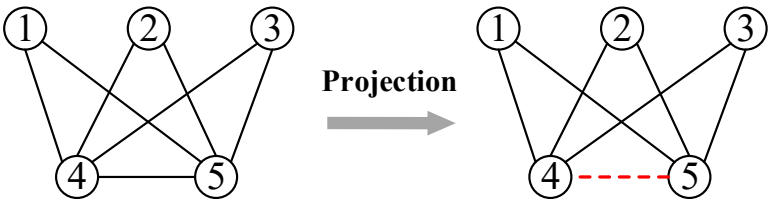


Figure 3.3. Limitation of random deletion. Assume that d'_{max} is equal to 3, if user v_4 (or v_5) projects the adjacent list by deleting the edge $\langle v_4, v_5 \rangle$, all triangles in a graph will be removed.

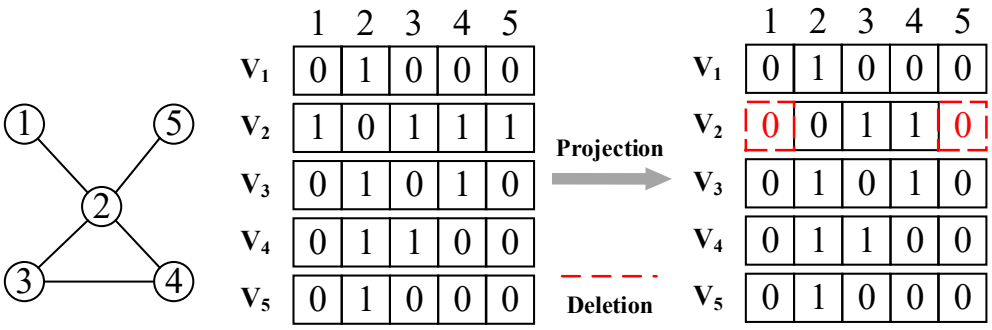


Figure 3.4. Local graph projection. Assume that d'_{max} is equal to 2, user v_2 projects the adjacent bit vector by deleting the edge $\langle v_2, v_1 \rangle$ and edge $\langle v_2, v_5 \rangle$.

avoids deleting neighboring friends from an adjacent list ideally; i.e., it avoids the utility loss during the projection. In the untrusted scenario, however, each user knows no about d_{max} since it has a limited view of the global graph. To handle this, local users privately compute d_{max} with the leverage of the server's global view. Specifically, as shown in Algorithm 7, each user v_i first adds $\mathsf{Lap}(\frac{1}{\varepsilon_1})$ to her node degree d_i. Here, we use Edge LDP (Definition 5), and the sensitivity is one since two edges $\langle v_i, v_j \rangle$ and $\langle v_j, v_i \rangle$ between user v_i and v_j are different secrets [5], and any change of one edge will influence one node degree. Then each user sends noisy degree d'_i to the untrusted server. In CARGO, one of two parties S_1 and S_2 can be used for computing d'_{max}. Finally, the server computes the max value of the noisy degree sequence $\{d'_1, ..., d'_n\}$ as d'_{max}, and sends d'_{max} back to local users. We denote this algorithm by Max.

Local Graph Projection

After obtaining the estimation of the maximum degree d'_{max}, each user v_i transforms the original adjacent bit vector A_i into a d'_{max}-bounded adjacent bit vector

$\hat{A}_i$ via the graph projection. Although there have been some works involving graph projection methods [2, 3, 5, 78], they do not perform in the triangle counting within untrusted settings very well. For instance, the most related work is that Imola et al. [5] implements graph projection via randomly deleting edges in untrusted settings. However, this random projection possibly deletes key edges involved in many triangles. For example, in Fig. 3.3, if user v_4 randomly deletes the edge $\langle v_4, v_5 \rangle$ to bound the adjacent list, all triangles in a graph will disappear.

We propose a similarity-based projection for reducing the global sensitivity in untrusted scenarios. The main target of our methods is that the high global sensitivity can be reduced while more triangles can be preserved after the projection. As illustrated in Fig. 3.4, if $d_i > d'_{max}$, user v_i will delete $(d_i - d'_{max})$ friends from her adjacent bit vector. The candidates to be deleted are selected based on significant knowledge in the following observation:

Observation 1 (Triangle Homogeneity [63]) *Node degrees of a triangle are quite similar to each other in a graph (i.e., social and interaction graphs).*

Durak *et al.* [63] demonstrate this observation through experiments conducted on graphs from various scenarios. According to Observation 1, a significant number of triangles can be preserved by selecting nodes with a high degree of similarity. This is an intuition behind our proposed local graph projection method in triangle counting. The degree similarity between two nodes is quantified as outlined in Definition 7. It is important to note that Equation 3.1 reflects the relative difference in degrees between two nodes. Consequently, a lower value of $DS(d_1, d_2)$ indicates a higher degree of similarity between the nodes.

Definition 7 (Degree Similarity) *Given two node degrees d_1 and d_2 in a graph, the degree similarity between them is computed by*

$$DS(d_1, d_2) = \frac{|d_1 - d_2|}{d_1} \tag{3.1}$$

Algorithm 8 shows the details of our local graph projection method. It takes as input the adjacent matrix of a graph, true degree set D, noisy degree set D', and noisy maximum degree d'_{max}. If $d_i > d'_{max}$, each user v_i first initializes an array ds with size n and an empty set $\hat{A}_i$, where ds records degree similarities and $\hat{A}_i$ is the projected adjacent bit vector (line 3). Then user v_i computes the degree

Algorithm 8 Project: Similarity-based Projection

Input: Adjacent matrix A of a graph
True degree set $D = \{d_1, ..., d_n\}$
Noisy degree set $D' = \{d'_1, ..., d'_n\}$
Noisy maximum degree d'_{max}

Output: Projected adjacent matrix $\hat{A}$

1: **for** each user $v_i, i \in [1, n]$ in a graph **do**
2: **if** $d_i > d'_{max}$ **then**
3: Initialize: $ds = [0] * n, \hat{A}_i = \varnothing$
4: **for** $j = 1$ **to** n **do**
5: **if** $A_{ij} == 1$ **then**
6: $ds[j] \leftarrow \mathsf{DS}(d_i, d'_j)$
7: **end if**
8: **end for**
9: Sort ds in ascending order
10: $\hat{ds} \leftarrow ds[1 : d'_{max}]$
11: **for** $j = 1$ **to** n **do**
12: **if** $ds[j]$ in $\hat{ds}$ **then**
13: $\hat{A}_i \leftarrow \hat{A}_i \cup \{1\}$
14: **else**
15: $\hat{A}_i \leftarrow \hat{A}_i \cup \{0\}$
16: **end if**
17: **end for**
18: **else if** $d_i \leq d'_{max}$ **then**
19: $\hat{A}_i \leftarrow A_i$
20: **end if**
21: **end for**
22: **return** $\hat{A}$

similarities between d_i and all her friends based on Definition 7, and records the similarities using the array ds (line 5). Here, the neighboring node degrees are noisy degrees that have been calculated in Max function (Algorithm 7). Then the array ds is sorted in ascending order (line 6), and then the top d'_{max} elements are sliced and stored into the array $\hat{ds}$ (line 7). After that, user v_i traverses each neighboring friend $v_k, k \in [n]$ and checks the degree similarity $ds[k]$ between them. If the element $ds[k]$ is in $\hat{ds}$, the bit '1' will be added into $\hat{A}_i$; the bit '0' will be added otherwise (line 9-12). If $d_i \leq d'_{max}$, the projected adjacent bit vector $\hat{A}_i$ will be set as the original vector A_i. The final answer is the projected adjacent matrix $\hat{A}$. We denote this algorithm by Project.

3.3.4 Additive Secret Sharing-based Triangle Counting

Our triangle counting strategy is based on an intriguing fact that a triangle exists if three neighboring edges of a triple exist simultaneously, namely, $a_{ij} \times a_{ik} \times a_{jk} = 1$, where $i, j, k \in [1, n]$. The challenge lies in computing the multiplication of three adjacent bits while preserving the privacy of neighboring information. We seek help from the multiplication of secret values using additive secret sharing. However, existing protocols [75,76] mainly focus on multiplying two secret values, which cannot be directly employed for triangle counting.

Motivated by this, we propose a secure multi-party triangle counting protocol that executes the multiplication of three secret values utilizing additive secret sharing. Given three secret values a, b, c, our goal is to compute the multiplication of these three secrets while not leaking anything about secrets, namely, $d = a \times b \times c$. Like the multiplication of two secrets, two servers precompute Multiplication Groups (MGs) via oblivious transfer [79, 80]. MGs refer to a set of shared values:x, y, z, w, o, p, q, where $w = x \times y \times z, o = x \times y, p = x \times z, q = y \times z$. Each value is represented as an l-bit integer in the ring $\mathbb{Z}_{2^l}$. In offline phase, server S_1 receives $\langle x\rangle_1, \langle y\rangle_1, \langle z\rangle_1, \langle w\rangle_1, \langle o\rangle_1, \langle p\rangle_1, \langle q\rangle_1$, and server S_2 receives $\langle x\rangle_2, \langle y\rangle_2, \langle z\rangle_2, \langle w\rangle_2, \langle o\rangle_2, \langle p\rangle_2, \langle q\rangle_2$. After having shares of MGs, the multiplication is performed as follows:

1. Server S_i ($i \in \{1, 2\}$) computes $\langle e\rangle_i = \langle a\rangle_i - \langle x\rangle_i$,
   $\langle f\rangle_i = \langle b\rangle_i - \langle y\rangle_i$, and $\langle g\rangle_i = \langle c\rangle_i - \langle z\rangle_i$

2. Both server S_1 and S_2 communicate to reconstruct e, f, and g.

3. Server S_i computes its secret share of the multiplication result as: $\langle d\rangle_i = \langle w\rangle_i + \langle xy\rangle_i g + \langle xz\rangle_i f + \langle yz\rangle_i e + \langle x\rangle_i fg + \langle y\rangle_i eg + \langle z\rangle_i ef + (i-1)efg$

Theorem 4 *Our proposed multiplication of three secret values is correct.*

Proof of Theorem 4. $d = \langle d\rangle_1 + \langle d\rangle_2 = \langle w\rangle_1 + \langle xy\rangle_1 g + \langle xz\rangle_1 f + \langle yz\rangle_1 e + \langle x\rangle_1 fg + \langle y\rangle_1 eg + \langle z\rangle_1 ef + \langle w\rangle_2 + \langle xy\rangle_2 g + \langle xz\rangle_2 f + \langle yz\rangle_2 e + \langle x\rangle_2 fg + \langle y\rangle_2 eg + \langle z\rangle_2 ef + efg$
$= w + xyg + xzf + yze + xfg + yeg + zef + efg.$
Then, we put $e = a - x, f = b - y, g = c - z$ into above equation and obtain $d = a \times b \times c$.

Algorithm 9 shows how to securely calculate the secret shares of true triangle count based on our proposed multiplication protocol of three secret shares. It

Algorithm 9 Count: ASS-based Triangle Counting

Input: Projected adjacent matrix $\hat{A} = \{\hat{A}_1, ..., \hat{A}_n\}$
Output: Secret shares of triangle count $\langle T \rangle$

Initialize: $\langle T \rangle_1 = \langle T \rangle_2 = 0$
for $i = 1$ **to** n **do**
  for $j = i + 1$ **to** n **do**
    for $k = j + 1$ **to** n **do**
      Initialize: $w = xyz, o = xy, p = xz, q = yz$
        $\langle x \rangle_1, \langle y \rangle_1, \langle z \rangle_1, \langle w \rangle_1, \langle o \rangle_1, \langle p \rangle_1, \langle q \rangle_1 \rightarrow S_1$
        $\langle x \rangle_2, \langle y \rangle_2, \langle z \rangle_2, \langle w \rangle_2, \langle o \rangle_2, \langle p \rangle_2, \langle q \rangle_2 \rightarrow S_2$
      Server S_1: $\langle e \rangle_1 = \langle a_{ij} \rangle_1 - \langle x \rangle_1$
        $\langle f \rangle_1 = \langle a_{ik} \rangle_1 - \langle y \rangle_1$
        $\langle g \rangle_1 = \langle a_{jk} \rangle_1 - \langle z \rangle_1$
      Server S_2: $\langle e \rangle_2 = \langle a_{ij} \rangle_2 - \langle x \rangle_2$
        $\langle f \rangle_2 = \langle a_{ik} \rangle_2 - \langle y \rangle_2$
        $\langle g \rangle_2 = \langle a_{jk} \rangle_2 - \langle z \rangle_2$
      Server S_1 and S_2 communicate and obtain:
        $e = \langle e \rangle_1 + \langle e \rangle_2$, $f = \langle f \rangle_1 + \langle f \rangle_2$, $g = \langle g \rangle_1 + \langle g \rangle_2$
      Server S_1: $u_1 = \langle w \rangle_1 + \langle xy \rangle_1 g + \langle xz \rangle_1 f + \langle yz \rangle_1 e$
        $+\langle x \rangle_1 fg + \langle y \rangle_1 eg + \langle z \rangle_1 ef$
      Server S_2: $u_2 = \langle w \rangle_2 + \langle xy \rangle_2 g + \langle xz \rangle_2 f + \langle yz \rangle_2 e$
        $+\langle x \rangle_2 fg + \langle y \rangle_2 eg + \langle z \rangle_2 ef + efg$
      Server S_1: $\langle T \rangle_1 \leftarrow \langle T \rangle_1 + u_1$
      Server S_2: $\langle T \rangle_2 \leftarrow \langle T \rangle_2 + u_2$
    end for
  end for
end for
$\langle T \rangle \leftarrow \{\langle T \rangle_1, \langle T \rangle_2\}$
return $\langle T \rangle$

takes as input a projected graph that is represented as an adjacent matrix $\hat{A} = \{\hat{A}_1, ..., \hat{A}_n\}$. It traverses all possible triangle triples by computing $u = a_{ij} \times a_{ik} \times a_{jk}$. If $u = 1$, these three nodes and three edges constitute a triangle; otherwise not. Note that we reduce the repeat computation by only $u = a_{ij} \times a_{ik} \times a_{jk} (i < j < k)$. Each user secretly shares each bit of its adjacent bit vector to two servers. Currently, server S_1 obtains $\langle a_{ij} \rangle_1$ and S_2 owns $\langle a_{ij} \rangle_2$, where $i, j \in [1, n]$. For each possible triangle, it first initializes the multiplication groups including x, y, z, w, o, p, q, where $w = x \times y \times z, o = x \times y, p = x \times z, q = y \times z$. The multiplication groups can be precomputed in the offline phase and then shared

with two servers (line 5). After that, server S_1 computes $\langle e\rangle_1 = \langle a_{ij}\rangle_1 - \langle x\rangle_1$, $\langle f\rangle_1 = \langle a_{ik}\rangle_1 - \langle y\rangle_1$, $\langle g\rangle_1 = \langle a_{jk}\rangle_1 - \langle z\rangle_1$. Server S_2 computes $\langle e\rangle_2 = \langle a_{ij}\rangle_2 - \langle x\rangle_2$, $\langle f\rangle_2 = \langle a_{ik}\rangle_2 - \langle y\rangle_2$, $\langle g\rangle_2 = \langle a_{jk}\rangle_2 - \langle z\rangle_2$ (line 6-7). Next, two servers communicate and reconstruct e, f, and g. It is worth noting that secret values are masked with random values x, y, z, and thus any server knows nothing about a_{ij}, a_{ik}, a_{jk}. Then, server S_1 computes the secret share of the current triangle, namely, u_1. Server S_2 computes the secret share of the current triangle, namely, u_2 (line 9-10). S_1, S_2 adds u_1, u_2 into $\langle T\rangle_1$ and $\langle T\rangle_2$, respectively. Finally, each server obtains the secret share of the true triangle counting, namely, $\langle T\rangle_1$ and $\langle T\rangle_2$. The server cannot know any information about true triangle count T from $\langle T\rangle_1$ or $\langle T\rangle_2$. The final answer of Algorithm 9 is the secret shares of true triangles. We denote this algorithm by Count.

3.3.5 Distributed Perturbation

Upon computing the true triangle count T, we can add DP noise into the triangle count, in order to guarantee that the final output is differentially private. The state-of-the-art crypto-assisted differential privacy model [56] adds two instances of Laplace noise to the query result in tabular data analysis. However, the additional round randomization brings more noise than the CDP model. We propose a distributed perturbation method by combining the additive secret sharing [64, 75, 76] and distributed noise generation method [65–67]. Each user first generates sufficient but minimal noise and then sends it to two servers. Since such a noise is also sensitive, we employ additive secret sharing to encode the noise and share them with two servers, respectively. The server can not obtain any information about each added noise and the final noisy triangle count is protected under ε-Edge Distributed DP. The Laplace mechanism has been widely used for protecting the triangle counting in CDP or LDP model [2, 5, 7], and a key property of Laplace distribution, namely, infinite divisibility [67, 81] (as presented in Lemma 3), allows us to simulate the Laplace noise by summing up n other random variables from independent identically distribution.

Lemma 3 (Infinite Divisibility [81]) *Let $Lap(\lambda)$ denote a random variable that is sampled from a Laplace distribution with* PDF $f(x, \lambda) = \frac{1}{2\lambda} e^{\frac{|x|}{\lambda}}$. *Then the distribution of $Lap(\lambda)$ is infinitely divisible; i.e., $Lap(\lambda)$ can be expressed as the sum of an arbitrary number of independent and identically distributed (i.i.d.)*

Algorithm 10 Perturb: Distributed Perturbation

Input: Secret share of triangle count $\langle T \rangle$,
Noisy maximum degree d'_{max},
Privacy budget ε_2
Output: Noisy triangle count T'

1: **for** each user v_i, $i \in [1, n]$ **do**
2: $Gam_1 = \mathsf{Gamma}(n, \frac{d'_{max}}{\varepsilon_2})$
3: $Gam_2 = \mathsf{Gamma}(n, \frac{d'_{max}}{\varepsilon_2})$
4: $\gamma_i = (Gam_1 - Gam_2)$
5: Split γ_i into two secret shares
$\gamma_i = \langle \gamma_i \rangle_1 + \langle \gamma_i \rangle_2$
6: Send $\langle \gamma_i \rangle_1, \langle \gamma_i \rangle_2$ to two servers S_1, S_2
$\langle \gamma_i \rangle_1 \rightarrow S_1$, $\langle \gamma_i \rangle_2 \rightarrow S_2$
7: **end for**
8: Server S_1: Aggregate n shared distributed
$\langle \gamma \rangle_1 = \sum_{i=1}^{n} \langle \gamma_i \rangle_1$
9: Server S_2: Aggregate n secret shares
$\langle \gamma \rangle_2 = \sum_{i=1}^{n} \langle \gamma_i \rangle_2$
10: Server S_1: Compute the secret share of T'
$\langle T' \rangle_1 = \langle T \rangle_1 + \langle \gamma \rangle_1$
11: Server S_2: Compute the secret share of T'
$\langle T' \rangle_2 = \langle T \rangle_2 + \langle \gamma \rangle_2$
12: Server S_1, S_2: Communicate and compute
$T' = \langle T' \rangle_1 + \langle T' \rangle_2$
13: **return** T'

random variables. Specifically, for arbitrary integer $n \geq 1$,

$$Lap(\lambda) = \sum_{i=1}^{n} [Gam_1(n, \lambda) - Gam_2(n, \lambda)], \tag{3.2}$$

where $Gam_1(n, \lambda)$ and $Gam_2(n, \lambda)$ are independent Gamma distributed random variables with densities as follows,

$$Gamma(x, n, \lambda) = \frac{(1/\lambda)^{1/n}}{\Gamma(1/n)} x^{\frac{1}{n}-1} e^{-\frac{x}{\lambda}}, \tag{3.3}$$

where Γ is the Gamma function, such that $\Gamma(\beta) = \int_0^\infty x^{\beta-1} e^{-x} dx$.

Algorithm 10 contains the details of perturbing the triangle count using dis-

tributed noise. It takes as input the encoded triangle count $\langle T \rangle = \{\langle T \rangle_1, \langle T \rangle_2\}$, noisy maximum degree d'_{max}, and privacy budget ε_2. According to Lemma 3, one $Lap(.)$ random variable can be obtained by summing up $2n$ random variables. Each user v_i samples two random variables Gam_1 and Gam_2 from Gamma distribution, and then obtains a partial noise by computing $\gamma_i = (Gam_1 - Gam_2)$ (line 2-4). To avoid the size of noise, each user encodes γ_i using additive secret sharing, i.e., $\gamma_i = \langle \gamma_i \rangle_1 + \langle \gamma_i \rangle_2$, and then sends the shares to two servers (line 5-6). Server S_1, S_2 collects all secret shares of the partial noise and aggregates them as: $\langle \gamma \rangle_1 = \sum_{i=1}^{n} \langle \gamma_i \rangle_1$ and $\langle \gamma \rangle_2 = \sum_{i=1}^{n} \langle \gamma_i \rangle_2$, respectively (line 7-8). Each server adds the secret share of noise into the share of triangle count, namely, $\langle T' \rangle_1 = \langle T \rangle_1 + \langle \gamma \rangle_1$ and $\langle T' \rangle_2 = \langle T \rangle_2 + \langle \gamma \rangle_2$ (line 9-10). After communicating with each other, server S_1, S_2 can obtain the final noisy triangle counting result T'. This algorithm is denoted by Perturb.

