Counting Edges, Wedges, and Triangles over Fully Distributed Graphs with Distributed Trust

Journal:	*Transactions on Dependable and Secure Computing*
Manuscript ID	TDSC-2023-07-0667
Manuscript Type:	Regular Paper
Keywords:	B.1.1 Control Design Styles < B.1 Control Structures and Microprogramming < B Hardware, C.2.4.c Distributed databases < C.2.4 Distributed Systems < C.2 Communication/Networking and Information Technology < C Computer

Counting Edges, Wedges, and Triangles over Fully Distributed Graphs with Distributed Trust

Pinghui Wang, *Senior Member, IEEE*, Dongxu Zeng, Shiqi Lou, Haoxin Yang, and Tao Qin

Abstract—Graph pattern mining is vital for understanding complex graph structures, including counting edges, wedges, and triangles. In practice, graphs are frequently fully distributed, with each node (e.g., a mobile phone user) maintaining local connections (e.g., phone contacts) within the larger graph (e.g., phone contact network). Privacy concerns prevent centralizing the graph by gathering each node's connections. Existing research primarily concentrates on local differential privacy (LDP) mechanisms for computing subgraphs in large-scale graphs, overlooking the unique challenges that smaller graphs present. In this paper, we reveal the limitations of LDP-based techniques, which result in significant estimation inaccuracies for small to medium-sized graphs (e.g., those with fewer than 4,000,000 nodes). To address these limitations, we novelly introduce secure multi-party computation (MPC) protocols to effectively count edges, wedges, and triangles in fully distributed graphs. These protocols can safeguard against differential attacks while preserving the privacy of individual nodes' connections. Our comprehensive evaluations on a diverse range of real-world and synthetic datasets demonstrate the enhanced performance and effectiveness of our proposed protocols. Most notably, our triangle counting protocol exhibits a significant 100 times improvement in accuracy over baseline methods, solidifying our work as a compelling addition to the field of graph pattern mining.

Index Terms—triangle counting, differential privacy, secure multi-party computation.

1 Introduction

Graphs are a powerful and versatile data structure employed across a wide variety of applications in computer science, business, science, medicine, and other fields. Social networks, online shopping networks, and countless other real-world networks can be effectively modeled as graphs that capture interactions (i.e., edges) between entities (i.e., nodes). Owing to their ubiquity, graphs form the backbone of numerous systems, and mining graph structures is essential for comprehending the corresponding real-world scenarios.

The process of counting edges, wedges, and triangles in graphs holds paramount importance in subgraph counting and graph data mining. This process unveils critical insights into a graph's structure, such as clustering coefficients, transitivity, network motifs, and community structures. Clustering coefficients measure the extent to which nodes in a graph tend to cluster together, signaling the presence of tightly connected groups. Transitivity quantifies the likelihood of adjacent vertices of a vertex being interconnected, which can help identify strongly connected substructures. Network motifs are small subgraphs that recur more frequently in the graph than expected by chance, serving as building blocks for the larger network architecture. Community structures represent groups of nodes with dense internal connections and sparse external connections, which can assist in understanding the organization and function of various networks. By counting edges, wedges, and triangles, we can efficiently analyze these properties and gain a deeper understanding of the underlying graph's characteristics.

In real-world applications, the graph of interest may sometimes be presented in a fully distributed manner. Specifically, each node (e.g., a user) maintains its neighbor list locally and is unaware of any edges beyond its connections to neighboring nodes. Users will share only non-identifying summary statistics rather than their contact lists or local networks. Disclosing edge data between users could lead to severe and even legal repercussions for the owner. Imagine a skilled hacker gaining access to users' financial transactions, social media connections, and phone call records, all fully distributed graphs. It would be easy for them to deduce highly sensitive information about a specific user, such as their financial status, LGBT affiliation, or AIDS history. Owing to these privacy concerns, people are generally reluctant to release their contact lists. Furthermore, modern legislation prohibits organizations from constructing an entire graph by centrally collecting and storing users' neighbor lists. As a result, each node retains a local view of the distributed graph, necessitating secure solutions for mining the graph without exposing any node's connections.

Our research aims to address these challenges by developing reliable and secure protocols to mine fully distributed graphs, without compromising privacy. Recent works have leveraged centralized differential privacy (CDP) to protect users' personal information under the assumption of a trusted third party. However, centralized data holders are prone to attacks by adversaries, potentially resulting in extensive privacy breaches. Thus, an increasing number of researchers are investigating the implementation of differential privacy algorithms on local clients (in short, LDP). For instance, Imola et al. [1] proposed a two-round interaction strategy in local differential privacy, significantly reducing

• *P. Wang, D. Zeng, S. Lou, H. Yang, and T. Qin, are with the MOE Key Laboratory for Intelligent Networks and Network Security, Xi'an Jiaotong University, P.O. Box 1088, No. 28, Xianning West Road, Xi'an, Shaanxi 710049, China. E-mail: {phwang, qin.tao, }@xjtu.edu.cn, {754019499,1229616803}@qq.com.*

estimation error for triangle counting tasks. Liu et al. [2] considered the relationship between nodes and collected them to enhance data usability under LDP settings. Sun et al. [3] presented a multi-phase framework to protect each node's information and the privacy of its 2-hop neighbors.