3.4 Theoretical Analysis

3.4.1 Security and Privacy Analysis

Security Analysis

We first analyze the security of our proposed Algorithm 9 and Algorithm 10. Following the simulation-based paradigm [82], we prove the security guarantee by giving a simulator, ensuring that simulator's view and each server's real view are computationally indistinguishable.

Definition 8 *Let* Π *denote the protocol in the semi-honest and non-colluding scenario. Let* $\mathsf{view}_{S_i}^{\Pi}$ *be the view of the server* S_i*. Let* Sim_{S_i} *be the view of a simulator. The execution of* Π *is secure if* $\mathsf{view}_{S_i}^{\Pi}$ *and* Sim_{S_i} *are computationally indistinguishable, namely,* $\mathsf{view}_{S_i}^{\Pi} \approx \mathsf{Sim}_{S_i}$.

Theorem 5 *Given the security of additive secret sharing, our Algorithm 9 and Algorithm 10 are secure according to Definition 8.*

For multiplication in Algorithm 9 and addition in Algorithm 10 via additive secret sharing, the random split of secret values guarantees that both the simulator view and the real view are identical. Each server obtains no information about user's sensitive information, where the information leakage from aggregation result is bounded and qualified via formal DP (refer to *Privacy Analysis*).

Table 3.2. Summary of Theoretical Results.

	$\mathsf{CentralLap}_\triangle$	CARGO	$\mathsf{Local2Rounds}_\triangle$
Server	Trusted	Untrusted	Untrusted
Privacy	ε-Edge CDP	$(\varepsilon_1+\varepsilon_2)$-Edge DDP	ε-Edge LDP
Utility	$O(\frac{d^2_{max}}{\varepsilon^2})$	$O(\frac{d'^2_{max}}{\varepsilon_2^2})$	$O(\frac{e^\varepsilon}{(e^\varepsilon-1)^2}(d^3_{max}n+\frac{e^\varepsilon}{\varepsilon^2}d^2_{max}n))$
Time Complexity	$O(1)$	$O(n^3)$	$O(n^2+nd^2_{max})$

Privacy Analysis

Then, we present privacy guarantee of Algorithm 7 and Algorithm 6.

Theorem 6 *Algorithm 7 satisfies ε_1-Edge LDP.*

Proof of Theorem 6. Let A_i and A'_i be two neighboring adjacent lists that differ in one edge, and d_i and d'_i be their node degrees respectively. Clearly, $|d_i - d'_i| = 1$. Let the noise x and x' are two random values drawn from $Lap(\frac{1}{\varepsilon_1})$, the probability of outputting the same noisy degree d' can be bounded by:

$$\frac{Pr[d' = d_i + x]}{Pr[d' = d'_i + x']} = \frac{Pr[x = d' - d_i]}{Pr[x' = d' - d'_i]}$$
$$= \frac{e^{-\varepsilon_1 . |d' - d_i|}}{e^{-\varepsilon_1 . |d' - d'_i|}} = e^{\varepsilon_1 . (|d' - d'_i| - |d' - d_i|)} \leq e^{\varepsilon_1 |d_i - d'_i|} = e^{\varepsilon_1},$$

which proves that d_i satisfies ε_1-Edge LDP. Then, according to the post-processing property [54], Algorithm 7 satisfies ε_1-Edge LDP.

Theorem 7 *Algorithm 6 satisfies $(\varepsilon_1+\varepsilon_2)$-Edge DDP (Definition 6).*

Proof of Theorem 7. Algorithm 6 uses two privacy budgets: ε_1 in Max and ε_2 in $\mathsf{Perturb}$. By Theorem 6, Max provides ε_1-Edge LDP. Below, we analyze the privacy of $\mathsf{Perturb}$.

Let G and G' be two neighboring graphs that differ in one edge, and $T(G)$ and $T(G')$ are their corresponding triangle counts respectively. The sensitivity of triangle counting is denoted by $|T(G) - T(G')| = \triangle$. Let $\langle T\rangle_1$ and $\langle T\rangle_2$ be two secret shares for true triangle count T. Then, we have

$$T(G) = \langle T\rangle_1 + \langle T\rangle_2,\ T(G') = \langle T'\rangle_1 + \langle T'\rangle_2$$

Let $r = \{r_1, ..., r_n\}$ and $r' = \{r'_1, ..., r'_n\}$ be two sets of distributed noise from $Gam_1(n, \frac{\triangle}{\varepsilon_2}) - Gam_2(n, \frac{\triangle}{\varepsilon_2})$. Let x and x' be two random variables sampled from $Lap(\frac{\triangle}{\varepsilon_2})$ distribution. According to the infinite divisibility of *Laplace* distribution (Lemma 3), x and x' can be denoted by:

$$x = r_1 + ... + r_n,\ x' = r'_1 + ... + r'_n$$

The probability of outputting the same noisy triangle count $\widetilde{T}$ can be bounded by:

$$\begin{aligned}
&\frac{Pr[\widetilde{T} = \langle T\rangle_1 + \langle T\rangle_2 + (r_1 + ... + r_n)]}{Pr[\widetilde{T} = \langle T'\rangle_1 + \langle T'\rangle_2 + (r'_1 + ... + r'_n)]} \\
=&\frac{Pr[\widetilde{T} = T + (r_1 + ... + r_n)]}{Pr[\widetilde{T} = T' + (r'_1 + ... + r'_n)]} \\
=&\frac{Pr[\widetilde{T} = T(G) + x]}{Pr[\widetilde{T} = T(G') + x']} = \frac{Pr[x = \widetilde{T} - T(G)]}{Pr[x' = \widetilde{T} - T(G')]} \\
=&\frac{e^{\frac{-\varepsilon_2 \cdot |\widetilde{T} - T(G)|}{\triangle}}}{e^{\frac{-\varepsilon_2 \cdot |\widetilde{T} - T(G')|}{\triangle}}} = e^{\frac{\varepsilon_2 \cdot (|\widetilde{T} - T(G')| - |\widetilde{T} - T(G)|)}{\triangle}} \\
\leq& e^{\frac{\varepsilon_2 |T(G) - T(G')|}{\triangle}} = e^{\varepsilon_2}
\end{aligned}$$

Thus, Perturb provides ε_2-Edge DDP. As described in Section 3.2.2, both ε-Edge LDP and ε-Edge DDP protect one edge with privacy budget ε. Thus, the entire process of Algorithm 6 provides $(\varepsilon_1 + \varepsilon_2)$-Edge DDP.

3.4.2 Utility Analysis

The utility error of CARGO is mainly from the projection loss in Algorithm 8 and perturbation error in Algorithm 10.

Theorem 8 (Projection Loss) *Let $T(G)$ and $\hat{T}(G, d'_{max})$ be the triangle count before and after projection respectively. The projection parameter is set as the noisy maximum degree d'_{max}. Then, for any $d'_{max} \geq 0$ and G,*

$$\mathbb{E}[l_2^2(T(G), \hat{T}(G, d'_{max}))] = (T(G) - \hat{T}(G, d'_{max}))^2$$

Theorem 9 (Perturbation Error) *Let $T'(G, \varepsilon_2, d'_{max})$ and $\hat{T}(G, d'_{max})$ be the triangle count before and after perturbation, respectively. Then, for any privacy budget $\varepsilon_2 \geq 0$, noisy maximum degree $d'_{max} \geq 0$, and graph G,*

$$\mathbb{E}[l_2^2(T'(G,\varepsilon_2,d'_{max}),\hat{T}(G,d'_{max}))] = O(\frac{d'^2_{max}}{\varepsilon_2^2})$$

Proof of Theorem 9. According to the well-known bias-variance decomposition [83], the expected l_2 loss consists of the bias and variance, which can be written as follows:

$$\begin{aligned}&\mathbb{E}[l_2^2(T'(G,\varepsilon_2,d'_{max}),\hat{T}(G,d'_{max}))]\\=&(\mathbb{E}[T'(G,\varepsilon_2,d'_{max})]-\hat{T}(G,d'_{max}))^2+\mathbb{V}[T'(G,\varepsilon_2,d'_{max})]\end{aligned}$$

Since the mean of Laplacian noise $Lap(\frac{\triangle}{\varepsilon_2})$ is 0, the estimation $T'(G,\varepsilon_2,d'_{max})$ is unbiased. Then, the expected l_2 loss will be equal to the variance, which can be formalized as:

$$\begin{aligned}&\mathbb{E}[l_2^2(T'(G,\varepsilon_2,d'_{max}),\hat{T}(G,d'_{max}))]=\mathbb{V}[T'(G,\varepsilon_2,d'_{max})]\\=&\mathbb{V}[Lap(\frac{d'_{max}}{\varepsilon_2})]=O(\frac{d'^2_{max}}{\varepsilon_2^2})\end{aligned}$$

In fact, after empirical analysis, we find that $d'_{max} \approx d_{max}$, as presented in Table 3.5. Specifically, we find that $l_2^2(d'_{max}, d_{max}) < 0.009 d_{max}$, where d_{max} is the true maximum degree. In addition, $d'_{max} \geq d_{max}$ holds in most cases, which means that there is no deletion of edges during a projection and $\mathbb{E}[l_2^2(T(G),\hat{T}(G,d'_{max}))] = 0$. For a convenient comparison, we omit the projection loss in theoretical analysis. Thus, our CARGO attains the expected l_2 loss of $O(\frac{d'^2_{max}}{\varepsilon_2^2})$, where $\varepsilon_2 = 0.9\varepsilon$ in our setting.

Discussion. It is worth noting that the private maximum degree is the upper bound of local sensitivity, and there are some advantages and disadvantages. On the one hand, it provides a privacy guarantee for the worst case at the expense of utility. For instance, if we only consider bipartite graphs, the result of triangle counting is always 0. The local sensitivity is 0 while the maximum degree can be very large, which leads to much error. Recently, some works [84, 85] utilize the smooth sensitivity (SS) and residual sensitivity (RS) for common graph queries, which can achieve constant noise. Furthermore, we compare the value of d'_{max} with values of SS and RS in Table 1 in [84], as presented in Table 3.3. We find that d'_{max} can be larger than SS and RS in some graphs, for example, CondMat and HepTh, which means that our approach can add much more noise than SS and RS. On the other hand, our d'_{max} uses the Laplace distribution and the expected

Table 3.3. Comparison between SS, RS, and d'_{max}.

Graph	CondMat	AstroPh	HepPh	HepTh	GrQc
d'_{max}	560	1,008	984	130	162
SS	489	1,050	1,350	102	183
RS	493	1,054	1,354	205	222

Privacy budget $\varepsilon = 1$

l_2 loss (= variance) is finite. However, considering that RS can be regarded as an instantiation of SS [84], both SS and RS draw noise from a Cauchy distribution, which has an infinite variance.

3.4.3 Time Complexity

Let n and d_{max} denote the number of users and the maximum degree, respectively. In CARGO, the time complexity of Max in Algorithm 7 is $O(n)$. Then, the time complexity of project in Algorithm 8 is $O(nd_{max})$ since one user needs to compute degree similarities (Lemma 7) between her and her all neighboring users. Next, the time complexity of Count in Algorithm 9 is $O(n^3)$. This is because CARGO needs to traverse all triples to justify if three edges of a triple exist simultaneously. Finally, the time complexity of Perturb in Algorithm 10 is $O(n)$, and the time complexity of secret sharing and aggregation is also $O(n)$. Thus, the time complexity of our CARGO is $O(n^3)$.

Summary of Theoretical Results. Table 3.2 summarizes the theoretical results compared with the state-of-the-art methods, namely, CentralLap$_\triangle$ [5] and Local2Rounds$_\triangle$ [5]. The results of CentralLap$_\triangle$ and Local2Rounds$_\triangle$ have been analyzed (refer to Table 2 in [5]). On the one hand, CARGO can achieve a high-utility triangle count comparable to CentralLap$_\triangle$, but it does not necessarily have a trusted server. On the other hand, CARGO outperforms much better than Local2Rounds$_\triangle$ in terms of utility.

3.5 Experimental Evaluation

In this section, we conduct experiments to answer the following questions:

- **Q1**: What is the utility-privacy trade-off of our CARGO compared with the state-of-the-art CDP-based and LDP-based protocols?

Table 3.4. details of graph datasets.

Graph	$\|V\|$	$\|E\|$	d_{max}	Domain
Facebook	4,039	88,234	1,045	social network
Wiki	7,115	103,689	1,167	vote network
HepPh	12,008	118,521	982	citation network
Enron	36,692	183,831	2,766	communication network

Table 3.5. Noisy maximum degrees under various ε.

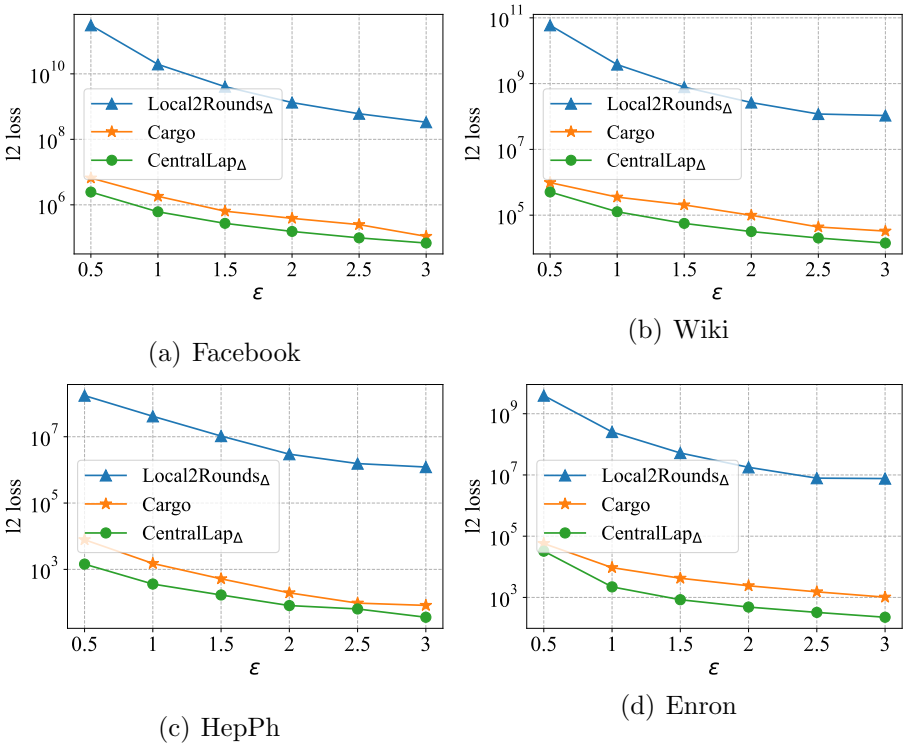
Graph \ ε	0.5	1	1.5	2	2.5	3
Facebook	1079	1063	1047	1037	1052	1047
Wiki	1153	1166	1173	1213	1155	1167
HepPh	967	1013	975	983	979	981
Enron	2834	2764	2754	2777	2767	2762

- **Q2**: How does our similarity-based projection method (Project algorithm) outperform the existing projection method?
- **Q3**: How much running time does our CARGO take compared with competitors?

3.5.1 Experimental Setting

We use four real-world graph datasets from SNAP [41], and all graphs are preprocessed into undirected and symmetric graphs. These graphs are from different domains and have different scales. Table 3.4 presents more details about each graph G, including the number of nodes $|V|$, the number of edges $|E|$, the true maximum degree d_{max}, and domains that graphs belong to. For each algorithm, we evaluate the l_2 loss and relative error while varying the number of users n and privacy budget ε. The default values of n and ε are 2×10^3 and 2, respectively. Usually, triangle counting needs more privacy budget than the other information (i.e., d_{max}) [5, 23]. Thus, we set $\varepsilon_1 = 0.1\varepsilon$ for publishing noisy maximum degree d'_{max} and $\varepsilon_2 = 0.9\varepsilon$ for perturbing triangles.

Competitors. To justify the utility-privacy tradeoff of our CARGO system, we compare it with the state-of-the-art CDP and LDP methods: (1) $\mathsf{CentralLap}_{\triangle}$ [5], an algorithm for triangle counting using Laplace mechanism in CDP. (2) $\mathsf{Local2Rounds}_{\triangle}$ [5], a two-round algorithm for counting triangles in LDP. To verify



(a) Facebook

(b) Wiki

(c) HepPh

(d) Enron

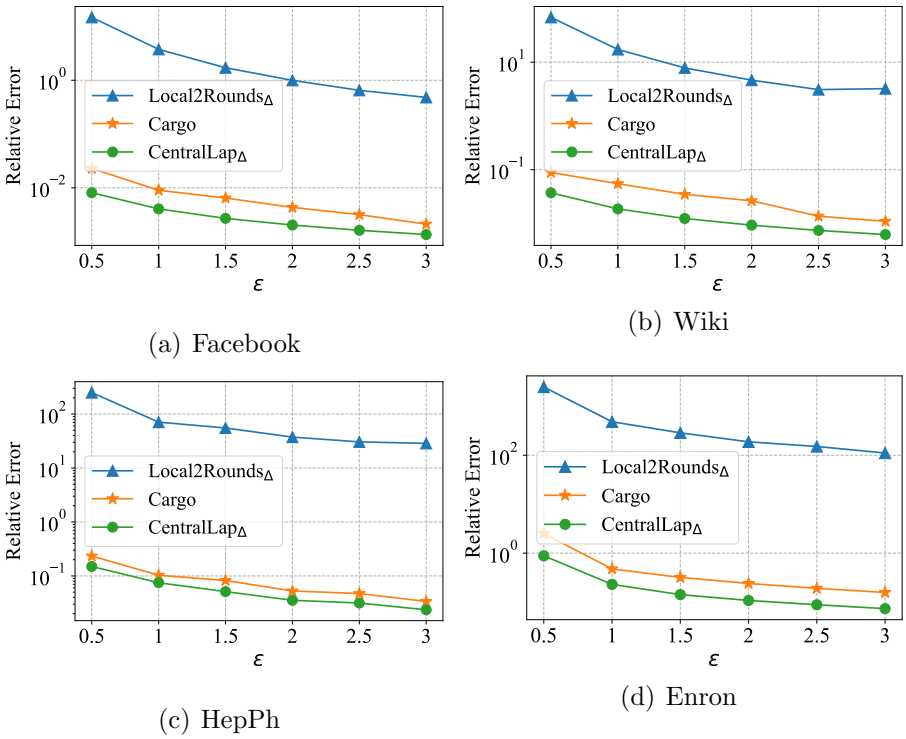
Figure 3.5. The l_2 loss of triangle counting with ε varying from 0.5 to 3.

the performance of our local projection algorithm (i.e., Project in Section 3.3.3), we compare it with the existing projection method in LDP (i.e., GraphProjection in [5]).

3.5.2 Experimental Results

Utility-privacy trade-off. We evaluate **Q1** by comparing the accuracy of our CARGO with that of the aforementioned state-of-the-art LDP and CDP methods when the privacy budget ε varies from 0.5 to 3. We also evaluate the accuracy of our Max algorithm in Section 3.3.3 that privately estimates the noisy maximum degree d'_{max}. As shown in Table 3.5, d'_{max} approaches the true maximum degree d_{max} in Table 3.4. For each graph, the average of relative error between d_{max} and d'_{max} is less than 1%. Thus, we allow an additional round to estimate d'_{max} in our experiments.

Fig. 3.5 and Fig. 3.6 present that CARGO significantly outperforms the LDP model Local2Rounds$_\triangle$ in all cases. For example, Fig. 3.5(a) shows that for



(a) Facebook

(b) Wiki

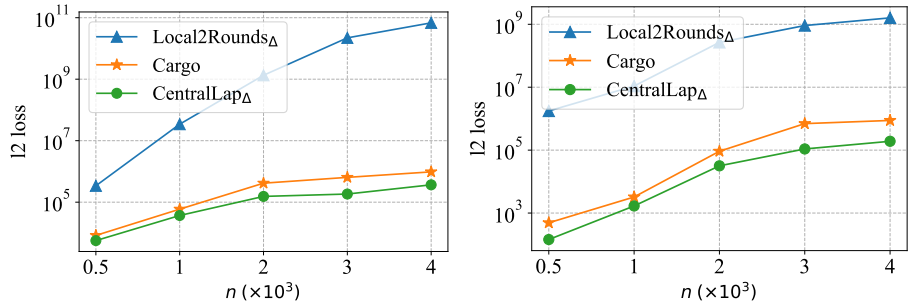
(c) HepPh

(d) Enron

Figure 3.6. The relative error of triangle counting with ε varying from 0.5 to 3.