Despite the advances in graph mining techniques, current methods still exhibit significant estimation errors, particularly in small and medium-sized graphs. For instance, experimental results from Imola et al. [4] showed that the relative error of triangle count estimation was approximately 10 in a graph sampled from the Gplus dataset containing 10^4 nodes. Moreover, in another study [1], the relative error reached 5 for a graph with 2×10^4 nodes from the Gplus dataset and skyrocketed to 100 for a graph with 2×10^5 nodes from the Orkut dataset. These errors render such methods impractical for real-world applications. In many cases, the graphs of interest are equal to or smaller than the sizes mentioned above. Examples include graphs representing individuals within a community, such as students in a school, users of a newly-released phone app, patients in a suburban clinic, scholars within a research community, university faculty, or company employees. These networks generally comprise a significantly smaller number of nodes, often less than a million or even just thousands. Existing methods struggle to achieve adequate accuracy on these smaller networks, resulting in a pressing need for more precise and reliable graph mining techniques.

To address the limitations of existing LDP-based methods for characterizing fully distributed graphs with small and medium sizes, we propose novel protocols based on secure multi-party computation (MPC). We utilize several non-colluding servers to gather information about each node's connections. Each server holds a piece (i.e., a secret share) of each node's edge/subgraph statistic, and a server cannot infer any information about a node from the node's secret share sent to the server. Based on the collected secret shares, we propose efficient protocols to compute the number of edges, wedges, and triangles using secure multi-party computations. Our main contributions are summarized as follows:

- To the best of our knowledge, we are the first to propose MPC-based protocols for counting the edges, wedges, and triangles in fully distributed graphs. We also include effective differential noise to protect against differential attacks.
- We theoretically analyze the performance of our MPC-based protocols. We show that they complement computationally lightweight LDP-based protocols and minimize estimation error for small and medium-sized graphs (e.g., graphs with fewer than 4,000,000 nodes). However, our protocols require extensive computation for larger graphs.
- We conduct extensive experiments on various real-world and synthetic graph datasets. We demonstrate that our protocols outperform baseline methods in accuracy by at least two orders of magnitude under comparable or even lower computational costs for small and medium-sized graphs.

The remainder of this paper is organized as follows. Section 2 presents the problem formulation. Section 3 describes our MPC protocols for counting the edges, wedges, and triangles. Section 4 discusses performance evaluation and testing results. Section 5 summarizes related work. The conclusion is provided in Section 6.

TABLE 1: Table of notations.

G	the undirected graph of interest
V	the set of nodes in graph G, that is, $V = \{1, \ldots, n\}$
E	the set of edges in graph G
$f(G)$	the real number of subgraphs in graph G
$f^{'}(G)$	the estimated number of subgraphs in graph G
n	the number of nodes in graph G
m	the number of edges in graph G
$\wedge$	the number of wedges in graph G
Δ	the number of triangles in graph G
N_i	the set of node i's neighbors
d_i	the degree of node i, i.e., $d_i = \|N_i\|$
A_{ij}	(i, j) entry of the adjacency matrix A of graph G
W_{ij}	the number of wedges including endpoints i, j
c	the number of servers used in our MPC protocols
$[\![x]\!]_s$	the secret share of a variable x holding by server s
L	L is defined as $L = \frac{1}{2}n(n-1)$
Δ_f	the sensitivity of function f

2 Problem Formulation

Notation. Let $G = (V, E)$ be the undirected graph of interest, where $V = \{1, \ldots, n\}$ and E are the node set and the edge set, respectively. Each node $i \in V$ represents a user (e.g., a community member V). For each user i, the set of its neighbors (e.g., friends in the community), i.e., $N_i = \{j : (i, j) \in E\}$ is held by user i locally. Denote d_i as the number of different neighbors of user i, i.e., $d_i = |N_i|$. Let m denote the number of edges, i.e., $m = |E|$. Denote by $\wedge$ the number of wedges (i.e.,$\{(u, v, w) : (u, v), (v, w) \in E, u, v, w \in V\}$, and Δ the number of the triangles (i.e.,$\{\{u, v, w\} : (u, v), (u, w), (v, w) \in E, u, v, w \in V\}$). We summarize notations used throughout the paper in Table 1.
Target. We aim to design a protocol for privately counting m (i.e., the number of edges), $\wedge$ (i.e., the number of wedges), and Δ (i.e., the number of triangles) for graph G without leaking information about any node i's neighbor list N_i.
Threat models. Each user $i \in V$ is semi-honest (i.e., honest yet curious). In addition, there exist c non-colluded data-collection servers where each server is also semi-honest. In other words, a participant will not maliciously poison data to attack the protocol but intend to infer sensitive information from other participants. The non-colluding assumption is widely adopted in recent lightweight privacy-preserving systems built on distributed trust including academic systems [5], [6], [7], [8], [9], [10], [11], [12], [13] and industrial systems [14], [15]. Our protocols are designed to provide the following security guarantees:
• **Completeness.** If all users and all data-collection servers honestly follow the protocol, then the results given by our protocols are correct.

Algorithm 1 Function SecShare(x,c).

```
Input: Secret variable: $x$; the number of shares $c$.
Output: $c$ shares of variable $x$.
Initialize a large $p$;
for $i=1,\dots,c-1$ do
    randomly generate a variable $[\![x]\!]_i$ from field $\mathbb{F}_p$;
Compute $[\![x]\!]_c \leftarrow (x - \sum_{i=1}^{c-1}[\![x]\!]_i) \mod p$;
Output $([\![x]\!]_1, \dots, [\![x]\!]_c)$.
```

• **Privacy against the semi-honest users and servers.** Each user and each server cannot infer any confidential information linked to any user from the output and the intermediate results of our protocols.

3 MPC-BASED PROTOCOLS

In this section, we elaborate on our protocols for securely computing the total number of edges, wedges, and triangles in an undirected graph $G = (V, E)$, respectively.

3.1 Protocol for Counting Edges

Basic Idea. For each user $i \in V$, let d_i^* denote the number of its neighbors with ID greater than i, i.e.,

$$d_i^* = |\{j : j \in N_i, j > i\}|.$$

It is not difficult to find that the number of edges in graph G can be computed as $m = \sum_{i=1}^{n} d_i^*$. Based on the above observation, we let each user $i \in V$ secretly share its d_i^* to the data-collection servers and then compute the value of m based on aggregating secret shares of all d_i^*. Next, we introduce our protocol MPC-CE for implementing the above idea. The pseudo-code of our MPC-CE and the required secret sharing algorithms are given in Algorithms 1 and 2.

Data Collection. We first introduce how a user $i \in V$ securely shares its d_i^* to the c servers. As shown in Algorithm 1, to secretly share an integer x in a field $\mathbb{F}_p = \{0, \dots, p-1\}$ over c data-collection servers, where p is a large prime, the user first generates random variables $[\![x]\!]_1, \dots, [\![x]\!]_{c-1}$ selected independently and uniformly from field $\mathbb{F}_p$. Each of $[\![x]\!]_1, \dots, [\![x]\!]_c$ is a secret share of x. Then, it computes $[\![x]\!]_c = x - \sum_{s=1}^{c-1}[\![x]\!]_s \mod p$. Finally, the user sends $[\![x]\!]_1, \dots, [\![x]\!]_c$ to servers 1, ..., c, respectively.

Using the above method, each user i securely shares its degree d_i with all the servers.

Data Aggregation. Each server $s \in \{1, \dots, c\}$ computes its secret share of the number of edges m as: $[\![m]\!]_s = \sum_{i=1}^{n}[\![d_i^*]\!]_s$. It is not difficult to find that $[\![m]\!]_s$ can be computed incrementally. In other words, each server s can update $[\![m]\!]_s$ as soon as receiving a user i's shares $[\![d_i]\!]_s$.

Edge Count Estimation Finally, we compute $m = \sum_{s=1}^{c}[\![m]\!]_s$ and reveal the value of m.

3.2 Protocol for Counting Wedges

Basic Idea. We find that the number of wedges in graph G can be computed as: $\wedge = \sum_{i=1}^{n} \frac{d_i(d_i-1)}{2}$. Based on the above observation, we let each user $i \in V$ compute $\frac{d_i(d_i-1)}{2}$ and secretly share the value to the data-collection servers and then compute $\wedge$ based on aggregating secret shares of all $\frac{d_i(d_i-1)}{2}$. Next, we introduce our protocol MPC-CW for implementing the above idea. The pseudo-code of our MPC-CW is given in Algorithm 3.

Algorithm 2 Protocol MPC-CE for counting all edges in graph G.

```
Data collection phase:
foreach user $i \in \{1, \dots, n\}$ do
    $d_i^* \leftarrow |\{j : j \in N_i, j > i\}|$;
    $([\![d_i^*]\!]_1, \dots, [\![d_i^*]\!]_c) \leftarrow$ SecShare($d_i^*, c$);
    send $[\![d_i^*]\!]_1, \dots, [\![d_i^*]\!]_c$ to servers 1, ..., $c$, respectively;
Data aggregation phase:
foreach server $s = 1, \dots, c$ do  $[\![m]\!]_s \leftarrow \sum_{i=1}^{n}[\![d_i^*]\!]_s$ ;
Edge count estimation phase: $m \leftarrow \sum_{s=1}^{c}[\![m]\!]_s$.
```

Algorithm 3 Protocol MPC-CW for counting all wedges in graph G.

```
Data collection phase:
foreach user $i = 1, \dots, n$ do
    $\wedge_i = 0.5 \times d_i(d_i - 1)$;
    $([\![\wedge_i]\!]_1, \dots, [\![\wedge_i]\!]_c) \leftarrow$ SecShare($\wedge_i, c$);
    send $[\![\wedge_i]\!]_1, \dots, [\![\wedge_i]\!]_c$ to servers 1, ..., $c$, respectively;
Data aggregation phase:
foreach server $s = 1, \dots, c$ do  $[\![\wedge]\!]_s \leftarrow \sum_{i=1}^{n}[\![\wedge_i]\!]_s$ ;
Wedge count estimation phase: $\wedge \leftarrow \sum_{s=1}^{c}[\![\wedge]\!]_s$.
```

Data Collection. Similar to the protocol of counting edges, each user i securely shares a variable $\wedge_i = \frac{d_i(d_i-1)}{2}$ to all the servers.

Data Aggregation. Each server $s \in \{1, \dots, c\}$ computes its secret share of the number of wedges $\wedge_s$ as $[\![\wedge]\!]_s = \sum_{i=1}^{n}[\![\wedge_i]\!]_s$.