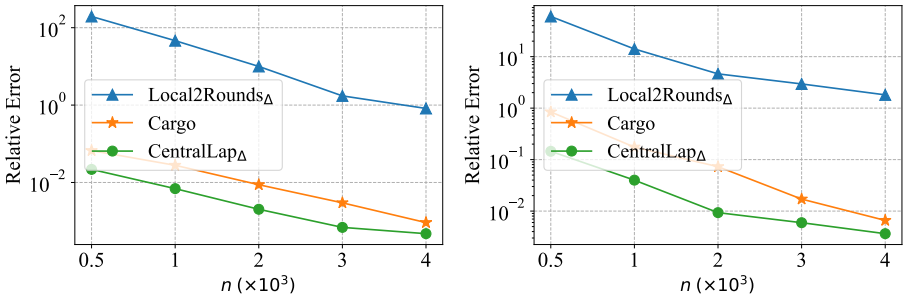
CARGO, $\varepsilon = 3$ results in l_2 loss of 1.09×10^5 as compared to an error of 3.33×10^8 achieved by Local2Rounds$_\triangle$. Fig. 3.5(c) illustrates that for $\varepsilon = 3$, CARGO owns a l_2 loss of 82 while Local2Rounds$_\triangle$ has an error of 1.22×10^6. Similarly, in Fig. 3.6(a), CARGO gives a relative error of only 2.11×10^{-3} when $\varepsilon = 3$. In contrast, Local2Rounds$_\triangle$ has a relative error of 0.48. Fig. 3.6(c) shows that Local2Rounds$_\triangle$ has a relative error of 28.6 while CARGO only has an error of 3.37×10^{-2} for $\varepsilon = 3$. Thus, CARGO improves the Local2Rounds$_\triangle$ significantly.

Another observation is that the error of CARGO is around $1\times \sim 2\times$ larger than that of CentralLap$_\triangle$ for most of cases. With the increase of ε, the error gap between CARGO and CentralLap$_\triangle$ roughly decreases. For example, in Fig. 3.6(a), CARGO has a relative error of 2.29×10^{-2} as CentralLap$_\triangle$ has an error of 8.10×10^{-3} when $\varepsilon = 0.5$. Similarly, when $\varepsilon = 3$, Fig. 3.6(a) also shows that the relative error of CARGO is 2.11×10^{-3} when CentralLap$_\triangle$ owns a relative error of 1.35×10^{-3}. This is intuitive because an additional round for privately estimating d_{max} (Max algorithm) leads to a little error (as shown in Table 3.5), influencing the overall utility. On the other hand, increasing ε improves the accuracy of



(a) Facebook (b) Wiki

Figure 3.7. The l_2 loss of triangle counting with different n.



(a) Facebook (b) Wiki

Figure 3.8. The relative error of triangle counting with different n.

Max. Therefore, the error of CARGO is not significantly larger than that of CentralLap$_\triangle$.

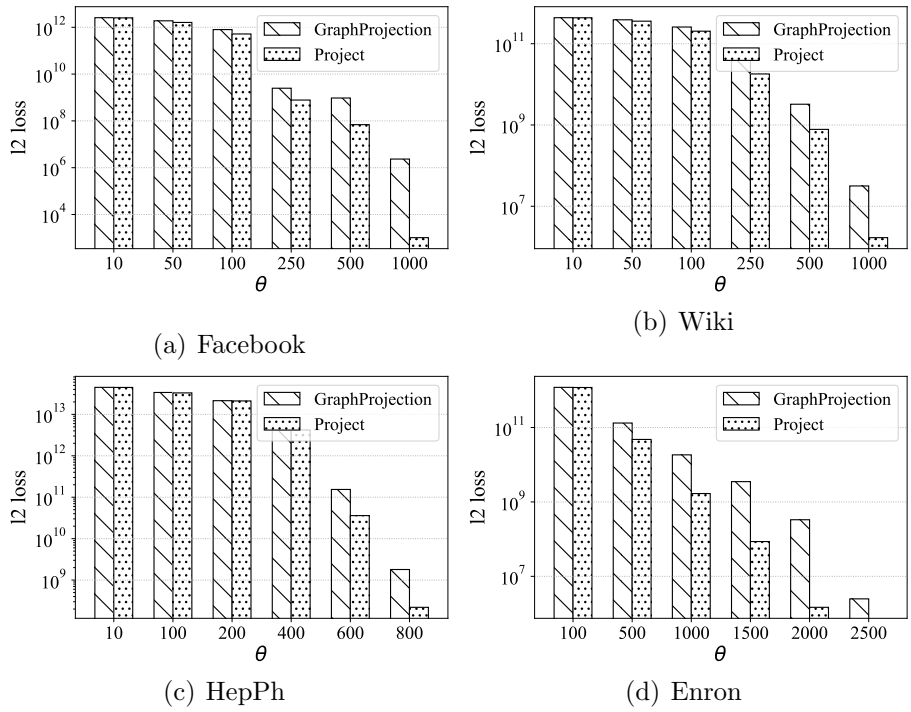
Fig. 3.7 and Fig. 3.8 show that CARGO outperforms Local2Rounds$_\triangle$ and achieves the high accuracy comparable to CentralLap$_\triangle$ while varying n ($\varepsilon = 2$). Here, we just present the results of two graphs due to the limited space. To be specific, in Fig. 3.7(a), the l_2 loss of CARGO is 9.68×10^5 when $n = 4000$. In contrast, Local2Rounds$_\triangle$ owns an l_2 loss of 6.8 $\times10^{10}$. Furthermore, in Fig. 3.8(b), CARGO has a relative error of 6.59×10^{-3} when CentralLap$_\triangle$ has an error of 3.64×10^{-3} for $n = 4000$. Therefore, CARGO significantly outperforms Local2Rounds$_\triangle$ and performs similarly to CentralLap$_\triangle$.

Local graph projection. Next, we evaluate **Q2** by performing a comparative analysis between our local projection method (Project) and the existing projection method in local settings (GraphProjection). We set projection parameter θ from 10 to 1000, and compute the projection loss by comparing triangle counts before and after projection. Fig. 3.9 and Fig. 3.10 show that Project owns better

utility than the baseline GraphProjection in all cases. On the other hand, when θ increases, the projection loss for both of them decreases and the improvement of Project becomes more significant. For example, in Fig. 3.9(a), the l_2 loss of GraphProjection is 1.01× larger than that of Project when $\theta = 10$; In contrast, for $\theta = 1000$, the l_2 loss of Project is at least 8× less than that of GraphProjection. This is roughly consistent with our theoretical analysis in Section 3.3.3. Namely, randomly deleting edges in GraphProjection is likely to remove some key edges that involve in many triangles, resulting in losing many triangles in a graph.

Running time. Finally, we evaluate **Q3** by testing the performance of CARGO and competitors while changing n. Due to limited space, we only present the results of two graphs. Fig. 3.11 and Fig. 3.12 show the execution time over Facebook and Wiki, respectively. We can observe that the time cost of all methods grows with graph size n increases, which is consistent with our theoretical analysis in Section 3.4. The running time of $\mathsf{Local2Rounds}_{\triangle}$ is approximately 2× higher than that of $\mathsf{CentralLap}_{\triangle}$. This is because $\mathsf{Local2Rounds}_{\triangle}$ needs an additional round for collecting a noisy global graph. Another important observation is that CARGO needs more time overhead than the other two methods. For example, in Fig. 3.11, CARGO takes the time of 485s while $\mathsf{Local2Rounds}_{\triangle}$ takes 0.235s and $\mathsf{CentralLap}_{\triangle}$ only takes 0.105s. After further evaluation, we find that most of time overhead in CARGO is from the computation of secure triangle counting, namely, Count in Algorithm 9. As shown in Fig. 3.12, the execution time of Count accounts for at least 90% of overall running time. This is because CARGO needs to traverse all triples and the time complexity of triangle counting is up to $O(n^3)$.

Summary of Experimental Results. In summary, our answers to three questions at the start of Section 3.5 are as follows. **Q1**: Our CARGO outperforms significantly than $\mathsf{Local2Rounds}_{\triangle}$ and achieves comparative accuracy to $\mathsf{CentralLap}_{\triangle}$ without a trusted server. **Q2**: Our Project algorithm significantly reduces the projection loss compared with the existing GraphProjection. **Q3**: The running time of CARGO is higher than that of other methods and most of computation overhead is from Count.



(a) Facebook (b) Wiki

(c) HepPh (d) Enron

Figure 3.9. The l_2 loss of projection with various parameters.

3.6 Related Works

3.6.1 Triangle Counting in DP

Differentially private triangle counting has been widely studied, and previous works are mainly based on either central DP (CDP) or local DP (LDP).

Triangle Counting in CDP. Existing works related to triangle counting in the central model mainly focus on how to reduce the global sensitivity. Ding et al. [2] propose novel projection methods, namely, two edge-deletion strategies (DL and DS) for triangle counts distributions (histogram of triangle counts and cumulative histogram of triangle counts) while satisfying Node DP. Karwa et al. [22] give a differentially private algorithm for releasing the triangle counts based on the higher-order local sensitivity, which adds less noise than methods with global sensitivity. Kasiviswanathan et al. [15] develop algorithms for private graph analysis under Node DP. They design a projection operator that projects the input graph into a bounded-degree (low-degree) graph, resulting in a lower sensitivity. But these works assume that there is a trusted server that owns the

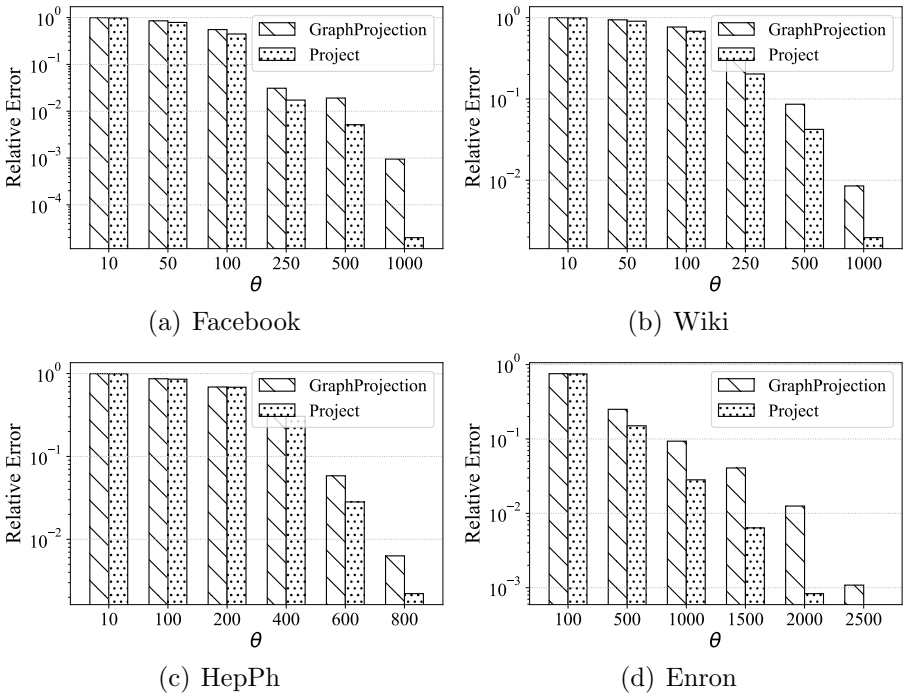


Figure 3.10. The relative error of projection with various parameters.

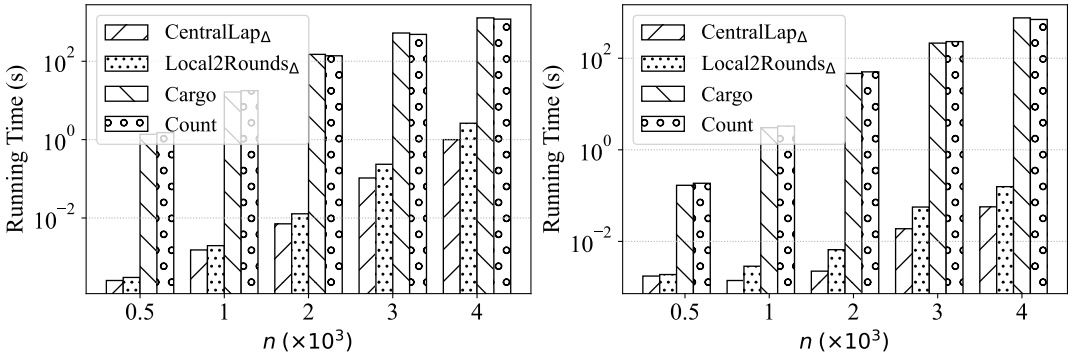


Figure 3.11. Running time on Facebook.

Figure 3.12. Running time on Wiki.

entire graph, which may not be practical in many applications due to the risk of privacy leaks [28].

Triangle Counting in LDP. Triangle counting in LDP has recently attracted much attention. The main challenge is from the complex inter-dependencies involving multiple people. For example, each user cannot see edges between other users. Imola et al. [5] propose a two-round interaction for triangle counts

to handle the above challenge. However, local randomization aggregates many errors and two-round interaction brings additional communication overhead. Sun et al. [23] assume that each user has an extended view, and thus each user can see the third edges between others. Different from this, we make a minimal assumption where each user only knows her friends.

In a nutshell, there is an apparent tradeoff between CDP-based models and LDP-based models in terms of privacy and utility while calculating the triangles.

3.6.2 Crypto-assisted DP

The approach of using cryptographic tools to enhance the utility of differential privacy (we call it as *crypto-assistend DP*) has been studied in the literature [55–62]. He *et al.* [55] compose differential privacy and two-party computation for private record linkage while ensuring three desiderata: correctness, privacy, and efficiency. Bohler et al. propose Cryptε [56] is a system and a programming framework for supporting a rich class of state-of-the-art DP programs, and it achieves the accuracy of CDP without a trusted server. Honeycrisp [57] combines cryptographic techniques and differential privacy for answering periodic queries, which can sustainably run queries like the one from Apple's deployment while protecting user privacy in the long run. Some works such as [58–61] combine cryptography and differential privacy for federated learning, reducing the injected noise without sacrificing privacy. Fu et al. [62] proposes a crypto-assisted differentially private framework for hierarchical count histograms under untrusted servers. These protocol, therefore, achieve nearly the same accuracy as CDP models with untrusted servers. Although the crypto-assisted DP model has been applied to tabular data [55–57, 62] and gradients in federated learning [58–61], characteristics of graph data, such as high-dimensional and inter-correlated, lead to significant challenges (discussed in Section 3.1). Some works [86, 87] focus on securely computing the triangles using cryptographic tools, which is orthogonal to the formal privacy guarantee provided by Edge DDP. To our knowledge, our work is the first work for triangle counting using crypto-assisted DP model.

3.7 Conclusions

To conclude, we propose the first crypto-assisted differentially private graph analysis framework, CARGO, which achieves high-utility triangle counting comparable to CDP-based models but without requiring a trusted server like LDP-based models. Through theoretical and experimental analysis, we verify the privacy and utility achieved by our framework.

CHAPTER 4

Crypto-Assisted Differentially Private Federated Graph Analytics

4.1 Introduction

Graph data has become a crucial resource for analyzing big data in a variety of applications such as finance, social networks, and healthcare due to its widespread usage. Owing to escalating privacy concerns and regulatory measures like the GDPR, conducting centralized graph analysis has become increasingly challenging. In this paper, we define the *federated graph analytics* (FGA), a new problem for collaborative graph analysis with the privacy guarantee, which is motivated by the following scenarios:

Example 1.1. Social Network Analytics. Various social media platforms, including Facebook, Twitter, and LINE, collaborate to estimate different metrics of a global social network within a particular region. Each platform has its own local subgraph, which is a subset of a ground-truth global social network graph. In a graph, a node represents a user, and an edge represents a friendship between two users. As users may use multiple platforms, these clients (i.e., subgraphs) may have *overlapping* edges.

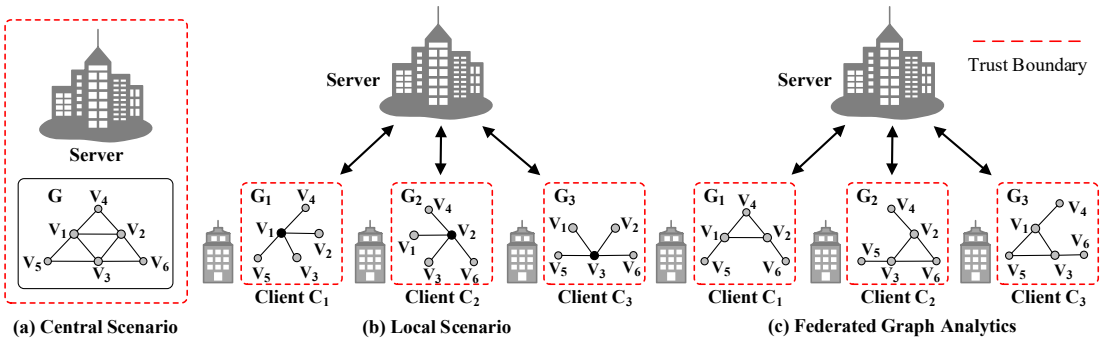


Figure 4.1. Comparisons among central, local and federated scenarios. (a) In a central scenario [1–4], one trusted server owns the entire graph. (b) In a local scenario [5–8], each client owns one node and its 1-*hop* path information. (c) In a federated scenario, each client owns a subgraph that consists of multiple nodes and edges among them.

Example 1.2. Financial Transaction Analytics. Several banks work together to analyze transaction data [88] for financial risk management or macroeconomic analysis over transaction graphs, in which each node represents a bank account owned by a user, and each edge represents a money transaction between two accounts.

Example 1.3. Disease Transmission Analytics. Several medical institutions are collaborating to study the transmission of diseases, such as COVID-19 [89], in a particular region. Each hospital is responsible for a subgraph that includes nodes representing patients and edges representing the transmission of the disease between those patients.

This study is the first to discuss the problem of *federated graph analytics* (FGA) under the *differential privacy* (DP) [14, 54]. Different from existing differentially private graph analytic works, such as central models [1–4, 24] and local models [5–8,25], FGA considers a more general setting. In particular, in a central scenario (as shown in Figure 4.1(a)), a trusted server owns the entire global graph that consists of multiple nodes and edges. Nevertheless, a central server is amenable to privacy issues in practice, such as data leaks and breaches [90, 91]. In a local scenario (as shown in Figure 4.1(b)), each client manages an user and her 1-*hop* path information (i.e, neighboring information). Each client doesn't trust the server and directly perturbs local sensitive data. In contrast, in federated graph analytics (as shown in Figure 4.1(c)), each client possesses a subgraph consisting

of multiple nodes and edges. Each client does not trust other parties, including the server and other clients. In fact, local scenarios can be viewed as an extreme case of federated graph analytics when each client contains a subgraph consisting of one user and her 1-hop path. At this point, m is equal to n, where m and n are the number of clients and users, respectively. Additionally, FGA is similar to cross-silo federated learning [77, 92–94] where different silos (or clients) collaboratively train machine learning models without collecting the raw data. Nevertheless, federated learning focuses on optimization-based questions (i.e., learning models) that differ from graph statistics.

Although differentially private graph analysis has been widely studied [1–8, 24, 25], this does not apply to FGA due to the following reasons. On the one hand, the *limited view* of each local client leads to utility issues. Each client only possesses a portion of the entire graph, making it hard to calculate accurate statistics. For instance, if a query task Q is to count triangles, each client in Figure 4.1(c) returns the answer $Q_i = 1$. Although their sum is 3, the true answer is 4. This discrepancy happens because the edges of the triangle $\langle v_1, v_2, v_3 \rangle$ come from three different clients. Consequently, it is impossible for any individual client to obtain the ground truth answer. On the other hand, *overlapping information* among different subgraphs causes privacy issues. An edge may exist in multiple subgraphs owned by different clients. In Figure 4.1(c), for example, the edge $\langle v_1, v_4 \rangle$ appears in both client C_1 and client C_3. Although each client can individually provide sufficient privacy guarantees for $\langle v_1, v_4 \rangle$, multiple reports of the same information amplify the probability of distinguishing such an edge multiple times, leaking the edge privacy.

In this paper, we propose a **fe**derated gr**a**ph analy**t**ic framework, named FEAT, which enables arbitrary downstream common graph statistics while preserving individual privacy. The main idea is to let the server privately collect the subgraph information from local clients and then aggregate a noisy global graph for executing targeted query tasks, thereby overcoming the limited view problem. To avoid collecting the same edge multiple times, FEAT leverages the private set union (PSU) technique [95–98] to aggregate the subgraph information. However, existing multi-party private set union protocols do not satisfy DP. Hence, we design a differentially private set union (DPSU) algorithm, which ensures that the sensitive information is reported only once and the output global graph is protected under DP.

Moreover, we observe that there is still room for improving the accuracy by leveraging true local subgraphs. To this end, we introduce an improved framework FEAT+ that allows additional communication between the server and clients. In FEAT+, each client reports the intermediate answer based on its local subgraph and the global graph. However, a key challenge arises from the possibility of different clients reporting the same edge multiple times, thereby compromising individual privacy. To mitigate this risk, we devise a degree-based node partition method to partition entire nodes into multiple disjoint sets. Consequently, the query answer associated with each set is collected only once.