Wedge Count Estimation. Finally, we compute $\wedge = \sum_{s=1}^{c}[\![\wedge]\!]_s$ and reveal the value of $\wedge$.

3.3 Protocols for Counting Triangles

Basic Protocol. We first introduce a straightforward protocol MPC-CT to implement the above idea. The pseudo-code of our MPC-CT is given in Algorithm 4.

• **Data Collection.** Each user $k \in V$ uses a vector $\mathbf{W}_k$ of size $\frac{(n-k)(n-k-1)}{2}$ to record the wedges it observed, where n denotes the total number of users. Each element of $\mathbf{W}_k$ is initialized to 0. User k enumerates each pair of its neighbors $i, j \in N_k$ such that $i < j$ and then sets

$$W_k = \begin{cases} W_k[j-i] = 1, & i = k+1 \\ W_k[\frac{1}{2}(i-2)(2n-i-1) + j - i] = 1, & i > k+1 \end{cases}$$

Similarly, user k uses a vector $\mathbf{A}_k$ of size $n-k$ to record the edges it observed. Each element of $\mathbf{A}_k$ is initialized to 0. User k enumerates each of its neighbors $i \in N_k$ and then sets

$$A_k[i-k] = 1.$$

Algorithm 4 Protocol MPC-CT for counting all triangles in graph G.

```
Data collection phase:
foreach user $k = 1, \dots, n$ do
    $A_k \leftarrow (A_k[1], \dots, A_k[n-k]) \leftarrow (0, \dots, 0)$;
    $W_k \leftarrow (W_k[1], \dots, W_k[\frac{1}{2}(n-k)(n-k-1)]) \leftarrow (0, \dots, 0)$;
    foreach $i \in N_k$ do
        $A_k[i-k] \leftarrow 1$;
        foreach $j \in N_k$ do
            if $i < j$ & $i = k+1$ then
                $w_k[(i-1)n + j] \leftarrow 1$;
            if $i < j$ & $i > k+1$ then
                $W_k[\frac{1}{2}(i-2)(2n-i-1) + j - i] \leftarrow 1$;
    $([\![A_k]\!]_1, \dots, [\![A_k]\!]_c) \leftarrow$ SecShare$(A_k, c)$;
    $([\![W_k]\!]_1, \dots, [\![W_k]\!]_c) \leftarrow$ SecShare$(W_k, c)$;
    send $([\![A_k]\!]_1, [\![W_k]\!]_1), \dots, ([\![A_k]\!]_c, [\![W_k]\!]_c)$ to servers 1, ..., $c$, respectively;

Data aggregation phase:
foreach server $s \in \{1, \dots, c\}$ do
    $[\![A]\!]_s \leftarrow$ Concat$([\![A_2]\!]_s, \dots, [\![A_n]\!]_s)$;
    $[\![W]\!]_s \leftarrow \sum_{k=1}^{n} [\![W_k]\!]_s$;

Triangle count estimation phase:
$([\![\Delta]\!]_1, \dots, [\![\Delta]\!]_c) \leftarrow$ Product$([\![A]\!], [\![W]\!])$;
/* Product performed collaboratively by all servers
   computes the product of vector $[\![A]\!]$ and $[\![W]\!]$, and
   each server will generate a portion of the result
   $[\![\Delta]\!]_c$ stored in a secret share form. */
Output $\Delta \leftarrow \sum_{s=1}^{c} [\![\Delta]\!]_s$.
```

After computing the $\mathbf{A}_k$ and $\mathbf{W}_k$, similar to the protocol of counting edges, each user k securely shares the vectors $\mathbf{A}_k$ and $\mathbf{W}_k$ to all the servers.

• **Data Aggregation.** For each A and W, each server $s \in \{1, \dots, c\}$ computes their secret shares as:

$$[\![A]\!]_s = \texttt{Concat}([\![A_1]\!]_s, \dots, [\![A_n]\!]_s), \quad [\![W]\!]_s = \sum_{k=1}^{n} [\![W_k]\!]_s,$$

where `Concat` represents the concatenation of the given vectors.

• **Triangle Count Estimation.** Denote

$$[\![A]\!] = [\![A]\!]_1 + \dots + [\![A]\!]_c, \quad [\![W]\!] = [\![W]\!]_1 + \dots + [\![W]\!]_c.$$

Each server s holds shares $[\![A]\!]_s$ and $[\![W]\!]_s$ of the vectors $[\![A]\!]$ and $[\![W]\!]$. All the servers can collaboratively compute the $[\![A]\!] \times [\![W]\!]$ and the result is also stored in the secret share form. Specifically, each server computes a secret share $[\![\Delta]\!]_s$ of the final result Δ. Thus, the secret shares of $[\![A]\!] \times [\![W]\!]$ are computed as:

$$([\![\Delta]\!]_1, \dots, [\![\Delta]\!]_c) = \texttt{Product}([\![A]\!], [\![W]\!]),$$

where `Product` [16] is a function computing the product of two variables stored in a secret share from. Finally, we reveal the value of Δ as:

$$\Delta = \sum_{s=1}^{c} [\![\Delta]\!]_s.$$

Let A denote the adjacency matrix of graph $G = (V, E)$. The (i, j) entry A_{ij} of A equals 1 when existing an edge $(i, j) \in E$ and 0 otherwise. In addition, we denote W as the wedge count matrix, where each element W_{ij} records how many neighbors nodes i and j have in common. Formally, we define W_{ij} as:

$$W_{ij} = |\{(i, k, j) : (i, k) \in E, (k, j) \in E, i, j, k \in V\}|.$$

Because each triangle is reported three times by its three nodes, then, we easily find

$$\Delta = \frac{1}{3} \sum_{1 \le i < j \le n} A_{ij} \times W_{ij}. \tag{1}$$

Improved protocol. The MPC-CT protocol only involves three servers in the calculation, and each server needs to perform multiplications on the 4 received shared variables. It's inefficient on either communication overhead or computation complexity. Next, we introduce a protocol MPC-CT++ which extends MPC-CT to the case that three servers participate in our protocol, and meanwhile cut down on unnecessarily expensive operations. The pseudo-code of our MPC-CT++ is given in Algorithm 5.

• **Data Collection.** Our MPC-CT++ uses the 2-out-of-3 replicated secret sharing from Araki *et al.* [17]. Specifically, to secretly share a number $x \in \mathbb{F}_p$ across three servers, we first split x into three shares. That is, generate two random variables $[\![x]\!]_1, [\![x]\!]_2 \in \mathbb{F}_p$ according to the uniform distribution and then compute $[\![x]\!]_3 = x - [\![x]\!]_1 - [\![x]\!]_2 \bmod p$ in field $\mathbb{F}_p$. Finally, we send $([\![x]\!]_1, [\![x]\!]_2)$ to server 1, $([\![x]\!]_2, [\![x]\!]_3)$ to server 2, and $([\![x]\!]_1, [\![x]\!]_3)$ to server 3.

After each user k computing $\mathbf{A}_k$, using the above protocol, we secretly share each element $A_k[l]$, $l = 1, \dots, n-k$ as the following procedure. Send $([\![A_k]\!]_1, [\![A_k]\!]_2)$ to server 1, $([\![A_k]\!]_1, [\![A_k]\!]_3)$ to server 2, and $([\![A_k]\!]_2, [\![A_k]\!]_3)$ to server 3.

After each user k computing $\mathbf{W}_k$, we use another different protocol to secretly share each element $W_k[l]$, $l = 1, \dots, \frac{1}{2}(n-k)(n-k-1)$ among the three servers. Specifically, we generate three independently generated secret share tuples, i.e., $([\![W_k]\!]_1^{(1)}, [\![W_k]\!]_1^{(2)})$, $([\![W_k]\!]_2^{(1)}, [\![W_k]\!]_1^{(3)})$, $([\![W_k]\!]_2^{(2)}, [\![W_k]\!]_2^{(3)})$, where the following equations hold:

$$W_k = [\![W_k]\!]_1^{(1)} + [\![W_k]\!]_1^{(2)} = \dots = [\![W_k]\!]_2^{(2)} + [\![W_k]\!]_2^{(3)}.$$

Then, we arrange all six elements in three tuples and send $([\![W_k]\!]_1^{(1)}, [\![W_k]\!]_2^{(1)})$ to server 1, send $([\![W_k]\!]_1^{(2)}, [\![W_k]\!]_2^{(2)})$ to server 2, and send $([\![W_k]\!]_1^{(3)}, [\![W_k]\!]_2^{(3)})$ to server 3.

Therefore, we list the vector tuples each server possesses as below:

$$\text{server 1} : ([\![A_k]\!]_1, [\![A_k]\!]_2, [\![W_k]\!]_1^{(1)}, [\![W_k]\!]_2^{(1)}), k = 1, \dots, n;$$

$$\text{server 2} : ([\![A_k]\!]_1, [\![A_k]\!]_3, [\![W_k]\!]_2^{(1)}, [\![W_k]\!]_1^{(3)}), k = 1, \dots, n;$$

$$\text{server 3} : ([\![A_k]\!]_2, [\![A_k]\!]_3, [\![W_k]\!]_2^{(2)}, [\![W_k]\!]_2^{(3)}), k = 1, \dots, n.$$

• **Data Aggregation.** For adjacent vectors, after receiving the secret share tuples of $[\![A]\!]$, each sever concatenates them to form aggregated vectors. In particular, server 1 obtains:

$$[\![A]\!]_1^{(1)} = \texttt{Concat}([\![A_1]\!]_1, \dots, [\![A_n]\!]_1),$$

$$[\![A]\!]_2^{(1)} = \texttt{Concat}([\![A_1]\!]_2, \dots, [\![A_n]\!]_2);$$

server 2 obtains:

$$[\![A]\!]_1^{(2)} = \texttt{Concat}([\![A_1]\!]_1, \ldots, [\![A_n]\!]_1),$$
$$[\![A]\!]_2^{(2)} = \texttt{Concat}([\![A_1]\!]_3, \ldots, [\![A_n]\!]_3);$$

server 3 obtains:

$$[\![A]\!]_1^{(3)} = \texttt{Concat}([\![A_1]\!]_2, \ldots, [\![A_n]\!]_2),$$
$$[\![A]\!]_2^{(3)} = \texttt{Concat}([\![A_1]\!]_3, \ldots, [\![A_n]\!]_3).$$

Each aggregated vector indicates a secret share of the complete adjacent vector $[\![A]\!]$, and its size is L in total.