In summary, our contributions in this work are elaborated as follows:

- We investigate the federated graph analytics (FGA) under DP for the first time. By comparing with previous protocols, we conclude unique challenges in FGA.
- We present a generalized federated graph analytic framework with differential privacy (FEAT) based on our proposed DPSU protocol, which supports a wide range of common graph statistics, e.g., subgraph counting.
- We introduce an optimized framework (FEAT+) based on our proposed degree-based partition algorithm, which improves the overall utility by leveraging true subgraphs.
- We verify the effectiveness of our proposed methods through extensive experiments. FEAT reduces the error than baseline approach by up to an order of 4. FEAT+ outperforms FEAT by at least an order of 1.

Section 4.2 introduces the preliminaries. Our generalized framework FEAT and improved framework FEAT+ are proposed in Section 4.3 and Section 4.4. Section 4.5 presents experimental results. Section 4.6 reviews the related work and Section 4.7 draws a conclusion.

4.2 Preliminary

4.2.1 Problem Formulation

System Model

In our work, we consider undirected, unattributed graphs, represented as $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ is the set of nodes, and $E \subseteq V \times V$ is the set of edges. We study the common graph statistics in cross-silo federated scenario, where there are an untrusted server and m silos, i.e., clients $C = \{C_1, ..., C_m\}$. Each client C_i owns a subgraph G_i, which is represented as a $n \times n$ adjacent matrix. The virtual global graph is the union of all subgraphs, which can be represented as $G = \bigcup_{i=1}^{m} G_i$. It is worth noting that each client is mutually independent of others and there may exist overlapping information among different subgraphs, denoted as $G_i \cap G_j \neq \emptyset$, where $i \neq j$. Clients collaboratively support graph queries over their subgraph data while preserving user privacy. Table 4.1 lists the major notations used in this paper.

Trust Assumption

Our objective is to create a protocol that enables the server to coordinate graph statistics while ensuring that none of the clients' sensitive information is disclosed. Similar to prior works [77, 99, 100], we assume that clients are *semi-honest*. In other words, each client follows the protocol honestly but is curious about the sensitive information on other clients. The server is untrusted and has no access to individual sensitive information. Furthermore, we presume that any parties beyond the system, such as servers, analysts, or other individuals, are adversaries who are computationally constrained.

4.2.2 Privacy Model

Like previous works [5–8, 25], the private information considered in this study is the edge privacy. We assume that the server knows the node information, i.e., $V = \{v_1, ..., v_n\}$, which makes sense in some real-world applications. For example, consider that the healthy administration is examining the spread of COVID-19 in a certain region. It collects the disease transmission paths according to the released census in this area.

Table 4.1. Summary of Notations.

Notation	Definition
G	True global graph
V	Node set
E	Edge set
G'	Noisy global graph
C	A set of all clients
m	Number of clients in C
C_i	The i-th client
G_i	Subgraph of client C_i
n	Number of nodes in G
D'	Noisy degree sequence
S'_k	Noisy k-stars counts
T'	Noisy triangle counts

Differential privacy (DP) [14,54] has become a de-facto standard for preserving individual privacy, which can be formalized in Definition 9. In our work, we use global sensitivity [54] to achieve the DP, defined as Definition 10. It considers the maximum difference between statistic results on two neighboring graphs.

Definition 9 (Differential Privacy [54]) *Let $\varepsilon > 0$ be the privacy budget and n be the number of users. A randomized algorithm $\mathcal{M}$ with domain $\mathbb{D}^n$ satisfies ε-DP, iff for any neighboring datasets $D, D' \in \mathbb{D}^n$ that differ in a single user's data and any subset $S \subseteq Range(\mathcal{M})$,*

$$Pr[\mathcal{M}(D) \in S] \leq e^{\epsilon} Pr[\mathcal{M}(D') \in S],$$

Definition 10 (Global Sensitivity [54]) *For a query function $f: D \to \mathbb{R}$, the global sensitivity is defined by*

$$\triangle_{GS} = \underset{D \sim D'}{max} |f(D) - f(D')|,$$

where D and D' are neighboring databases that differ in a single user's data.

The Laplace mechanism is one of common techniques to achieve DP. The formal definition is as follows:

Definition 11 (Laplace Mechanism [101]) *Given any function $f : D \to R^k$, let $\triangle f$ be the sensitivity of function f. $M(x) = f(x) + (Y_1, ..., Y_k)$ satisfies $(\varepsilon, 0)$-differential privacy, where Y_i are i.i.d random variables drawn from Lap($\triangle/\varepsilon$).*

Definition 12 (Edge LDP [32]) *For any $i \in [n]$, let $\mathcal{M}_i$ be a randomized algorithm of user v_i. $\mathcal{M}_i$ satisfies ε-Edge LDP, iff for any two neighboring adjacent bit vectors A_i and A_i' that differ in one edge and any subset $S \subseteq Range(\mathcal{M}_i)$,*

$$Pr[\mathcal{M}_i(A_i) \in S] \leq e^{\epsilon} Pr[\mathcal{M}_i(A_i') \in S],$$

where $\varepsilon > 0$ is the privacy budget.

Existing locally differentially private models, such as *edge local differentially privacy* (Definition 12) [5–7] is a promising model. However, it fails to provide privacy guarantee in federated scenarios. While the same edge may exist in multiple different clients, each client only considers its own edge information. Multiple reports of the same information increase the probability of distinguishing such an edge multiple times, leading to privacy issues. To address this challenge, we introduce edge distributed differential privacy to achieve our privacy objectives. The formal definition is as follows:

Definition 13 (Neighboring Graphs) *Two graphs G and G' are neighboring graphs if G and G' differ in one edge.*

Definition 14 (Edge Distributed Differential Privacy (Edge DDP)) *Let $G = (V, E)$ and $G' = (V, E')$ be two neighboring global graphs. Let $C = \{C_1, ..., C_m\}$ be the client set. Let G_i and G_i' ($i \in [1, m]$) be graphs owned by client C_i in G and G', respectively. A set of randomized mechanisms $\{\mathcal{M}_i, i \in [1, m]\}$ collectively satisfy ε-Edge DDP iff. for any subsets of possible outputs $S_i \subseteq range(\mathcal{M}), i \in [1, m]$, we have the following inequality.*

$$\begin{aligned} &Pr[\mathcal{M}_1(G_1) \in S_1, ..., \mathcal{M}_m(G_m) \in S_n] \\ &\leq e^{\epsilon} \cdot Pr[\mathcal{M}_1(G_1') \in S_1, ..., \mathcal{M}_m(G_m') \in S_m]. \end{aligned}$$

Edge DDP guarantees that the server cannot distinguish the presence or absence of any edge based on all reports collected from clients. It also guarantees that the information about which client C_i an edge in E belongs to is private, if the edge exists. For example, in Figure 4.1(c), both the presence of the edge $\langle v_2, v_3 \rangle$ in G and the absence of the edge $\langle v_1, v_6 \rangle$ in G are private. Furthermore, no party knows that the edge $\langle v_2, v_3 \rangle$ belongs to clients C_2 even if the existence of $\langle v_2, v_3 \rangle$ has been disclosed.

4.2.3 Private Set Union

Private Set Union (PSU) [95–98] is a secure multiparty computation cryptographic technique designed for securely computing the union of private sets held by different parties. At its core, neither party reveals anything to the counterparty except for the elements in the union. From a high-level perspective, a typical PSU protocol involves the following steps: (1) *Set Encoding*: each party privately encodes its set into a cryptographic form suitable for secure computation: $[\![X_i]\!] \leftarrow \mathsf{Enc}(X_i), i \in [1, m]$. (2) *Union Computation*: parties engage in computing the union of their encoded sets without revealing the underlying elements: $[\![X]\!] \leftarrow \mathsf{Enc}(X_1) \cup \mathsf{Enc}(X_2)...\mathsf{Enc}(X_m)$. (3) *Result Decoding*: once the computation is complete, parties decode the computed union from its cryptographic representation to obtain the set union: $X \leftarrow \mathsf{Dec}([\![X]\!])$.

4.3 A General Framework: FEAT

In this section, we introduce our proposed general framework for federated graph analytics with differential privacy, called FEAT. The main idea is to let the server privately collect subgraphs from local clients and aggregate a noisy global graph, which facilitates common graph statistics. As shown in Figure 4.2, in general, FEAT enhances the utility by (1) reducing the added noise by introducing the crypto technique (i.e., PSU) into differentially private graph statistics; (2) calibrating the noisy results to suppress the estimation bias.

We first present a baseline approach which revises the prior protocol (e.g., randomized response) in order to satisfy our privacy goal, and discuss its limitations. Then, we introduce the overview of our proposed general framework FEAT, which substantially improves the baseline approach, and elaborate on its details in subsequent sections.

4.3.1 A Baseline Approach

Randomized response (RR) [54] is a common methodology for enhancing local differential privacy. However, it fails to provide ε-Edge DDP, as the same one edge could be reported by different clients multiple times. One edge may exist in multiple subgraphs. In the worst case, an edge is included by all subgraphs $G_i, i \in [1, m]$. Without loss of generality, assume G_i and G_i' are neighboring

Algorithm 11 Baseline: Plain Randomized Response

Input: Subgraph set $G = \{G_1, ..., G_m\}$, privacy budget ε,
G_i is represented as adjacent matrix $M_i \subseteq \{0,1\}^{n \times n}$
Output: Noisy global graph G'
Each client C_i perturbs M_i
1: **for** each bit b in M_i **do**
2:

$$b\prime = \begin{cases} b & w.p. \ \frac{e^{\varepsilon_l}}{1+e^{\varepsilon_l}} \\ 1-b & w.p. \ \frac{1}{1+e^{\varepsilon_l}} \end{cases}$$

where $\varepsilon_l = \varepsilon/m$
3: **end for**
4: Client C_i: Send the noisy subgraph G_i' to server
5: Server: $G' \leftarrow \bigcup_{i=1}^{m} G_i'$
6: **return** G'

graphs and differ in edge e_1, we have $Pr[\mathcal{M}_i(G_i) \in S_i] \leq e^{\epsilon} \cdot Pr[\mathcal{M}_i(G_i') \in S_i]$. Then, we have

$$\begin{aligned} &\frac{Pr[\mathcal{M}_1(G_1) \in S_1, ..., \mathcal{M}_m(G_m) \in S_m]}{Pr[\mathcal{M}_1(G_1') \in S_1, ..., \mathcal{M}_m(G_m') \in S_m]} \\ =&\frac{Pr[\mathcal{M}_1(G_1) \in S_1]...Pr[\mathcal{M}_m(G_m) \in S_m]}{Pr[\mathcal{M}_1(G_1') \in S_1]...Pr[\mathcal{M}_m(G_m') \in S_m]} \\ \leq&(e^{\varepsilon})^m = e^{m\varepsilon}. \end{aligned}$$

Theorem 10 Baseline *approach satisfies ε-Edge DDP.*

To address this privacy issue, we could consider the following baseline method. It divides the overall privacy budget into m portions equally. Then each client randomizes each subgraph using the randomized response with $\varepsilon_l = \varepsilon/m$. As a result, the baseline can provide ε-Edge DDP, i.e.,

$$\begin{aligned} &\frac{Pr[\mathcal{M}_1(G_1) \in S_1, ..., \mathcal{M}_m(G_m) \in S_m]}{Pr[\mathcal{M}_1(G_1') \in S_1, ..., \mathcal{M}_m(G_m') \in S_m]} \\ \leq&(e^{\varepsilon_l})^m = e^{m \cdot \frac{\varepsilon}{m}} = e^{\varepsilon}. \end{aligned}$$

Algorithm 11 shows the details of the baseline approach. It takes as input the subgraph set $G = \{G_1, ..., G_m\}$ and privacy budget ε. Each graph G_i is represented as a $n \times n$ adjacent matrix M_i, where each bit $\in \{0,1\}$. If there is an edge between two users, the bit will be set as 1; otherwise, it will be set as 0.

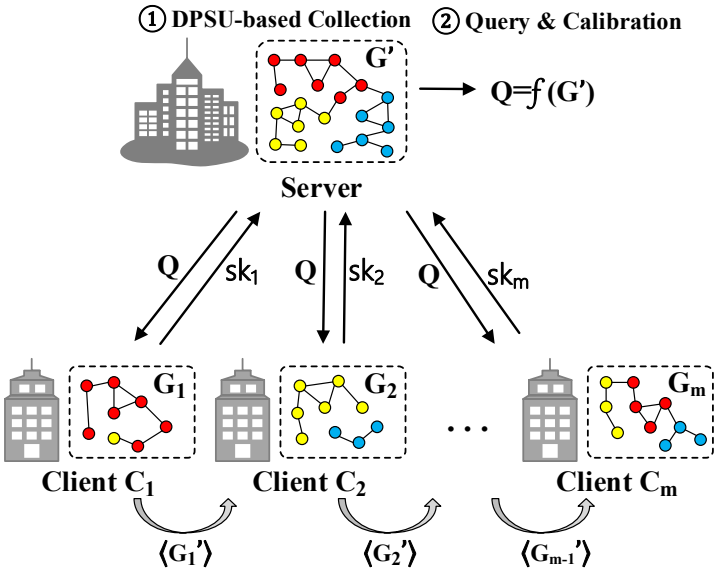


Figure 4.2. Overview of FEAT.

After that, each client flips each bit in the upper triangular part of the matrix with the flipping probability $\frac{1}{1+\varepsilon_l}$, where $\varepsilon_l = \varepsilon/m$. After that, the server collects all noisy edges and aggregates an union as a noisy global graph G'. Finally, the server computes the targeted graph statistics $f(G)'$ based on the noisy global graph G'. This algorithm is denoted as Baseline.

Discussion. To provide a strong privacy guarantee, too small privacy budget is allocated to each client. Although the baseline approach achieves our privacy goal discussed in Section 4.2, much redundant noise is added into results. For example, for k-star counting, Baseline obtains the expected l_2 loss errors of $O(\frac{(mn)^{2k-2}}{\varepsilon^2})$; For triangle counting, it attains the expected l_2 loss errors of $O(\frac{(mn)^2}{\varepsilon^2})$. Our experiments in Section 4.5 further shows that the baseline approach cannot obtain competitive result accuracy under various cases.

4.3.2 Overview of FEAT

To alleviate the limitations of Baseline method, we propose a **fe**derated gr**a**ph analy**t**ic framework, called FEAT, as shown in Figure 4.2. It supports arbitrary downstream common graph statistics while satisfying ε-Edge DDP.

As discussed in the above section, one edge appears in m clients in the extreme case and then the overall privacy budget ε should be allocated to m clients, i.e., $\varepsilon_l = \varepsilon/m$, which leads to much noise. In FEAT, we leverage the private set union

Algorithm 12 Overall Protocol of FEAT.

Input: Subgraph set $G = \{G_1, ..., G_m\}$,
where G_i represented as $n \times n$ adjacent matrix,
Privacy budget ε, Targeted query Q
Output: Query result Q'
Step 1: DPSU-based Graph Collection
1: $G' \leftarrow \mathsf{CollectGraph}(G, \varepsilon)$ ▷ Algorithm 13
Step 2: Graph Query and Calibration
2: $Q' \leftarrow \mathsf{Query}(G', Q)$ ▷ Section 4.3.4
3: **return** Q'

(PSU) technique [95–98] to achieve $\varepsilon_l = \varepsilon$, reducing the noise scale. PSU allows parties to collaboratively compute the union of multiple sets held by different parties without revealing the individual elements of the sets to other parties. It guarantees that an element appears in the final union only once. However, previous PSU protocols cannot be easily applied in our FGA setting. Most of PSU works [95, 96, 98] focus on a two-party setting, which is different from our multi-party scenario. Although the paper [97] proposes a multi-party protocol, it faces the security and efficiency issue. Additionally, this multi-party PSU [97] outputs the true union and is unable to provide the differential privacy guarantee. To this end, we propose a differentially private set union protocol to compute the union of multiple subgraphs while satisfying ε-Edge DDP. The detailed analysis will be presented in Section 4.3.3.

Algorithm 12 presents the overall protocol of FEAT. It involves two kinds of entities: clients and a server. The local clients C_i, $i \in [m]$, owns a subgraph that is represented as a $n \times n$ adjacent matrix (n is the number of users). It takes as input the set of subgraphs $G = \{G_1, ..., G_m\}$, privacy budget ε, and the targeted graph query Q. At the beginning, FEAT recalls a CollectGraph() function to collect a noisy global graph (step ①). Each client perturbs its subgraph with suitable noise and then encrypts the noisy subgraph. Next, clients communicate with each other to compute the union of subgraphs. All clients collaboratively decrypts the union and outputs a noisy global graph (details in Section 4.3.3). Once obtaining the noisy global graph, it executes the targeted graph statistics (step ②). In this paper, we compute the subgraph counts (i.e., k-stars, triangles) to verify the effectiveness of our framework. Considering the estimation bias, the server further calibrates the noisy results to improve the utility (refer to Section

4.3.4 for details).

4.3.3 DPSU-based Graph Collection

We propose a DPSU protocol based on the state-of-the-art PSU method for computing the global graph under DP.

PSU Protocol. Suppose that each client $C_{i\in[1,m]}$ owns a set $X_i \subseteq X = \{x_1, ..., x_n\}$. Our goal is to compute the union $X = \bigcup_{i=1}^{m} X_i$.

The SOTA multi-party PSU method [97] supports computing the union among multiple clients as follows:
(1) *Initialization.* Each client C_i creates a flag vector W_i. If $x_{j\in[1,n]} \in X_i$, $W_{i,j} = 1$; otherwise, $W_{i,j} = 0$.
(2) *Key Generation.* Each client $C_{i\in[1,m]}$ generates a pair of keys $\langle pk_i, sk_i\rangle$, where $pk_i = g^{sk_i}$. All clients jointly generate the public key: $pk = \prod_{i=1}^{m} pk_i = g^{sk_1+...+sk_m}$.
(3) *Encryption.* Each client $C_{i\in[1,m]}$ encrypts W_i with the public key pk: $\mathsf{Enc}(W_i) = (\mathsf{Enc}(W_{i,1}), ..., \mathsf{Enc}(W_{i,n}))$.
(4) *Modification.* Each client $C_{i\in[2,m]}$ modifies the flag vector W_i in sequence as follows: if $x_{j\in[1,n]} \notin X_i$, $\mathsf{Enc}(W_{i,j}) = \mathsf{Enc}(W_{i-1,j})$; otherwise, $\mathsf{Enc}(W_{i,j}) = \mathsf{Enc}(W_{i,j})$.
(5) *Decryption.* All clients jointly decrypt $\mathsf{Enc}(W_m)$ with secret keys $\{sk_1, ..., sk_m\}$: $W_m \leftarrow \mathsf{Dec}[\mathsf{Enc}(W_m)]$. For $j \in [1, n]$, if $W_{m,j} = 1$, $X \leftarrow X \cup \{x_j\}$.

However, the above protocol may face the security issue. It is implemented based on finite field cryptography (FFC) [102] and in fact the prime number p with 512 bits is not secure according to Table 2 in [103]. If this multi-party PSU is implemented in a secure way, the prime number p should be set as 3072 bits. As a result, it is inefficient for the large-scale graph analytics. To improve the tradeoff between security and efficiency, we implement the PSU protocol based on the Elliptic Curve Cryptography (ECC) [104], which is more efficient than finite field cryptography (FFC). In particular, there are three cryptographic libraries for implementing ECC, namely, Libsodium, OpenSSL, and MCL. As shown in Table 4.2, Libsodium is more suitable for processing large-scale graph data, which motivates us to implement PSU protocol based on Libsodium library.

DPSU Protocol. We then propose a differentially private set union (DPSU) protocol for aggregating a global graph. One challenge is that the output of the

Table 4.2. Performance of ECC with Three Libraries.

ECC \ n	10	10^2	10^3	10^3	10^4	10^5
Libsodium	0.0056	0.059	0.558	5.26	53.14	528.10
OpenSSL	0.0098	0.060	0.549	5.49	53.76	539.52
MCL	0.0541	0.504	4.983	49.84	498.84	4986.43

n: data size

PSU protocol does not satisfy DP. The individual privacy could be disclosed via the union of subgraphs. Although some previous works [105–108] discuss how to calculate the private set union while satisfying DP, their system models differ from ours and can not be used directly. One naive solution is that each client C_i perturbs the flag vector W_i using the randomized response (RR) mechanism [109] before encrypting. Specifically, each client C_i flips each bit in her flag vector W_i with probability $p = \frac{1}{1+e^{\varepsilon}}$ in the step *Initialization* of PSU protocol, where ε is the privacy budget. Subsequently, the differentially private set union can be computed according to step (2)$\sim$(5) in the above PSU protocol.