Similarly, for wedge vectors, each server collects the secrete share tuples of $[\![W]\!]$ and performs element-wise addition to achieve aggregated vectors. In particular, server 1 obtains:

$$[\![W]\!]_1^{(1)} = \sum_{k=1}^{n} [\![W_k]\!]_1^{(1)}, \quad [\![W]\!]_2^{(1)} = \sum_{k=1}^{n} [\![W_k]\!]_2^{(1)};$$

server 2 obtains:

$$[\![W]\!]_1^{(2)} = \sum_{k=1}^{n} [\![W_k]\!]_1^{(2)}, \quad [\![W]\!]_2^{(2)} = \sum_{k=1}^{n} [\![W_k]\!]_2^{(2)};$$

server 3 obtains:

$$[\![W]\!]_1^{(3)} = \sum_{k=1}^{n} [\![W_k]\!]_1^{(3)}, \quad [\![W]\!]_2^{(3)} = \sum_{k=1}^{n} [\![W_k]\!]_2^{(3)}.$$

The total length of the aggregated vectors is also L.

• **Triangle Count Estimation.** Given secret share vectors $[\![A]\!]_1^{(1)}$, $[\![A]\!]_2^{(1)}$, $[\![W]\!]_1^{(1)}$, and $[\![W]\!]_2^{(1)}$ holding by server 1, server 1 computes:

$$[\![\Delta]\!]_1 \leftarrow \sum_{l=1}^{L} [\![A[l]]\!]_1^{(1)} \times [\![W[l]]\!]_1^{(1)} + [\![A[l]]\!]_2^{(1)} \times [\![W[l]]\!]_2^{(1)}.$$

Similarly, server 2 computes:

$$[\![\Delta]\!]_2 \leftarrow \sum_{l=1}^{L} [\![A[l]]\!]_1^{(2)} \times [\![W[l]]\!]_1^{(2)} + [\![A[l]]\!]_2^{(2)} \times [\![W[l]]\!]_2^{(2)},$$

and server 3 computes:

$$[\![\Delta]\!]_3 \leftarrow \sum_{l=1}^{L} [\![A[l]]\!]_1^{(3)} \times [\![W[l]]\!]_1^{(3)} + [\![A[l]]\!]_2^{(3)} \times [\![W[l]]\!]_2^{(3)}.$$

Finally, we obtain the value of $\Delta = ([\![\Delta]\!]_1 + [\![\Delta]\!]_2 + [\![\Delta]\!]_3)$.

3.4 Distributed Additive Noise Mechanisms for Protecting Count Results

Before revealing the values of m, $\wedge$, and Δ, we can protect these variables against differential attacks by using additive noise mechanisms for achieving differential privacy. Dwork et al. [18] give the formal definition of DP as:

Differential Privacy (DP). Let $\mathcal{D}$ be a collection of all datasets. A randomized algorithm $M : \mathcal{D} \to \mathcal{Y}$ satisfies (ε, δ)-differential privacy if and only if for all pairs of $x, x' \in \mathcal{D}$ differ in at most one element, for all events $E \subseteq \mathcal{Y}$, the following inequality holds:

$$P(M(x) \in E) \le \exp(\varepsilon) P(M(x') \in E) + \delta.$$

Algorithm 5 Protocol MPC-CT++ for counting all triangles in graph G.

```
Data collection phase:
foreach user $k = 1, \ldots, n$ do
    $A_k \leftarrow (A_k[1], \ldots, A_k[n-k]) \leftarrow (0, \ldots, 0)$;
    $W_k \leftarrow (W_k[1], \ldots, W_k[\frac{1}{2}(n-k)(n-k-1)]) \leftarrow (0, \ldots, 0)$;
    foreach $i \in N_k$ do
        $A_k[i-k] \leftarrow 1$;
        foreach $j \in N_k$ do
            if $i < j$ & $i = k+1$ then
                $w_k[(i-1)n + j] \leftarrow 1$;
            if $i < j$ & $i > k+1$ then
                $W_k[\frac{1}{2}(i-2)(2n-i-1) + j - i] \leftarrow 1$;
    $([\![A_k]\!]_1, [\![A_k]\!]_2, [\![A_k]\!]_3) \leftarrow \texttt{SecShare}(A_k, 3)$;
    send $([\![A_k]\!]_1, [\![A_k]\!]_2)$ to server 1;
    send $([\![A_k]\!]_1, [\![A_k]\!]_3)$ to server 2;
    send $([\![A_k]\!]_2, [\![A_k]\!]_3)$ to server 3;

    $([\![W_k]\!]_1^{(1)}, [\![W_k]\!]_1^{(2)}) \leftarrow \texttt{SecShare}(W_k, 2)$;
    $([\![W_k]\!]_2^{(1)}, [\![W_k]\!]_1^{(3)}) \leftarrow \texttt{SecShare}(W_k, 2)$;
    $([\![W_k]\!]_2^{(2)}, [\![W_k]\!]_2^{(3)}) \leftarrow \texttt{SecShare}(W_k, 2)$;
    send $([\![W_k]\!]_1^{(1)}, [\![W_k]\!]_2^{(1)})$ to server 1;
    send $([\![W_k]\!]_1^{(2)}, [\![W_k]\!]_2^{(2)})$ to server 2;
    send $([\![W_k]\!]_1^{(3)}, [\![W_k]\!]_2^{(3)})$ to server 3;

Data aggregation phase:
server 1 computes:
$[\![A]\!]_1^{(1)} \leftarrow \texttt{Concat}([\![A_2]\!]_1, \ldots, [\![A_n]\!]_1)$;
$[\![A]\!]_2^{(1)} \leftarrow \texttt{Concat}([\![A_2]\!]_2, \ldots, [\![A_n]\!]_2)$;
$([\![W]\!]_1^{(1)}, [\![W]\!]_2^{(1)}) \leftarrow (\sum_{k=1}^{n} [\![W_k]\!]_1^{(1)}, \sum_{k=1}^{n} [\![W_k]\!]_2^{(1)})$;
server 2 computes:
$[\![A]\!]_1^{(2)} \leftarrow \texttt{Concat}([\![A_2]\!]_1, \ldots, [\![A_n]\!]_1)$;
$[\![A]\!]_2^{(2)} \leftarrow \texttt{Concat}([\![A_2]\!]_3, \ldots, [\![A_n]\!]_3)$;
$([\![W]\!]_1^{(2)}, [\![W]\!]_2^{(2)}) \leftarrow (\sum_{k=1}^{n} [\![W_k]\!]_1^{(2)}, \sum_{k=1}^{n} [\![W_k]\!]_2^{(2)})$;
server 3 computes:
$[\![A]\!]_1^{(3)} \leftarrow \texttt{Concat}([\![A_2]\!]_2, \ldots, [\![A_n]\!]_2)$;
$[\![A]\!]_2^{(3)} \leftarrow \texttt{Concat}([\![A_2]\!]_3, \ldots, [\![A_n]\!]_3)$;
$([\![W]\!]_1^{(3)}, [\![W]\!]_2^{(3)}) \leftarrow (\sum_{k=1}^{n} [\![W_k]\!]_1^{(3)}, \sum_{k=1}^{n} [\![W_k]\!]_2^{(3)})$;

Triangle count estimation phase:
sever 1 computes:
$[\![\Delta]\!]_1 \leftarrow \sum_{l=1}^{L} [\![A[l]]\!]_1^{(1)} \times [\![W[l]]\!]_1^{(1)} + [\![A[l]]\!]_2^{(1)} \times [\![W[l]]\!]_2^{(1)}$;
sever 2 computes:
$[\![\Delta]\!]_2 \leftarrow \sum_{l=1}^{L} [\![A[l]]\!]_1^{(2)} \times [\![W[l]]\!]_1^{(2)} + [\![A[l]]\!]_2^{(2)} \times [\![W[l]]\!]_2^{(2)}$;
sever 3 computes:
$[\![\Delta]\!]_3 \leftarrow \sum_{l=1}^{L} [\![A[l]]\!]_1^{(3)} \times [\![W[l]]\!]_1^{(3)} + [\![A[l]]\!]_2^{(3)} \times [\![W[l]]\!]_2^{(3)}$;
Output $\Delta \leftarrow ([\![\Delta]\!]_1 + [\![\Delta]\!]_2 + [\![\Delta]\!]_3)$.
```

Additive Noise Mechanisms (ANMs). ANMs [18], [19] are differential private mechanisms that add controlled noise from predetermined distributions. ANMs are useful for designing private mechanisms for real-valued functions on sensitive data. Formally, let $f : \mathcal{D} \to \mathbb{R}$ be a real-value function with sensitivity Δ_f defined as

$$\Delta_f = \max |f(x) - f(y)|, \tag{2}$$

where the maximum is over all pairs of datasets x and y

in $\mathcal{D}$ differing in at most one element. ANMs protect $f(x)$ against differential attacks by adding a noise variable r, i.e.,

$$\mathcal{M}(x, f) = f(x) + r.$$

Gaussian and Laplacian distributions are the commonly used distributions for adding noise. When $r \sim \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{2\ln(1.25/\delta)(\Delta_f)^2}{\varepsilon^2}\right)$, where parameter μ is the expectation of the Gaussian distribution and parameter σ is its standard deviation, function $\mathcal{M}(x, f)$ achieves (ε, δ)-differential privacy. When $r \sim \text{Lap}\left(\mu = 0, b = \frac{\Delta_f}{\varepsilon}\right)$ where parameter μ is the expectation of the Laplace distribution and parameter b is the scale parameter, function $\mathcal{M}(x, f)$ achieves $(\varepsilon, 0)$-differential privacy.

In the setting of our problem, each server is semi-honest and we cannot rely on a single server to generate the additive differential noise variable r. To solve this issue, we use the following distributed additive noise mechanisms.

Distributed Additive Gaussian Noise Mechanism. Each data-collection server $s \in \{1, \ldots, c\}$ generates a noise variable $r_s \sim \mathcal{N}(0, \sigma^2/c)$. Then, the sum of all noise variables generated by the servers is also a Gaussian variable according to $\mathcal{N}(0, \sigma^2)$, i.e.,

$$\sum_{s=1}^{c} r_s \sim \mathcal{N}(0, \sigma^2).$$

To protect against differential attacks, therefore, each server s generates r_s and adds r_s to the secret share $[\![x]\!]_s$ of variable x holding by itself before revealing the value of x.