Although this natural solution can provide the DP guarantee, it leads to much bias in a noisy graph. In fact, most of real-world graphs tend to be sparse, which means that there are much more 0s than 1s in an adjacent matrix. After the randomly flipping bits, however, the number of 1s is much larger than that of 0s, making the noisy global graph denser than the original one. Additionally, the step *Modification* further amplifies the denser problem. Even if '0' bit is flipped into '1' bit in one of m subgraphs, the according bit in final union will become '1'. In particular, assume that the number of '1' bits in a true global graph is t and the number of '0' bits is $(n^2 - t)$. After flipping, the number of '1' bits in a noisy global graph becomes $m[(1-p)t + p(n^2 - t)]$. Take the Facebook graph as an example, which has 4,039 nodes (i.e., n) and 88,234 edges (i.e., t). Even with a fairly small privacy budget $\varepsilon = 0.1$ and the number of clients $m = 5$, the expected number of '1' bits will become 3.8×10^7, increasing at least 439 times than before.

To address the above issue, we propose a novel differentially private set union protocol. In particular, '0' bits are perturbed only by the first client and '1' bits are randomized by all clients. The server can then obtain the whole noisy matrix by computing the union of them. Theoretically, our method can reduce the denser problem by a factor of m. As a result, the utility loss can be alleviated by at least

a factor of $O(m^2)$. For instance, for k-star counting, FEAT achieves the expected l_2 loss errors of $O(\frac{n^{2k-2}}{\varepsilon^2})$; For triangle counting, it attains the expected l_2 loss errors of $O(\frac{n^2}{\varepsilon^2})$. To further suppress the bias from the randomized response, we calibrate the noisy results during the graph query processing. The details will be discussed in next Section 4.3.4.

Algorithm 13 describes the details of our DPSU-based edge collection method. It takes as input the subgraph set G and the privacy budget ε. Each client C_i initializes an edge domain $\mathbb{E}$ according to the node information V. In particular, each client C_i first constructs a $n \times n$ adjacent matrix M_i, where $n = |V|$, and then transforms the upper triangular part of M_i into a vector $\mathbb{E}$ with the size $N = |\mathbb{E}| = \frac{n(n-1)}{2}$. For example, $\mathbb{E}_1 = e_1 = \langle v_1, v_2 \rangle$, $\mathbb{E}_2 = e_2 = \langle v_1, v_3 \rangle$. The server initializes an empty set E for collecting the edge information. Each client C_i generates a pair of keys $\langle ps_i, sk_i \rangle$, and all clients jointly generate the public key pk. Client C_i first initializes a flag vector Y according to the principle in line 5. If $\mathbb{E}_j$ exists in E_1, the according bit will be set as 1; otherwise, it will be set as 0. After, client C_1 perturbs each bit of Y with the flipping probability $\frac{1}{1+e^{\varepsilon}}$ using the randomized response, and encrypts the noisy Y' with the public key pk. Then, client C_1 sends $\mathsf{Enc}(Y')$ to the client C_2. For each client C_i from C_2 to C_m, C_i updates $\mathsf{Enc}(Y')$ according to the principle in lines 10 to 15. If $\mathbb{E}_j$ is in E_i, client C_i will flip '1' with RR and then replace $\mathsf{Enc}(Y'_j)$ with $\mathsf{Enc}(\mathsf{RR}(1))$. Once the client C_m completes updating $\mathsf{Enc}(Y')$ and sending it to the server, all clients jointly decrypt $\mathsf{Enc}(Y')$ and send decrypted Y' to the server. Finally, the server generates and releases E according to Y'. We call this algorithm by CollectGraph.

Example 3.1. Table 4.3 shows how Algorithm 13 works for computing the union of local edge sets while satisfying the DP. Specially, clients C_1, C_2, C_3 owns private edge sets $E_1 = \{e_1, e_2\}, E_2 = \{e_2, e_3\}, E_3 = \{e_5\}$, and node set is $V = \{v_1, v_2, v_3, v_4\}$. Thus, $E_1, E_2, E_3 \subseteq \mathbb{E} = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, where $e_1 = \langle v_1, v_2 \rangle$, $e_2 = \langle v_1, v_3 \rangle$, $e_3 = \langle v_1, v_4 \rangle$, $e_4 = \langle v_2, v_3 \rangle$, $e_5 = \langle v_2, v_4 \rangle$, $e_6 = \langle v_3, v_4 \rangle$. The goal is to compute $E = E_1 \cup E_2 \cup E_3$. Firstly, client C_1 constructs a flag vector Y according to line 5 in Algorithm 13. Then, C_1 perturbs each bit of Y and encrypts it to obtain Y'. Secondly, clients C_2 and C_3 update Y based on the principle of lines 10 to 15 in Algorithm 13. Thirdly, all clients jointly decrypt $\mathsf{Enc}(Y')$ and send Y' to the server. Finally, the server can obtain the edge union $E = \{e_1, e_2, e_4, e_5\}$ according to Y'.

Table 4.3. An Example of DPSU Protocol.

Y	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6
C_1	$[\![(1)]\!]$	$[\![(1)]\!]$	$[\![(0)]\!]$	$[\![(0)]\!]$	$[\![(0)]\!]$	$[\![(0)]\!]$
C_2	$[\![(1)]\!]$	$[\![(1)]\!]$	$[\![(1)]\!]$	$[\![(0)]\!]$	$[\![(0)]\!]$	$[\![(0)]\!]$
C_3	$[\![(1)]\!]$	$[\![(1)]\!]$	$[\![(1)]\!]$	$[\![(0)]\!]$	$[\![(1)]\!]$	$[\![(0)]\!]$
Y'	1	1	0	1	1	0
E	e_1	e_2	-	e_4	e_5	-

$[\![x]\!]$: $\mathsf{Enc}(x)$ (x): $\mathsf{RR}(x)$.
In this example, Y_3' and Y_4' are flipped by RR.

Theorem 11 *Algorithm 13 satisfies ε-Edge DDP.*

Proof of Theorem 11. Let $G = (V, E)$ and $G' = (V, E')$ be two neighboring graphs. Let G_i and G_i' $(i \in [1, m])$ be the neighboring graphs of client C_i with respect to G and G', respectively. Let $\{\mathcal{M}_i, i \in [1, m]\}$ be a set of randomized mechanisms with any subsets of possible outputs $S_i \subseteq range(\mathcal{M}_i), i \in [1, m]$. Given the privacy budget ε, we can easily obtain $Pr[\mathcal{M}_i(G_i) \in S_i] \leq e^{\varepsilon} Pr[\mathcal{M}_i(G_i') \in S_i]$. Due to using the threshold ElGamal encryption, any change of adding or deleting an edge from m clients is only collected by the server once. Thus, we have

$$\begin{aligned}
&\frac{Pr[\mathcal{M}_1(G_1) \in S_1, ..., \mathcal{M}_m(G_m) \in S_m]}{Pr[\mathcal{M}_1(G_1') \in S_1, ..., \mathcal{M}_m(G_m') \in S_m]} \\
=&\frac{Pr[\mathcal{M}_1(G_1) \in S_1]...Pr[\mathcal{M}_m(G_m) \in S_m]}{Pr[\mathcal{M}_1(G_1') \in S_1]...Pr[\mathcal{M}_m(G_m') \in S_m]} \\
=&e^{\varepsilon}
\end{aligned}$$

Therefore, Algorithm 13 satisfies ε-Edge DDP. Furthermore, by the immunity to post-processing [54], Algorithm 12 provides ε-Edge DDP guarantee.

4.3.4 Graph Query Processing

In this section, we execute two common subgraph counting queries, i.e., k-star counting [5, 7, 110] and triangle counting [2, 5, 10], in order to explain how to execute the Query function of Algorithm 12 and how to calibrate the noisy results.

A k-star refers to a subgraph consisting of a central node connecting to k other nodes. To count the number of k-stars in a given graph, we iterate through each

vertex in the graph and compute $\binom{d'_i}{k}$, where d'_i is the noisy degree of node v_i. The k-star counts S of the whole graph is equal to the summation of each node's k-stars, i.e., $S = \sum_{i=1}^{n} \binom{d'_i}{k}$. However, a direct computation of node degrees from G' can lead to significant bias induced by the randomized response in Algorithm 13. To mitigate the bias, we leverage the post-processing property of DP [54], yielding an unbiased estimation $\widetilde{d}_i$ of d_i as Proposition 1. Hence, we can obtain an unbiased estimation of k-stars, i.e., $S = \sum_{i=1}^{n} \binom{\widetilde{d}_i}{k}$.

Proposition 1 *Let G' be a noisy global graph. Let d'_i be the node degree of v_i in G' and $\widetilde{d}_i$ be an unbiased estimate of d'_i. Let n be the number of nodes in G'. Let $p = \frac{1}{1+e^{\varepsilon}}$ be the flipping probability, where ε is the privacy budget in* FEAT. *We have*

$$\widetilde{d}_i = \frac{1}{1-2p}(d'_i - np). \tag{4.1}$$

Proof of Proposition 3. The mapping relationship between d'_i and $\widetilde{d}_i$ can be represented as:

$$d'_i = \widetilde{d}_i(1-p) + (n - \widetilde{d}_i)p. \tag{4.2}$$

Then we can prove Proposition 3.

A triangle in a graph refers to a subgraph consisting of three nodes connected by three edges, forming a closed loop. To count the number of triangles in a graph, one common approach is to iterate each subgraph with three nodes and check if it is a loop. However, simply counting the triangles in a noisy graph G' introduces a significant bias. We continue to calibrate the biased triangle counts through the post-processing. Building upon the insights of [5], we categorize triplets in G_i into four types based on the number of edges they involve: t_0, t_1, t_2, and t_3, where t_j ($j \in 0, 1, 2, 3$) represents the count of triplets in G_i involving j edges (referred to as *j-edge*; 3-edges are identical to triangles). Thus, we can compute the unbiased triangle counts $\widetilde{T}$ according to Proposition 2.

Proposition 2 *Let G' be a noisy global graph. Let t_0, t_1, t_2, t_3 be the number of 0-edges, 1-edges, 2-edges, and triangles in G', respectively. Let ε be the privacy budget used in* FEAT. *We have*

$$\widetilde{T} = \frac{1}{(e^{\varepsilon}-1)^3}(-t_0 + t_1 e^{\varepsilon} - t_2 e^{2\varepsilon} + t_3 e^{3\varepsilon}). \tag{4.3}$$

Proof of Proposition 2. Let u_0, u_1, u_2, u_3 be the number of 0-edges, 1-edges, 2-edges, and triangles in G, respectively, when we do not flip 1/0 using the randomized response. Let $x = e^{\varepsilon}$. Then we have:

$$(t_0, t_1, t_2, t_3) = (u_0, u_1, u_2, u_3)\boldsymbol{A}, \tag{4.4}$$

$$\boldsymbol{A} = \frac{1}{(x+1)^2} \begin{bmatrix} x^3 & 3x^2 & 3x & 1 \\ x^2 & x^3+2x & 2x^2+1 & x \\ x & 2x^2+1 & x^3+2x & x^2 \\ 1 & 3x & 3x^2 & x^3 \end{bmatrix}.$$

$\boldsymbol{A}$ is a transition matrix from a type of subgraph (0-edge, 1-edge, 2-edge, and triangle) in an original graph to a type of subgraph in a noisy graph. Let $(\widetilde{u}_0, \widetilde{u}_1, \widetilde{u}_2, \widetilde{u}_3)$ be an unbiased estimate of (u_0, u_1, u_2, u_3). Then we obtain:

$$(\widetilde{u}_0, \widetilde{u}_1, \widetilde{u}_2, \widetilde{u}_3) = (t_0, t_1, t_2, t_3)\boldsymbol{A}^{-1} \tag{4.5}$$

Let $\boldsymbol{A}^{-1}_{i,j}$ be the (i, j)-th element of $\boldsymbol{A}^{-1}$. Then we have:

$$\boldsymbol{A}^{-1}_{1,1} = \frac{x^3}{(x-1)^3}, \boldsymbol{A}^{-1}_{2,1} = -\frac{x^2}{(x-1)^3}, \tag{4.6}$$

$$\boldsymbol{A}^{-1}_{3,1} = \frac{x}{(x-1)^3}, \boldsymbol{A}^{-1}_{4,1} = -\frac{1}{(x-1)^3}. \tag{4.7}$$

By combining above equations, Proposition 2 is proved.

4.4 An Improved Framework: FEAT+

Although FEAT reduces the problems from the limited view and overlaps, there is still large space for improving the overall utility. FEAT calculates common graph statistics based on a noisy global graph collected from local clients and without considering the true local subgraph information. The graph statistics such as k-stars and triangles can be calculated more accurately via an additional round of interaction between the server and clients. After the server publishes G', each client C_i can concatenate her local subgraph G_i with G', which removes the limited view issues. For example, in Figure 4.3, if the targeted query is to count triangles, client C_1 will return $Q_1 = 4$ instead of $Q_1 = 1$ since she can see

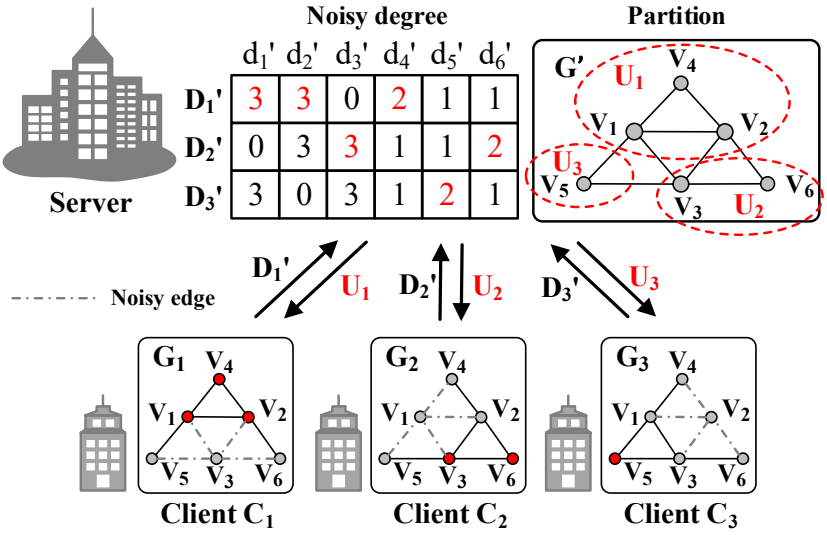


Figure 4.3. Motivation of FEAT+.

the edges $\langle v_1, v_3\rangle, \langle v_2, v_3\rangle, \langle v_3, v_5\rangle$ and $\langle v_3, v_6\rangle$ via G'. Thus, the final answer will be $(4+4+4)/3 = 4$ (redundant counts will be removed).

Nevertheless, one edge may be reported by different clients multiple times, which leaks individual privacy (as discussed in Section 4.1). For example, in Figure 4.3, the triangle $\langle v_1, v_2, v_3\rangle$ may be collected by the server three times during counting triangles. In intuition, if one edge is reported by one of m clients only once, the probability of distinguishing such an edge will be reduced. Hence, the problem is reduced to determining how to assign the same information to one of the clients for reporting. To do this effectively, we propose a degree-based node partition method that splits nodes into m disjoint node sets so that each node is assigned to a client with the highest node degree. In this context, the degree information plays a significant role: the higher a node degree, the more edges it possesses, and the more likely it is to be involved in k-stars or triangles. For instance, in Figure 4.3, the degree of node v_4 in client C_1 ($= 2$) is larger than the degrees of clients C_2 and C_3 ($= 1$). Thus, client C_1 can count 2-stars or triangles of v_4 more accurately than clients C_2 and C_3. This is an intuition behind our FEAT+.

4.4.1 Overview of FEAT+

Algorithm 14 presents the overall protocol of FEAT+. It mainly consists of two phases: global graph collection and local query collection. The global graph collection is used to publish a noisy global graph G', which is finished by FEAT (Algorithm 12). The local query collection consists of three main steps: degree-based partition (Algortihm 15), graph query processing (Section 4.4.3), and perturbation. To be specific, it takes as input local subgraph set $G = \{G_1, ..., G_m\}$, privacy budget $\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_3$, and query sensitivity $\triangle$. FEAT+ first collects a noisy global graph G' via FEAT using ε_1. Then, this global graph is published to each client C_i for local query collection. Next, FEAT+ recalls a PartitionNode function with privacy budget ε_2, to split the node set V into m disjoint node sets U (Step 1). These node sets are distributed to the respective clients. After that, each client C_i computes the targeted query. In our paper, we take k-star and triangle counting as examples to explain the rationale behind Query (Step 2). Since the noisy global graph G' is dense, we carefully design calibration techniques to obtain unbiased estimates of k-stars or triangles, as presented in Section 4.4.3. Upon obtaining the query results, each client C_i perturbs the local answers using a suitable noise (Step 3). Specifically, each client C_i calculates the noisy query $Q'_i = Q_i + \mathsf{Lap}(\frac{\triangle}{\varepsilon_3})$ by adding the Laplacian noise to Q_i, where $\triangle$ is the global sensitivity. Finally, the server computes $Q' \leftarrow \sum_{i=1}^{m} Q'_i$, which is an unbiased estimate of Q and satisfies DP.

Theorem 12 *Algorithm 14 satisfies $(\varepsilon_1 + \varepsilon_2 + \varepsilon_3)$-Edge DDP.*

Proof of Theorem 12. Algorithm 14 uses three kinds of privacy budgets: ε_1 in FEAT, ε_2 in PartitionNode, and ε_3 in Perturbation. By Theorem 11, FEAT satisfies ε_1-Edge DDP. In PartitionNode, each client $C_{i\in[m]}$ adds the Laplacian noise $Lap(\frac{m}{\varepsilon_2})$ to the node degree, which satisfies ε_2-Edge DDP. In Perturbation, each client $C_{i\in[m]}$ adds the Laplacian noise $Lap(\frac{\triangle}{\varepsilon_3})$ to the query result Q_i, which satisfies ε_3-Edge DDP. Following the post-processing property of DP, the aggregated result Q' satisfies ε_3-Edge DDP. Thus, the entire process of Algorithm 14 provides $(\varepsilon_1 + \varepsilon_2 + \varepsilon_3)$-Edge DDP.

4.4.2 Degree-based Node Partition

We propose a degree-based user partition technique to split the node set V into multiple disjoint user sets $\{\widetilde{V}_1, ..., \widetilde{V}_m\}$. For each node, the server collects m node degrees $\{d'_1, ..., d'_m\}$ from m clients privately. The node (i.e., user) will be distributed to the client that sends the maximum degree to the server. Algorithm 15 presents how to partition users into multiple disjoint sets. It takes as input the true subgraph sets $G = \{V, E\} = \{G_1, ..., G_m\}$ and privacy budget ε. At the outset, the server initializes the empty set $U = \{U_1, ..., U_m\}$ for each client, in order to record the partition information. For each node v_i in the node set V, each client C_j computes the node degree $d_{i,j}$ based on its true subgraph G_j. Then, client C_j perturbs the degree with the Laplace mechanism and sends the noisy degree $d'_{i,j}$ to the server. After that, the server computes the max noisy degree $d'_{i,k} = \mathsf{max}\{d'_{i,1}, ..., d'_{i,m}\}$ for each node v_i and adds the user index i to the corresponding user partition set U_k. The final output is the user partition sets $U = \{U_1, ..., U_m\}$, where $U_i \cap U_j = \varnothing$ and $\bigcup_{i=1}^{m} U_i = V$. This algorithm is denoted by PartitionNode.

4.4.3 Graph Query Processing

In this section, we execute two basic subgraph counting queries, i.e., k-star counting [5, 7, 110] and triangle counting [2, 5, 10], in order to explain how to execute the Query function in line 4 of Algorithm 14.

k-Star Counting

A k-star refers to a subgraph consisting of a central node connecting to k other nodes. The key is to compute the total number of edges in a graph, i.e., the summation of node degrees. From each client C_i's view, each node degree is contributed from a local subgraph G_i and a noisy global graph G'. Thus, the problem is reduced to compute the true node degree d_1 and noisy node degree d_2.

Algorithm 16 shows an instantiation of the Query function in k-star counting. It takes as input a noisy global graph G', i-th subgraph G_i, and i-th user partition U_i. Client C_i first initializes $d_1 = d_2 = 0$ to record two kinds of degrees. Then, it obtains a new global graph $\overline{G}$ by computing the union between G' and its true subgraph G_i (line 2). After that, it traverses each node in the user partition set

U_i and calculates d_1 and d_2. If the edge $\langle v_j, v_k \rangle$ exists in subgraph G_i, the true degree will be updated as $d_1 \leftarrow d_1 + 1$; otherwise, the noisy degree will be updated as $d_2 \leftarrow d_2 + 1$ (lines 5-6). However, simply computing the degrees from G' can introduce a significant bias, as the randomized response in Algorithm 13 makes a graph dense [5,6,32]. By the post-processing property of DP [54], we obtain an unbiased estimate $\widetilde{d}_2$ of d_2 according to Proposition 3 (line 7). Upon obtaining the node degree $d = d_1 + \widetilde{d}_2$, client C_i can calculate the k-stars by $\binom{d}{k}$. The final output is the number of k-stars.

Proposition 3 *Let $\overline{G}$ be the union of a noisy global graph G' and local subgraph G_i. Let G^C be the absolute complement of G_i in $\overline{G}$. Let d'_i be the node degree of v_i in G^C and $\widetilde{d}_i$ be an unbiased estimate of d'_i. Let n be the number of nodes in G^C. Let $p = \frac{1}{1+e^{\varepsilon_1}}$ be the flipping probability, where ε_1 is the privacy budget in* FEAT *(i.e., line 1 of Algorithm 14). We have*

$$\widetilde{d}_i = \frac{1}{1-2p}(d'_i - np). \tag{4.8}$$

Proof of Proposition 3. The mapping relationship between d'_i and $\widetilde{d}_i$ can be represented as:

$$d'_i = \widetilde{d}_i(1-p) + (n - \widetilde{d}_i)p. \tag{4.9}$$

Then we can prove Proposition 3.

Triangle Counting

Next, we focus on triangle counting. This is more challenging because three edges of a triangle can be from the local subgraph G_i or a noisy global graph G'. There are four kinds of triangles according to the number of edges from G_i: T_0, T_1, T_2, and T_3, where T_j ($j \in \{0,1,2,3\}$) is the number of triangles involving j edges from G_i (referred to as *j-edge* in G_i). Since T_0, T_1, and T_2 involves some noisy edges from G', simply counting the noisy triangles can introduce a bias. We propose different empirical estimation methods to obtain unbiased counts, as presented in Propositions 4, 5, and 6 at the end of this subsection.

Algorithm 17 presents an instantiation of the Query function in triangle counting. It takes as input a noisy global graph G', the i-th subgraph G_i, the i-th user partition U_i, and the privacy budget ε_1. Client C_i first obtains a graph $\widetilde{G}$ by merging G' and G_i. Then it calculates four kinds of triangles (T_0, T_1, T_2, T_3),

i.e., 0-edge, 1-edge, 2-edge, and 3-edge in G_i. After that, the unbiased estimates $(\widetilde{T}_0, \widetilde{T}_1, \widetilde{T}_2)$ of (T_0, T_1, T_2) are computed based on Proposition 4, 5, and 6. The final triangle count T can be obtained by aggregating $\widetilde{T}_0, \widetilde{T}_1, \widetilde{T}_2$, and T_3.

Proposition 4 *Let $\overline{G}$ be the union of a noisy global graph G' and local subgraph G_i. Let G^C be the absolute complement of G_i in $\overline{G}$. Let t_0, t_1, t_2, t_3 be the number of 0-edges, 1-edges, 2-edges, and triangles in G^C, respectively. Let T_0 be the number of 0-edges in G_i, i.e., $T_0 = t_3$. Let $\widetilde{T}_0$ be an unbiased estimate of T_0. Let ε_1 be the privacy budget used in* FEAT *(i.e., line 1 of Algorithm 14). We have*

$$\widetilde{T}_0 = \frac{1}{(e^{\varepsilon_1}-1)^3}(-t_0 + t_1 e^{\varepsilon_1} - t_2 e^{2\varepsilon_1} + T_0 e^{3\varepsilon_1}). \tag{4.10}$$

The proof details of Proposition 4 can refer to Proposition 2.

Proposition 5 *Let $\overline{G}$ be the union of a noisy global graph G' and local subgraph G_i. Let G^C be the absolute complement of G_i in $\overline{G}$. Let t_0, t_1, t_2 be the number of 0-edges, 1-edges, and 2-edges in G^C, respectively. Let T_1 be the number of 1-edges in G_i, i.e., $T_1 = t_2$. Let $\widetilde{T}_1$ be an unbiased estimate of T_1. Let ε_1 be the privacy budget used in* FEAT. *We have*

$$\widetilde{T}_1 = \frac{(e^{\varepsilon_1}+1)[e^{2\varepsilon_1}t_0 - e^{\varepsilon_1}(e^{\varepsilon_1}+1)t_1 + (2e^{\varepsilon_1}+1)T_1]}{(e^{\varepsilon_1}-1)(e^{2\varepsilon_1}-2e^{\varepsilon}-1)}. \tag{4.11}$$

Proof of Proposition 5. Consider that one edge of a triangle is from G_i and another two edges are from G^C. Let u_0, u_1, u_2 be the number of 0-edges, 1-edges, and 2-edges in G^C, respectively, when we do not flip 1/0 using the randomized response. Let $x = e^{\varepsilon_1}$. Then we have:

$$(t_0, t_1, t_2) = (u_0, u_1, u_2)\boldsymbol{A} \tag{4.12}$$

$$\boldsymbol{A} = \tfrac{1}{(x+1)^2}\begin{bmatrix} x^2 & 2x & 1 \\ x & 1+x & x \\ 1 & 2x & x^2 \end{bmatrix}$$

$\boldsymbol{A}$ is a transition matrix from a type of subgraph (0-edge, 1-edge, 2-edge) in an original graph to a type of subgraph in a noisy graph. Let $\widetilde{u}_0, \widetilde{u}_1, \widetilde{u}_2$ be the unbiased estimation of (u_0, u_1, u_2). Then we obtain:

$$(\widetilde{u}_0, \widetilde{u}_1, \widetilde{u}_2) = (t_0, t_1, t_2)\boldsymbol{A}^{-1} \tag{4.13}$$

Let $\boldsymbol{A}_{i,j}^{-1}$ be the (i,j)-th element of $\boldsymbol{A}^{-1}$. Then we have:

$$\boldsymbol{A}_{1,1}^{-1} = \frac{(x+1)x^2}{(x-1)(x^2-2x-1)}, \tag{4.14}$$

$$\boldsymbol{A}_{2,1}^{-1} = \frac{(x+1)(-x^2-x)}{(x-1)(x^2-2x-1)}, \tag{4.15}$$

$$\boldsymbol{A}_{3,1}^{-1} = \frac{(x+1)(2x+1)}{(x-1)(x^2-2x-1)}, \tag{4.16}$$

By combining above equations, Proposition 5 is proved.

Proposition 6 *Let G' be a noisy global graph and G_i be the local subgraph. Let T_2 be the number of 2-edges in G_i. Let S be the number of 2-stars in G_i. Let $\widetilde{T}_2$ be an unbiased estimate of T_2. Let $p = \frac{1}{1+e^{\varepsilon_1}}$ be the flipping probability. We have*

$$\widetilde{T}_2 = \frac{1}{1-2p}(T_2 - Sp). \tag{4.17}$$

Proof of Proposition 6. The mapping relationship between d'_i and $\widetilde{d}_i$ can be represented as:

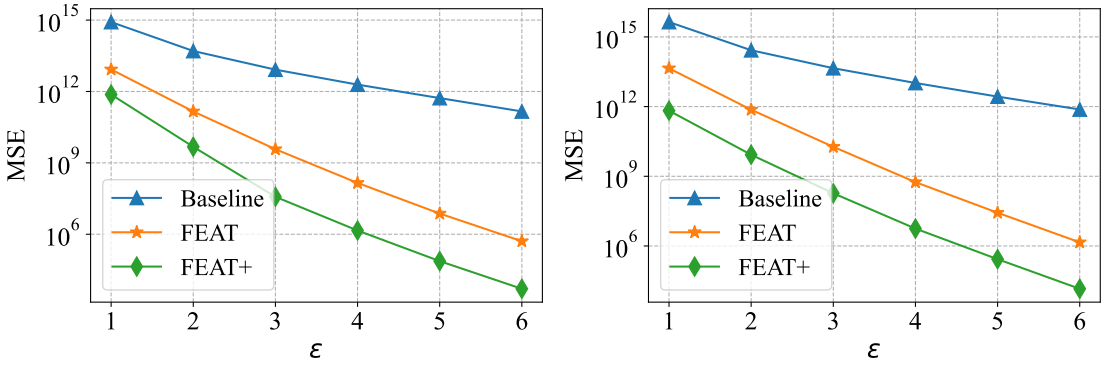
$$T_2 = \widetilde{T}_2(1-p) + (S - \widetilde{T}_2)p. \tag{4.18}$$

Then we can prove Proposition 6.

4.5 Experimental Evaluation

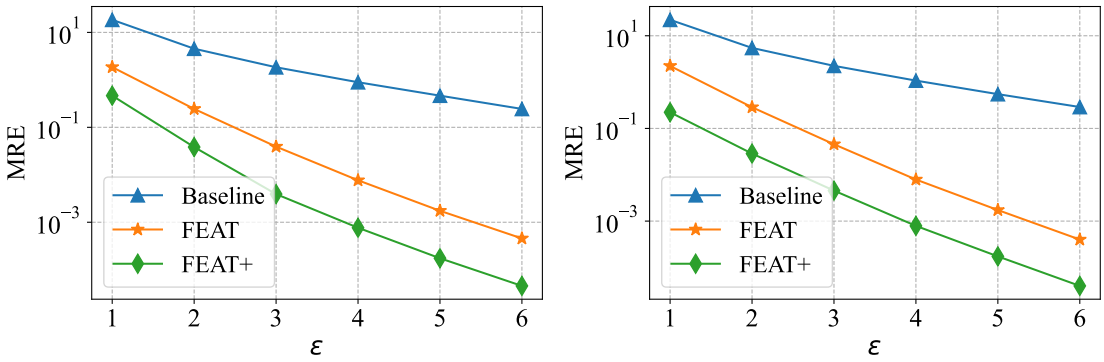
In this section, we evaluate our FEAT and FEAT+ along two dimensions: utility and running time. To simulate the federated scenario, we split a graph randomly into multiple local subgraphs by controlling two key parameters: *sampling rate* ρ and *overlapping rate* σ. Then, we conducted experiments to answer the following questions:

- **Q1**: How do our general FEAT and improved FEAT+ compare with baseline approaches (denoted by Baseline) in terms of the utility-privacy trade-off?
- **Q2**: How do the sampling rate ρ and overlapping rate σ affect accuracy?
- **Q3**: How much do our FEAT and FEAT+ compare with Baseline in terms of the running time?



(a) Facebook (b) Wiki

Figure 4.4. The MSE in 2-star counting.

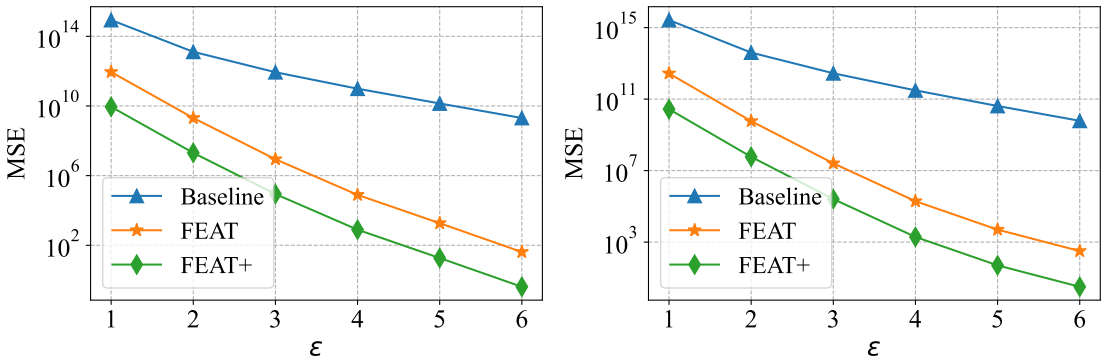


(a) Facebook (b) Wiki

Figure 4.5. The MRE in 2-star counting.

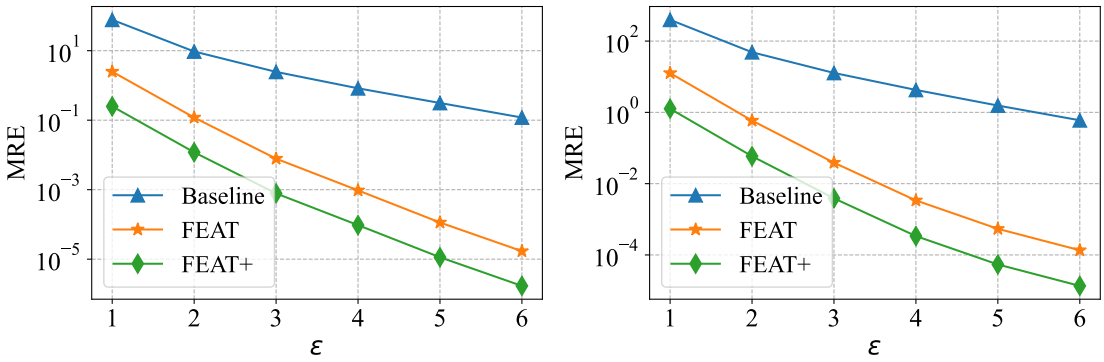
Evaluation Highlights:

- FEAT reduces the error of Baseline by up to an order of 4. FEAT+ outperforms FEAT by at least an order of 1 (Figures 4.4 to 4.7).
- FEAT and FEAT+ significantly outperform Baseline and FEAT, respectively, under various values of ρ and σ (Figures 4.8 and 4.11).
- FEAT and FEAT+ takes more time than Baseline by at least an order of 1 (Figure 4.12).



(a) Facebook (b) Wiki

Figure 4.6. The MSE in triangle counting.



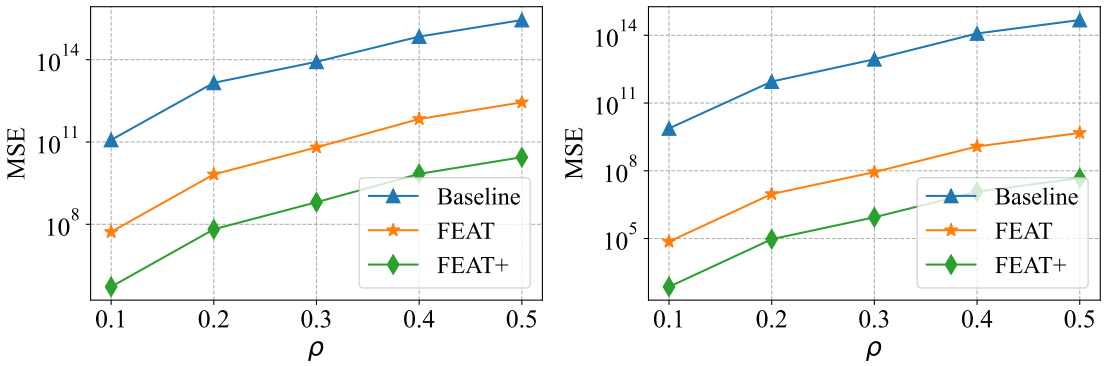
(a) Facebook (b) Wiki

Figure 4.7. The MRE in triangle counting.

4.5.1 Experimental Setup

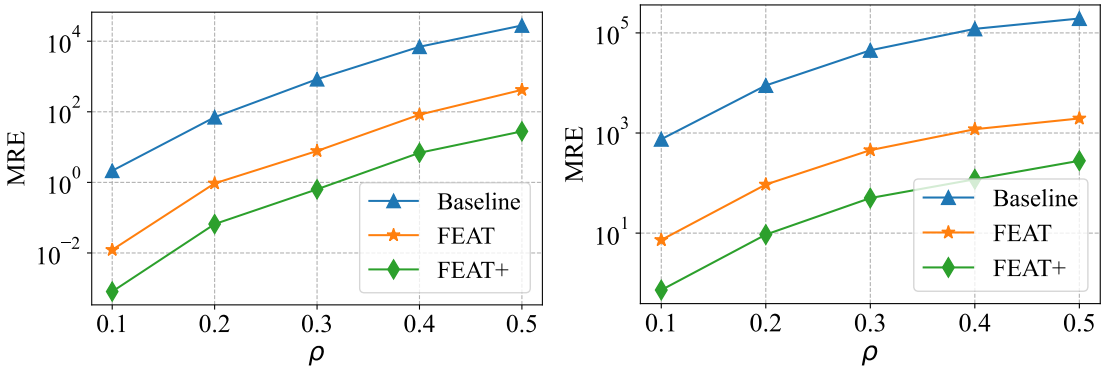
Datasets. We use two real-world graph datasets from SNAP [41] as follows: (1) *Facebook*. The Facebook graph is collected from survey participants using the Facebook app, which includes 4039 nodes and 88234 edges. The average degree of the Facebook graph is 21.85 ($=\frac{88234}{4039}$). (2) *Wiki-Vote*. The Wiki-Vote graph contains all the Wikipedia voting data from the inception of Wikipedia till 2008, which includes 7115 nodes and 103689 edges. The average degree of Wiki-Vote graph is 14.57 ($=\frac{103689}{7115}$). Thus, Wiki-Vote graph is more sparse than the Facebook graph. As explained above, we split a graph randomly into 4 local subgraphs by controlling the sampling rate ρ and overlapping rate σ.

Parameters. There are some key parameters that influence the overall accuracy of the FEAT system. (1) *Privacy Budget* ε. The privacy budget varies from 1 to



(a) 2-Star counting (b) Triangle counting

Figure 4.8. The MSE with various ρ.

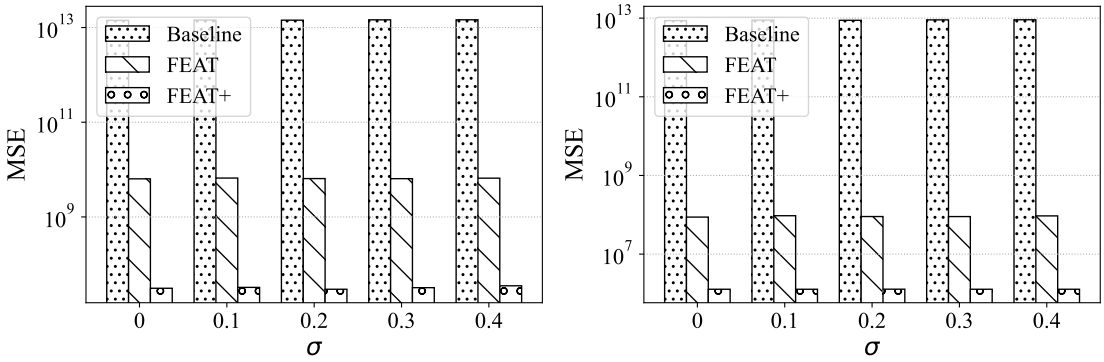


(a) 2-Star counting (b) Triangle counting

Figure 4.9. The MRE with various ρ.

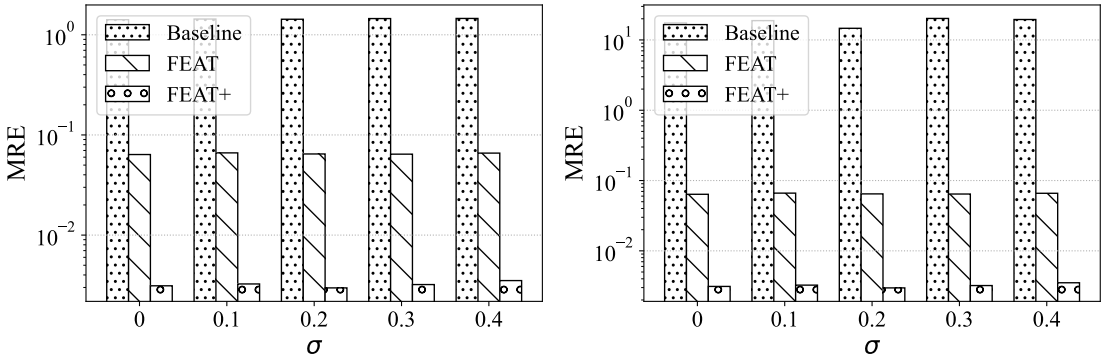
6, and the default is 3. Note that FEAT+ involves three kind of privacy budgets, namely, $\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_3$. We set $\varepsilon_1 = \varepsilon_3 = 0.45\varepsilon$ and $\varepsilon_2 = 0.1\varepsilon$. (2) *Sampling rate* ρ. The sampling rate is the ratio of each local subgraph to the global graph. It is set from 0.1 to 0.5, and the default is 0.3. (3) *Overlapping Rate* σ. The overlapping rate is the ratio of edges shared among multiple local subgraphs to the total number of edges. It varies from 0 to 0.4, and the default is 0.2.

Graph Statistics and Metrics. For graph analytic tasks, we evaluate two common graph statistics: 2-star counts and triangle counts, as in [2,5,7,8,10,110]. For each query, we compare the results Q and Q' from the true graph and noisy graph respectively. We use two common measures to assess the accuracy of our algorithms: mean squared error (MSE) [111] and mean relative error (MRE) [112]. We evaluate the average results over 10 repeated runs.



(a) 2-Star counting (b) Triangle counting

Figure 4.10. The MSE with various σ.



(a) 2-Star counting (b) Triangle counting

Figure 4.11. The MRE with various σ.

4.5.2 Experimental Results

Utility-Privacy Trade-off (Q1). We first answer Q1 by comparing the utility of our FEAT and FEAT+ with that of Baseline when the privacy budget varies from 1 to 6. Figures 4.4 to 4.7 show that the utility loss of all methods decreases with an increase in ε. Our general framework FEAT significantly outperforms Baseline in all cases, and our improved framework FEAT+ can further improve the accuracy of FEAT.

In particular, FEAT outperforms Baseline by at least an order of 1. For instance, for 2-star counting, Figure 4.4(b) shows that when $\varepsilon = 6$, FEAT owns an MSE of 5.09×10^5 while Baseline gives an MSE of 1.45×10^{11}. Similarly, for triangle counting, Figure 4.7(a) shows that FEAT gives an MRE of only $9.53{\times}10^{-4}$ when $\varepsilon = 4$. In contrast, Baseline has an MRE of $8.27{\times}10^{-1}$. This is because Baseline

uses small privacy budgets (i.e., ε/m) to perturb sensitive data, which leads to much noise. Instead, FEAT collects noisy graphs using ε by combining PSU and DP. The same information is collected only once, and therefore, individual privacy is not leaked.

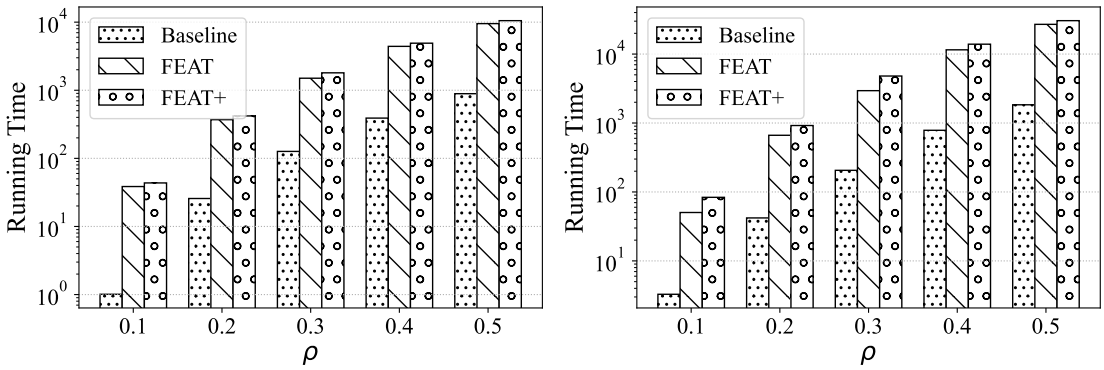
Another observation is that the error of FEAT+ is smaller than that of FEAT in all cases. For example, Figure 4.5(b) shows that when $\varepsilon = 6$, FEAT+ only owns an MRE of 3.86×10^{-5}, whereas FEAT has an MRE of 2.98×10^{-4}. Similarly, Figure 4.6(a) shows that when $\varepsilon = 3$, the MSE of FEAT+ is 4.68×10^{2} when FEAT owns a MSE of 6.25×10^{4}. This is mainly because FEAT+ calculates graph statistics by utilizing local true subgraphs. In contrast, the results of FEAT are computed based on noisy graphs, which results in much utility loss. Thus, FEAT outperforms Baseline significantly, and FEAT+ improves the utility of FEAT.

Parameter Effects (Q2). Next, we evaluate the key parameters that may influence the overall utility of FEAT, i.e., sampling rate ρ and overlapping rate σ. ρ determines the size of local subgraphs $G_i, i \in [m]$, and the size of G_i becomes larger as ρ increases. σ determines the number of the same edges that exist in multiple local subgraphs.

Figures 4.8 and 4.9 show that in all cases, the MSE and MRE increase with an increase in ρ. This is because graph analytic tasks are executed over a noisy graph G'. The added error becomes larger as the graph size increases. We can also observe that FEAT and FEAT+ owns better utility than Baseline and FEAT, respectively, for all ρ.

Figures 4.10 and 4.11 show that Baseline performs the worst while FEAT+ owns the best utility over all values of the overlapping rate σ. Another interesting observation is that the overlapping rate σ has little influence on the overall utility. For instance, Figure 4.10(a) shows that with the increase in σ, the MSE of Baseline increases from 1.014×10^{16} to 1.0442×10^{16}. The slight growth is from multiple perturbations of the same edge information. There are little changes in the results of FEAT and FEAT+ over various σ. This is because the same information is randomized and collected only once.

Execution Time (Q3). Finally, we answer the third question by evaluating the running time over graphs with different sizes. Here, we use different sampling rates ρ to generate multiple graphs with different scales. Figure 4.12 presents the running time of Baseline, FEAT, and FEAT+ for various values of ρ. We can find that the running time increases when the graph scale becomes larger.



(a) Facebook (b) Wiki

Figure 4.12. Running time with various ρ.

This is because when ρ increases, there are more edges collected and computed accordingly. Another important observation is that the running time of FEAT is approximately 10× higher than that of Baseline. This is because the computation of cryptographic techniques leads to additional time overhead. FEAT+ takes more time than FEAT by about 50%. This is because additional communication between clients and the server consumes more time. Thus, utilizing cryptographic tools improves the utility while not leaking sensitive information but at the cost of efficiency.

4.6 Related Works

Federated Analytics. The term "federated analytics" is first introduced by Google in 2020 [113], which is explored in support of federated learning for Google engineers to measure the quality of federated learning models against real-world data. Bharadwaj *et al.* [114] introduces the notion of federated computation, which is a means of working with private data at a rather large scale. Wang *et al.* [115] clarify what federated analytics is and its position in literature and then presents the motivation, application, and opportunities of federated analytics. Elkordy *et al.* [116] gives a comprehensive survey about federated analytics. Nevertheless, they only focus on tabular data analytics, which totally differs from graph analytics in our work. Roth *et al.* [117] introduce Mycelium for large-scale distributed graph queries with differential privacy. Yuan *et al.* [118] define the notion of graph federation for subgraph matching, where graph data sources are temporarily federated. However, they assume that all clients are mutually in-

dependent, which is different from ours. Our (informal) previous work [119]* introduces the concept of federated graph analytics for the first time. However, it encountered privacy leakage issues due to the potential disclosure of intersection information between two clients.

Cross-silo Federated Learning. There exist several works related to cross-silo federated learning [77, 92–94, 120, 121]. Huang *et al.* [92] propose FedAMP that employs federated attentive message passing to facilitate the collaboration effectiveness between clients without infringing their data privacy. Li *et al.* [121] propose a practical one-shot federated learning algorithm by using the knowledge transfer technique. Liu *et al.* [93] empirically show that MR-MTL is a remarkably strong baseline under silo-specific sample-level DP. Tang *et al.* [94] propose an incentive mechanism for cross-silo federated learning, addressing the public goods feature. Zheng *et al.* [77] propose a one-server solution based solely on homomorphic encryption and a two-server protocol based on homomorphic encryption and additive secret sharing, which are designed for contribution evaluation in cross-silo federated learning. However, these protocols are designed for machine learning cannot be used for graph analytics.

Differentially Private Graph Analytics. The standard way to calculate graph statistics is through differential privacy (DP) [14, 54], which is a golden standard in the privacy community. However, existing protocols [2, 3, 5, 7–9, 110] are designed for different scenarios and not applicable to federated graph scenarios. To be specific, central protocols [2, 3] rely on a trusted server to collect the entire graph information from local users and then release accurate graph statistics privately. Nevertheless, a centrally trusted server is amenable to privacy issues in practical such as data leaks and breaches [90, 91]. Instead, local protocols [5, 7] remove the assumption of a trusted server, and each user directly perturbs local sensitive data. However, they cannot protect the user-client membership and faces the edge privacy issues due to the overlaps. Extended local view (ELV)-based protocols [8, 23] consider an extension of local scenarios, where each client can see not only her 1-hop path but also her 2-hop path. Similar to local mechanisms, ELV-based protocols fail to protect the user-client membership. Additionally, decentralized differential privacy in [23] can protect edge privacy in ELV but fails to do it in federated settings since overlaps of ELV are different

*Note that [119] was not published in proceedings in accordance with the policy of the KDD conference, cf. `https://fl4data-mining.github.io/calls/`.

from those of federated graph analytics. In a nutshell, this paper is the first work to formulate the federated graph analytics (FGA) to our best of knowledge. Our proposed FEAT is a general framework for various common graph analytics.

4.7 Conclusion

We made the first attempt to study federated graph analytics with privacy guarantee. We showed unique challenges in federated graph analytics, namely, utility issue from the limited view and privacy issue due to overlaps. To alleviate them, we proposed a general federated graph analytic framework FEAT, based on our proposed differentially private set union protocol. Furthermore, we observed that it calculates graph statistics over a noisy global graph without considering true local subgraphs, and there is still room for improving the overall utility. To address this issue, we introduced an improved framework FEAT+ by combining a noisy global graph with true local subgraphs. Comprehensive experiments verify that our proposed methods significantly outperform the baseline approaches in various graph analytics.

Algorithm 13 CollectGraph: DPSU-based Graph Collection.

Input: Subgraph set $G = \{G_1, ..., G_m\}, G_i = (V_i, E_i)$,
Privacy budget ε

Output: Noisy global graph G'

Initialize: An edge domain $\mathbb{E}$ according to V,
where $N = |\mathbb{E}| = \frac{n(n-1)}{2}$

Server: Initialize $E' = \varnothing$

Each client $C_{i\in[1,m]}$ generates a pair of keys $\langle pk_i, sk_i \rangle$,
where $pk_i = g^{sk_i}$

All clients jointly generate the public key:

$$pk = \prod_{i=1}^{m} pk_i = g^{sk_1 + ... + sk_m}$$

Client C_1: Initialize a flag vector Y,

$$Y_{j\in[1,N]} = \begin{cases} 1, & \mathbb{E}_j \in E_1 \\ 0, & \mathbb{E}_j \notin E_1 \end{cases}$$

Client C_1: Perturb Y with RR,

$$Y'_{j\in[1,N]} = \begin{cases} Y_j & w.p. \frac{e^\varepsilon}{1+e^\varepsilon} \\ 1 - Y_j & w.p. \frac{1}{1+e^\varepsilon} \end{cases}$$

Client C_1: Encrypt Y' with pk to

$$\mathsf{Enc}(Y') = [\mathsf{Enc}(Y'_1), ..., \mathsf{Enc}(Y'_{\tilde{n}})]$$

Send $\mathsf{Enc}(Y')$ to client C_2

for each client $C_i, i \in [2, m]$ **do**
  for each bit $Y'_j, j \in [1, N]$ **do**
    if $\mathbb{E}_j \in E_i$ **then**
      if $\mathsf{RR}(1) == 1$ **then** $\mathsf{Enc}(Y'_j) \leftarrow \mathsf{Enc}(1)$
      else if $\mathsf{RR}(1) == 0$ **then** $\mathsf{Enc}(Y'_j) \leftarrow \mathsf{Enc}(Y'_j)$
    else if $\mathbb{E}_j \notin E_i$ **then**
      if $\mathsf{RR}(0) == 1$ **then** $\mathsf{Enc}(Y'_j) \leftarrow \mathsf{Enc}(1)$
      else if $\mathsf{RR}(0) == 0$ **then** $\mathsf{Enc}(Y'_j) \leftarrow \mathsf{Enc}(Y'_j)$
    end if
  end for
end for

All clients jointly decrypt $\mathsf{Enc}(Y')$ with secret keys:

$$Y' \leftarrow \mathsf{Dec}[\mathsf{Enc}(Y')]$$

Send Y' to server

for each bit $Y', j \in [1, N]$ **do**
  if $Y'_j = 1$ **then** $E' \leftarrow E' \cup \{\mathbb{E}_j\}$
end for

Server: $G' \leftarrow (V, E')$

return G'

Algorithm 14 Overall Protocol of FEAT+.

Input: Subgraph set $G = \{G_1, ..., G_m\}$,
Privacy budget $\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_3$, sensitivity $\triangle$.
Output: Query result Q'

Phase I: Global Graph Collection

$G' \leftarrow \mathsf{FEAT}(G, \varepsilon_1)$ ▷ Algorithm 12

Phase II: Local Query Collection

$U \leftarrow \mathsf{PartitionNode}(G, \varepsilon_2)$ ▷ Step 1: Algorithm 15
for each client $C_{i\in[m]}$ in C **do**
  $Q_i \leftarrow \mathsf{Query}(G', G_i, U_i)$ ▷ Step 2: Section 4.4.3
  $Q'_i = Q_i + \mathsf{Lap}(\frac{\triangle}{\varepsilon_3})$ ▷ Step 3: Perturbation
end for
Server: $Q' \leftarrow \sum_{i=1}^{m} Q'_i$
return Q'

Algorithm 15 PartitionNode: Degree-based Node Partition.

Input: Subgraph set $G = \{V, E\} = \{G_1, ..., G_m\}$,
Privacy budget ε_2
Output: User partition $U = \{U_1, ..., U_m\}$

Initialize: $U = \{U_1, ..., U_m\}$, where $U_{i\in[m]} = \varnothing$
for each node v_i in V **do**
  Client $C_{j\in[1,m]}$: Compute i-th node degree $d_{i,j}$
    Perturb $d'_{i,j} \leftarrow d_{i,j} + \mathsf{Lap}(\frac{m}{\varepsilon_2})$
    Send $d'_{i,j}$ to Server
  Server: Compute $d'_{i,k} \leftarrow \mathsf{max}\{d'_{i,1}, ..., d'_{i,m}\}$
    Obtain index k
    Update $U_k \leftarrow U_k \cup \{i\}$
end for
return U

Algorithm 16 k-Star Counting.

Input: Noisy global graph G', i-th subgraph G_i
i-th user partition U_i.
Output: Number of k-stars S.

Initialize: $d_1 = d_2 = 0$
$\overline{G} \leftarrow G' \cup G_i$
for each node v_j in U_i **do**
  for each friend v_k of v_j in $\overline{G}$ **do**
    if $edge\langle v_j, v_k\rangle$ in G_i **then** $d_1 \leftarrow d_1 + 1$ ▷ True degree in G_i
    else $d_2 \leftarrow d_2 + 1$ ▷ Noisy degree in G'
  end for
end for
$p \leftarrow \frac{1}{1+e^{\varepsilon_1}}$, $\widetilde{d}_2 \leftarrow \frac{1}{1-2p}[d_2 - np]$ ▷ De-bias
$d \leftarrow d_1 + \widetilde{d}_2$, $S \leftarrow \binom{d}{k}$
return S

Algorithm 17 Triangle Counting.

Input: Noisy global graph G', i-th subgraph G_i
i-th user partition U_i.
Output: Number of triangles T.

Initialize: $T_0 = T_1 = T_2 = T_3 = 0$
$\overline{G} \leftarrow G' \cup G_i$
for each node v_j in U_i **do**
  for each friend v_k of v_j in $\overline{G}$ **do**
    for each friend v_l of v_j in $\overline{G}$ **do**
      if $j < k < l$ **then**
        if v_k, v_l are friends in $\overline{G}$ **then**
          Initialize: $\boldsymbol{e} = \{\langle v_j, v_k\rangle, \langle v_j, v_l\rangle, \langle v_k, v_l\rangle\}$
          if 0 edges of $\boldsymbol{e}$ in G_i **then** $T_0 \leftarrow T_0 + 1$
          if 1 edges of $\boldsymbol{e}$ in G_i **then** $T_1 \leftarrow T_1 + 1$
          if 2 edges of $\boldsymbol{e}$ in G_i **then** $T_2 \leftarrow T_2 + 1$
          if 3 edges of $\boldsymbol{e}$ in G_i **then** $T_3 \leftarrow T_3 + 1$
        end if
      end if
    end for
  end for
end for
$(\widetilde{T}_0, \widetilde{T}_1, \widetilde{T}_2) \leftarrow \mathsf{Debias}(T_0, T_1, T_2)$
$T \leftarrow \widetilde{T}_0 + \widetilde{T}_1 + \widetilde{T}_2 + T_3$
return T

CHAPTER 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we focus on crypto-assisted differentially private graph analytics. We present three works to demonstrate how cryptography can improve the trade-off between privacy and utility in differentially private graph analytics. The contributions in the three research topics described in this dissertation are listed as follows:

Topic 1: Crypto-assisted differentially private degree distribution.

- We propose and study the problem of publishing the degree distribution under Node-LDP for the first time. We give a detailed description of the problem definition and conclude the research gap. We present an overview of publishing the degree distribution under Node-LDP.

- We design two methods to select the projection parameter θ in the local setting: pureLDP and crypto-assisted. Crypto-assisted method guarantees the security of individual utility loss with cryptographic primitives, which achieves a higher accuracy than the baseline pureLDP method.

- We design two local graph projection approaches based on different granularity: node-level and edge-level. The improved edge-level method preserves

more information and provides better utility than the baseline node-level method.

- Extensive experiments on real-world graph datasets validate the correctness of our theoretical analysis and the effectiveness of our proposed methods.

Topic 2: Crypto-assisted differentially private triangle counting.

- We propose a crypto-assisted differentially private triangle counting system (CARGO) that (1) achieves the high-utility triangle counting of the central model (2) without a trusted server like the local model.

- We propose a novel local projection method to reduce the sensitivity of the triangle counting while preserving more triangles. We prioritize deleting the edges with the least possibility of constructing triangles, which results in preserving more triangles.

- We introduce a novel triangle counting algorithm based on an additive secret sharing (ASS) technique. We introduce a protocol for multiplying three secret values, which allows us to securely and accurately compute the triangle counts while protecting sensitive neighboring information.

- We present a distributed perturbation method by combining additive secret sharing and distributed noise generation. This partial noise is insufficient to provide an LDP guarantee but the aggregated noise is enough to provide a CDP protection.

- We provide a comprehensive theoretical analysis of our proposed protocols, including utility, privacy, and time complexity. Our CARGO achieves high-utility triangle counts comparable to central models, and significantly outperforms local models by at least an order of 5 in utility.

Topic 3: Crypto-assisted differentially private federated graph analytics.

- We investigate the federated graph analytics (FGA) under DP for the first time. By comparing with previous protocols, we conclude unique challenges in FGA.

- We present a generalized federated graph analytic framework with differential privacy (FEAT) based on our proposed DPSU protocol, which supports a wide range of common graph statistics, e.g., subgraph counting.
- We introduce an optimized framework (FEAT+) based on our proposed degree-based partition algorithm, which improves the overall utility by leveraging true subgraphs.
- We verify the effectiveness of our proposed methods through extensive experiments. FEAT reduces the error than baseline approach by up to an order of 4. FEAT+ outperforms FEAT by at least an order of 1.

5.2 Future Directions

The three research topics in this thesis aim to inspire greater interest and attention in crypto-assisted differentially private graph analytics. Several promising research avenues can be explored in future work.

- *Graph Synthesis.* The goal of private graph synthesis is to publish a synthetic graph that is semantically similar to the original graph while satisfying differential privacy (DP). This approach is superior to tailored algorithms because it enables arbitrary downstream graph data analysis tasks. Previous works rely on either the central DP model or the local DP model, resulting in a significant utility-privacy gap between CDP-based and LDP-based differentially private graph synthesis. This motivates us to explore how to achieve high-utility of the central model without the need for a trusted server like the local model, by leveraging cryptographic techniques.
- *Attributed Graph.* Social networks encode complex relationships among individuals (e.g., friendships, acquaintances, sexual relationships, disease transmission), which can be sensitive. Additionally, the nodes (e.g., users) in real-world social networks may be associated with various sensitive attributes, such as age, location, or sexual preference. Most prior works have focused primarily on the graph structure in isolation and have not provided methods to handle richer graphs with correlated attributes. There is significant potential to explore how cryptography can enhance differentially private attributed graph analytics.

- *Dynamic Graph.* Dynamic graphs better capture the temporal evolution and properties of networks. While several differentially private mechanisms have been proposed for static graph data mining, there are currently few algorithms for protecting and mining dynamic data. This motivates us to design a method for differentially private dynamic graph analytics. Furthermore, we can explore how to improve the utility of dynamic graph analytics by combining differential privacy (DP) and cryptography.

- *Benchmark on Private Graph Synthesis.* Differentially private graph synthesis has garnered significant attention from researchers. Generally, a pipeline for private graph synthesis includes steps such as graph representation, perturbation, generation, and evaluation, with many candidate methods for each step. For example, a graph can be represented as an adjacency matrix, degree information, or neighbor lists. Common perturbation mechanisms include the Laplace mechanism, randomized response mechanism, and Exponential mechanism. There are various graph generator models, such as the CL model, BTER model, and ER model. Different combinations of these methods can be suitable for different cases. This motivates us to design a benchmark for differentially private graph synthesis to yield more interesting findings.

Acknowledgements

This dissertation represents not just my efforts but a journey made possible by the invaluable support and inspiration from many. I am deeply grateful to all who have guided and aided me along this path.

First and foremost, I would like to express my deepest gratitude to my supervisors, Professor Yang Cao, Professor Masatoshi Yoshikawa, and Professor Takayuki Ito, for their invaluable guidance, patience, and support throughout the course of this research. Your expertise and insightful critiques have been instrumental in shaping this work.

My sincere thanks also go to my advisors, Professor Keishi Tajima and Professor Masayuki Abe, for their constructive feedback and encouragement at crucial stages of my research. My special thanks go to Associate Professor Takao Murakami, who is now at the Department of Interdisciplinary Statistical Mathematics in the Institute of Statistical Mathematics. He provided many invaluable comments and suggestions on my research work.

I am grateful to my colleagues and friends at Kyoto University, Hokkaido University, and Tokyo Institute of Technology, for their companionship and exchange of ideas. A special word of thanks goes to the Kyoto University Division of Graduate Studies SPRING Program for their financial support, without which this research would not have been possible.

Finally, I must acknowledge the support of my family, who have provided me with the motivation and strength needed to pursue my academic goals. Special thanks to my wife, Mrs. Xinru Ge, for her continuous understanding and encouragement.

Shang Liu, June 2024

References

[1] Xun Jian, Yue Wang, and Lei Chen. Publishing graphs under node differential privacy. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[2] Xiaofeng Ding, Shujun Sheng, Huajian Zhou, Xiaodong Zhang, Zhifeng Bao, Pan Zhou, and Hai Jin. Differentially private triangle counting in large graphs. *IEEE Transactions on Knowledge and Data Engineering*, 34(11):5278–5292, 2021.

[3] Wei-Yen Day, Ninghui Li, and Min Lyu. Publishing graph degree distribution with node differential privacy. In *Proceedings of the 2016 International Conference on Management of Data*, pages 123–138, 2016.

[4] Quan Yuan, Zhikun Zhang, Linkang Du, Min Chen, Peng Cheng, and Mingyang Sun. PrivGraph: Differentially private graph data publication by exploiting community information. In *USENIX Security Symposium*, 2023.

[5] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. Locally differentially private analysis of graph statistics. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 983–1000, 2021.

[6] Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. Lf-gdpr: A framework for estimating graph metrics with local differential privacy. *IEEE Transactions on Knowledge and Data Engineering*, 34(10):4905–4920, 2020.

[7] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. Communication-

efficient triangle counting under local differential privacy. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 537–554, 2022.

[8] Yuhan Liu, Suyun Zhao, Yixuan Liu, Dan Zhao, Hong Chen, and Cuiping Li. Collecting triangle counts with edge relationship local differential privacy. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2008–2020. IEEE, 2022.

[9] Shang Liu, Yang Cao, Takao Murakami, and Masatoshi Yoshikawa. A crypto-assisted approach for publishing graph statistics with node local differential privacy. In *IEEE Big Data*, pages 5765–5774, 2022.

[10] Shang Liu, Yang Cao, Takao Murakami, Jinfei Liu, and Masatoshi Yoshikawa. Cargo: Crypto-assisted differentially private triangle counting without trusted servers. *arXiv preprint arXiv:2312.12938*, 2023.

[11] Mark EJ Newman. Random graphs with clustering. *Physical review letters*, 103(5):058701, 2009.

[12] Vishesh Karwa, Sofya Raskhodnikova, Adam Smith, and Grigory Yaroslavtsev. Private analysis of graph structure. *Proc. VLDB Endow.*, 4(11):1146–1157, aug 2011.

[13] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[14] Ninghui Li, Min Lyu, Dong Su, and Weining Yang. Differential privacy: From theory to practice. *Synthesis Lectures on Information Security, Privacy, & Trust*, 8(4):1–138, 2016.

[15] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography Conference*, pages 457–476. Springer, 2013.

[16] Sofya Raskhodnikova and Adam Smith. Lipschitz extensions for node-private graph statistics and the generalized exponential mechanism. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 495–504. IEEE, 2016.

[17] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. Differentially private data analysis of social networks via restricted sensitivity. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 87–96, 2013.

[18] Qing Qian, Zhixu Li, Pengpeng Zhao, Wei Chen, Hongzhi Yin, and Lei Zhao. Publishing graph node strength histogram with edge differential privacy. In *International Conference on Database Systems for Advanced Applications*, pages 75–91. Springer, 2018.

[19] Michael Hay, Chao Li, Gerome Miklau, and David Jensen. Accurate estimation of the degree distribution of private networks. In *2009 Ninth IEEE International Conference on Data Mining*, pages 169–178. IEEE, 2009.

[20] Vishesh Karwa and Aleksandra B Slavković. Differentially private graphical degree sequences and synthetic graphs. In *International Conference on Privacy in Statistical Databases*, pages 273–285. Springer, 2012.

[21] Davide Proserpio, Sharon Goldberg, and Frank McSherry. A workflow for differentially-private graph synthesis. In *Proceedings of the 2012 ACM workshop on Workshop on online social networks*, pages 13–18, 2012.

[22] Vishesh Karwa, Sofya Raskhodnikova, Adam Smith, and Grigory Yaroslavtsev. Private analysis of graph structure. *Proceedings of the VLDB Endowment*, 4(11):1146–1157, 2011.

[23] Haipei Sun, Xiaokui Xiao, Issa Khalil, Yin Yang, Zhan Qin, Hui Wang, and Ting Yu. Analyzing subgraph statistics from extended local views with decentralized differential privacy. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 703–717, 2019.

[24] Sen Zhang, Weiwei Ni, and Nan Fu. Community preserved social graph publishing with node differential privacy. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1400–1405. IEEE, 2020.

[25] Chengkun Wei, Shouling Ji, Changchang Liu, Wenzhi Chen, and Ting Wang. Asgldp: collecting and generating decentralized attributed graphs

with local differential privacy. *IEEE Transactions on Information Forensics and Security*, 15:3239–3254, 2020.

[26] Christine Task and Chris Clifton. A guide to differential privacy theory in social network analysis. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 411–417. IEEE, 2012.

[27] Yang Li, Michael Purcell, Thierry Rakotoarivelo, David Smith, Thilina Ranbaduge, and Kee Siong Ng. Private graph data release: A survey. *arXiv preprint arXiv:2107.04245*, 2021.

[28] Mengmeng Yang, Lingjuan Lyu, Jun Zhao, Tianqing Zhu, and Kwok-Yan Lam. Local differential privacy and its applications: A comprehensive survey. *arXiv preprint arXiv:2008.03686*, 2020.

[29] John C Duchi, Michael I Jordan, and Martin J Wainwright. Local privacy and statistical minimax rates. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 429–438. IEEE, 2013.

[30] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[31] C. Wei, S. Ji, C. Liu, W. Chen, and T. Wang. Asgldp: Collecting and generating decentralized attributed graphs with local differential privacy. *IEEE Transactions on Information Forensics and Security*, 15:3239–3254, 2020.

[32] Zhan Qin, Ting Yu, Yin Yang, Issa Khalil, Xiaokui Xiao, and Kui Ren. Generating synthetic decentralized social graphs with local differential privacy. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 425–438, 2017.

[33] Mohamed Alie Kamara and Xudong Li. A review of order preserving encryption schemes. In *The International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery*, pages 707–715. Springer, 2020.

[34] Anselme Tueno and Florian Kerschbaum. Efficient secure computation of order-preserving encryption. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 193–207, 2020.

[35] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[36] Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. Lf-gdpr: A framework for estimating graph metrics with local differential privacy. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[37] Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

[38] Hailong Zhang, Sufian Latif, Raef Bassily, and Atanas Rountev. Differentially-private control-flow node coverage for software usage analysis. In *USENIX Security Symposium (USENIX Security)*, 2020.

[39] Daniel Kifer and Ashwin Machanavajjhala. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, page 193–204, New York, NY, USA, 2011. Association for Computing Machinery.

[40] Raluca Ada Popa, Frank H Li, and Nickolai Zeldovich. An ideal-security protocol for order-preserving encoding. In *2013 IEEE Symposium on Security and Privacy*, pages 463–477. IEEE, 2013.

[41] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[42] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574, 2004.

[43] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Annual Cryptology Conference*, pages 578–595. Springer, 2011.

[44] Florian Kerschbaum and Axel Schröpfer. Optimal average-complexity ideal-security order-preserving encryption. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 275–286, 2014.

[45] Daniel S Roche, Daniel Apon, Seung Geol Choi, and Arkady Yerukhimovich. Pope: Partial order preserving encoding. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1131–1142, 2016.

[46] Dongxi Liu and Shenlu Wang. Programmable order-preserving secure index for encrypted database query. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 502–509, 2012.

[47] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudorandom bits. *SIAM journal on Computing*, 13(4):850–864, 1984.

[48] Yuxuan Zhang, Jianghong Wei, Xiaojian Zhang, Xuexian Hu, and Wenfen Liu. A two-phase algorithm for generating synthetic graph under local differential privacy. In *Proceedings of the 8th International Conference on Communication and Network Security*, pages 84–89, 2018.

[49] Zach Jorgensen, Ting Yu, and Graham Cormode. Publishing attributed social graphs with formal privacy guarantees. In *Proceedings of the 2016 international conference on management of data*, pages 107–122, 2016.

[50] Comandur Seshadhri and Srikanta Tirthapura. Scalable subgraph counting: the methods behind the madness. In *Companion Proceedings of The 2019 World Wide Web Conference*, pages 1317–1318, 2019.

[51] Thomas Schank and Dorothea Wagner. Approximating clustering coefficient and transitivity. *Journal of Graph Algorithms and Applications*, 9(2):265–275, 2005.

[52] TE GOLDSMITH. Assessing structural similarity of graphs. *Pathfinder Associative Networks: Studies in Knowledge Organization*, pages 75–87, 1990.

[53] Chih-Hua Tai, Philip S Yu, De-Nian Yang, and Ming-Syan Chen. Privacy-preserving social network publication against friendship attacks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1262–1270, 2011.

[54] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[55] Xi He, Ashwin Machanavajjhala, Cheryl Flynn, and Divesh Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1389–1406, 2017.

[56] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. Cryptε: Crypto-assisted differential privacy on untrusted servers. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 603–619, 2020.

[57] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. Honeycrisp: large-scale differentially private aggregation without a trusted core. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 196–210, 2019.

[58] Xiaolan Gu, Ming Li, and Li Xiong. Precad: Privacy-preserving and robust federated learning via crypto-aided differential privacy. *arXiv preprint arXiv:2110.11578*, 2021.

[59] Timothy Stevens, Christian Skalka, Christelle Vincent, John Ring, Samuel Clark, and Joseph Near. Efficient differentially private secure aggregation for federated learning via hardness of learning with errors. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1379–1395, 2022.

[60] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM workshop on artificial intelligence and security*, pages 1–11, 2019.

[61] Lichao Sun and Lingjuan Lyu. Federated model distillation with noise-free differential privacy. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1563–1570, 2021.

[62] Congcong Fu, Hui Li, Jian Lou, and Jiangtao Cui. Dp-horus: Differentially private hierarchical count histograms under untrusted server. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 530–539, 2022.

[63] Nurcan Durak, Ali Pinar, Tamara G Kolda, and C Seshadhri. Degree relations of triangles in real-world networks and graph models. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1712–1716, 2012.

[64] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[65] Elaine Shi, HTH Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Annual Network & Distributed System Security Symposium (NDSS)*. Internet Society., 2011.

[66] Gergely Ács and Claude Castelluccia. I have a dream!(differentially private smart metering). In *Information hiding*, volume 6958, pages 118–132. Springer, 2011.

[67] Slawomir Goryczka and Li Xiong. A comprehensive comparison of multiparty secure additions with differential privacy. *IEEE transactions on dependable and secure computing*, 14(5):463–477, 2015.

[68] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. Aby2. 0: Improved mixed-protocol secure two-party computation. In *USENIX Security Symposium*, pages 2165–2182, 2021.

[69] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 325–342, 2020.

[70] Takao Murakami and Yusuke Kawamoto. {Utility-Optimized} local differential privacy mechanisms for distribution estimation. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1877–1894, 2019.

[71] Tianhao Wang, Jeremiah Blocki, Ninghui Li, and Somesh Jha. Locally differentially private protocols for frequency estimation. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 729–745, 2017.

[72] Rui Chen, Gergely Acs, and Claude Castelluccia. Differentially private sequential data publication via variable-length n-grams. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 638–649, 2012.

[73] Vincent Bindschaedler and Reza Shokri. Synthesizing plausible privacy-preserving location traces. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 546–563. IEEE, 2016.

[74] Sofya Raskhodnikova and Adam Smith. Differentially private analysis of graphs. *Encyclopedia of Algorithms*, 2016.

[75] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38. IEEE, 2017.

[76] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia conference on computer and communications security*, pages 707–721, 2018.

[77] Shuyuan Zheng, Yang Cao, and Masatoshi Yoshikawa. Secure shapley value for cross-silo federated learning. *Proceedings of the VLDB Endowment*, 16(7):1657–1670, 2023.

[78] Shang Liu, Yang Cao, Takao Murakami, and Masatoshi Yoshikawa. A crypto-assisted approach for publishing graph statistics with node local differential privacy. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 5765–5774, 2022.

[79] Michael O Rabin. How to exchange secrets with oblivious transfer. *Cryptology ePrint Archive*, 2005.

[80] Joe Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31, 1988.

[81] Samuel Kotz, Tomasz Kozubowski, and Krzysztof Podgórski. *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*. Number 183. Springer Science & Business Media, 2001.

[82] Yehuda Lindell. How to simulate it–a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pages 277–346, 2017.

[83] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[84] Wei Dong and Ke Yi. A nearly instance-optimal differentially private mechanism for conjunctive queries. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 213–225, 2022.

[85] W. Dong and K. Yi. Residual sensitivity for differentially private multi-way joins. In *SIGMOD/PODS'21: Proceedings of the 2021 International Conference on Management of Data*, 2021.

[86] Songlei Wang, Yifeng Zheng, Xiaohua Jia, Qian Wang, and Cong Wang. Mago: Maliciously secure subgraph counting on decentralized social graphs. *IEEE Transactions on Information Forensics and Security*, 2023.

[87] Yulin Wu and Lanxiang Chen. Structured encryption for triangle counting on graph data. *Future Generation Computer Systems*, 145:200–210, 2023.

[88] Dawei Cheng, Fangzhou Yang, Sheng Xiang, and Jin Liu. Financial time series forecasting with multi-modality graph neural network. *Pattern Recognition*, 121:108218, 2022.

[89] Stephanie J Dancer. Reducing the risk of covid-19 transmission in hospitals: focus on additional infection control strategies. *Surgery (Oxford)*, 39(11):752–758, 2021.

[90] Nelson Novaes Neto, Stuart Madnick, Anchises Moraes G De Paula, and Natasha Malara Borges. Developing a global data breach database and the challenges encountered. *JDIQ*, 13(1):1–33, 2021.

[91] Brandon Gibson, Spencer Townes, Daniel Lewis, and Suman Bhunia. Vulnerability in massive api scraping: 2021 linkedin data breach. In *IEEE CSCI*, 2021.

[92] Yutao Huang, Lingyang Chu, Zirui Zhou, Lanjun Wang, Jiangchuan Liu, Jian Pei, and Yong Zhang. Personalized cross-silo federated learning on non-iid data. In *Proc. AAAI*, volume 35, pages 7865–7873, 2021.

[93] Ken Liu, Shengyuan Hu, Steven Z Wu, and Virginia Smith. On privacy and personalization in cross-silo federated learning. *Advances in Neural Information Processing Systems*, 35:5925–5940, 2022.

[94] Ming Tang and Vincent WS Wong. An incentive mechanism for cross-silo federated learning: A public goods perspective. In *IEEE INFOCOM*, pages 1–10, 2021.

[95] Cong Zhang, Yu Chen, Weiran Liu, Min Zhang, and Dongdai Lin. Linear private set union from {Multi-Query} reverse private membership test. In *USENIX Security Symposium*, pages 337–354, 2023.

[96] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. Shuffle-based private set union: Faster and more secure. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2947–2964, 2022.

[97] Wenli Wang, Shundong Li, Jiawei Dou, and Runmeng Du. Privacy-preserving mixed set operations. *Information Sciences*, 525:67–81, 2020.

[98] Changyu Dong and Grigorios Loukides. Approximating private set union/intersection cardinality with logarithmic complexity. *IEEE Transactions on Information Forensics and Security*, 12(11):2792–2806, 2017.

[99] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. {BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning. In *USENIX ATC*, pages 493–506, 2020.

[100] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. A secure federated transfer learning framework. *IEEE Intelligent Systems*, 35(4):70–82, 2020.

[101] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Springer TCC*, pages 265–284, 2006.

[102] Whitfield Diffie and Martin E Hellman. New directions in cryptography. In *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, pages 365–390. 2022.

[103] Elaine B Barker, William C Barker, William E Burr, W Timothy Polk, and Miles E Smid. Sp 800-57. recommendation for key management, part 1: General (revised), 2007.

[104] Darrel Hankerson and Alfred Menezes. Elliptic curve cryptography. In *Encyclopedia of Cryptography, Security and Privacy*, pages 1–2. Springer, 2021.

[105] Sivakanth Gopi, Pankaj Gulhane, Janardhan Kulkarni, Judy Hanwen Shen, Milad Shokouhi, and Sergey Yekhanin. Differentially private set union. In *International Conference on Machine Learning*, pages 3627–3636. PMLR, 2020.

[106] Kunho Kim, Sivakanth Gopi, Janardhan Kulkarni, and Sergey Yekhanin. Differentially private n-gram extraction. *Advances in Neural Information Processing Systems*, 34:5102–5111, 2021.

[107] Ricardo Silva Carvalho, Ke Wang, and Lovedeep Singh Gondara. Incorporating item frequency for differentially private set union. In *Proc. AAAI*, volume 36, pages 9504–9511, 2022.

[108] XUE Qiao, ZHU Youwen, and Xingxin LI Jian WANG. Locally differentially private distributed algorithms for set intersection and union. *Information Sciences*, 64(219101):1–219101, 2021.

[109] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[110] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. Differentially private triangle and 4-cycle counting in the shuffle model. In *ACM CCS*, 2022.

[111] Kalyan Das, Jiming Jiang, and JNK Rao. Mean squared error of empirical predictor. 2004.

[112] Tron Foss, Ingunn Myrtveit, and Erik Stensrud. Mre and heteroscedasticity: An empirical validation of the assumption of homoscedasticity of the magnitude of relative error. In *Proc. ESCOM*, pages 157–164, 2001.

[113] D. Ramage and S. Mazzocchi. Federated analytics: Collaborative data science without data collection. `https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html`, 2020.

[114] Akash Bharadwaj and Graham Cormode. An introduction to federated computation. In *Proceedings of the 2022 International Conference on Management of Data*, pages 2448–2451, 2022.

[115] Dan Wang, Siping Shi, Yifei Zhu, and Zhu Han. Federated analytics: Opportunities and challenges. *IEEE Network*, 36(1):151–158, 2021.

[116] Ahmed Roushdy Elkordy, Yahya H Ezzeldin, Shanshan Han, Shantanu Sharma, Chaoyang He, Sharad Mehrotra, Salman Avestimehr, et al. Federated analytics: A survey. *APSIPA Transactions on Signal and Information Processing*, 12(1), 2023.

[117] Edo Roth, Karan Newatia, Yiping Ma, Ke Zhong, Sebastian Angel, and Andreas Haeberlen. Mycelium: Large-scale distributed graph queries with differential privacy. In *Proc. SOSP*, pages 327–343, 2021.

[118] Ye Yuan, Delong Ma, Zhenyu Wen, Zhiwei Zhang, and Guoren Wang. Subgraph matching over graph federation. *Proceedings of the VLDB Endowment*, 15(3):437–450, 2021.

[119] Shang Liu, Yang Cao, and Masatoshi Yoshikawa. Federated graph analytics with differential privacy. In *International Workshop on Federated Learning for Distributed Data Mining*, 2023 (informal paper not published in proceedings, cf., `https://fl4data-mining.github.io/calls/`.

[120] Yansheng Wang, Yongxin Tong, Dingyuan Shi, and Ke Xu. An efficient approach for cross-silo federated learning to rank. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1128–1139. IEEE, 2021.

[121] Qinbin Li, Bingsheng He, and Dawn Song. Practical one-shot federated learning for cross-silo setting.

Selected List of Publications

- **Journals**

  [1] Shang Liu, Yang Cao, Takao Murakami, Weiran Liu, Seng Pei Liew, Tsubasa Takahashi, Jinfei Liu, Masatoshi Yoshikawa. Federated Graph Analytics with Differential Privacy. *arXiv preprint*, May 2024 (Submitted to IEEE Transactions on Dependable and Secure Computing).

- **International Conferences and Workshops**

  [2] Shang Liu, Yang Cao, Takao Murakami, Masatoshi Yoshikawa. A crypto-assisted approach for publishing graph statistics with node local differential privacy. *Proceedings of 2022 IEEE International Conference on Big Data (Big Data 2022)*, pp. 5765-5774, Osaka, Japan, December 2022.

  [3] Shang Liu, Yang Cao, Masatoshi Yoshikawa. Federated Graph Analytics with Differential Privacy. *International Workshop on Federated Learning for Distributed Data Mining (KDD FL4Data-Mining 2023)*, Long Beach, USA, August 2023.

  [4] Shang Liu, Yang Cao, Takao Murakami, Jinfei Liu, Masatoshi Yoshikawa. CARGO: Crypto-Assisted Differentially Private Triangle Counting without Trusted Servers. *Proceedings of 40th IEEE International Conference on Data Engineering (ICDE 2024)*, Utrecht, Netherlands, May 2024.