Distributed Additive Laplacian Noise Mechanism. Similar to the above distributed additive Gaussian noise mechanism, we let each server s generate two independent variables $X_s, Y_s \sim \Gamma(k = 1/c, \theta = b)$, where k and θ are two parameters of the Gamma distribution. Then, we have

$$\sum_{s=1}^{c} (X_s - Y_s) \sim \text{Lap}(0, b).$$

To protect a variable x against differential attacks, thus, each server s generates X_s and Y_s and then adds $X_s - Y_s$ to the secret share $[\![x]\!]_s$ of variable x holding by itself before revealing the value of x.

Sensitivities of Edge, Wedge, and Triangle Counts. We easily find that one edge insertion (resp. deletion) increases (resp. decreases) the value of m (i.e., the number of edges) by 1. Therefore, the sensitivity of m defined in Eq. (2) is $\Delta_m = 1$. As we mentioned, graph G consists of n nodes. Given an edge (u, v) such that $u, v \in \{1, \ldots, n\}$ to be removed from or inserted into graph G, there exist at most $n - 2$ wedges including this edge (u, v). Therefore, the sensitivity of $\wedge$ (i.e., the number of wedges) is $\Delta_\wedge = n - 2$. Similarly, we find that the sensitivity of Δ (i.e., the number of triangles) is $\Delta_\Delta = n - 2$.

3.5 Utility, Overhead, and Complexity

Utility Analysis. The proposed method offers a significant benefit in the accurate computation of $[\![m]\!]_s$, $[\![\wedge]\!]_s$, and $[\![\Delta]\!]_s$ by each server s using the received secret shares. The source of error in this process is the additive noise introduced when a server s reveals its calculations. As a result, when aggregating these computations, the protocols deliver high data utility with the outcome $\hat{f}(G) = f(G) + r$, where the estimation error stems exclusively from the noise term r added to the ground truth subgraph count $f(G)$. By assessing the magnitude of r, one can establish an expected utility value.

It warrants emphasis that the MPC-CT and MPC-CT++ protocols yield identical error estimations. Therefore, in the forthcoming experimental section concerning error comparison, they are collectively treated as a single MPC-based protocol due to their common estimation error. The evaluation focuses on the MPC-based method in conjunction with the Gaussian noise mechanism and the Laplacian noise mechanism (i.e., *MPC-G* and *MPC-L*). Conversely, when scrutinizing overhead comparisons, MPC-CT and MPC-CT++ manifest disparities that necessitate discrete consideration, thus facilitating a more comprehensive comparison with other protocols.

To begin this process, we adopt *Mean Relative Error* (MRE):

$$\text{MRE} \triangleq \frac{|\hat{f}(G) - f(G)|}{f(G)} = \frac{|r|}{f(G)},$$

as our utility metric to exhibit our protocols' accuracy. A smaller value of MRE represents the better utility of the algorithm. For (ε, δ)-differential privacy, r is subject to the Gaussian distribution $\mathcal{N}(\mu, \sigma)$, and therefore we can compute the expectation of MRE by solving the following integral:

$$\text{MRE}_{\text{G}}^{\text{E}} = \frac{\mathbb{E}(|r|)}{f(G)} = \frac{\int_{-\infty}^{\infty} |r|\, \Phi(r) dr}{f(G)},$$

where $\Phi(r)$ is the probability density function (PDF) of the Gaussian distribution, i.e.,

$$\Phi(r) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r-\mu)^2}{2\sigma^2}},$$

where $\mu = 0$ and $\sigma = \frac{\sqrt{2\ln(1.25/\delta)} \cdot \Delta_f}{\varepsilon}$. Then, we substitute $\Phi(r)$ to the integral and obtain:

$$\text{MRE}_{\text{G}}^{\text{E}} = \frac{2\sqrt{\ln\left(\frac{1.25}{\delta}\right)}}{\sqrt{\pi}\varepsilon} \cdot \frac{\Delta_f}{f(G)}.$$

The first term of the above formula is decided by specific differential privacy settings of (ε, δ), and the properties of graph data determine the second term. Similar derivation as above, when $r \sim \text{Lap}(\mu, b)$, the PDF of the Laplacian distribution is written as $\Psi(r) = \frac{1}{2b} e^{-\frac{|r-\mu|}{b}}$, where $\mu = 0$ and $b = \frac{\Delta_f}{\varepsilon}$. Afterwards we substitute the formula of $\Psi(r)$ into the following integral function of $\text{MRE}_{\text{L}}^{\text{E}}$ and acquire the concluding expression:

$$\text{MRE}_{\text{L}}^{\text{E}} = \frac{\int_{-\infty}^{\infty} |r|\, \Psi(r) dr}{f(G)} = \frac{1}{\varepsilon} \cdot \frac{\Delta_f}{f(G)}.$$

It's apparent from the above equation that the $\text{MRE}_{\text{L}}^{\text{E}}$ is also mainly influenced by DP settings and graph properties.

Overhead and Complexity. When analyzing memory consumption and complexity, we find that the MPC-CE and MPC-CW protocols exhibit minimal memory requirements. In these protocols, each server is responsible for managing a single integer number from n users, leading to both

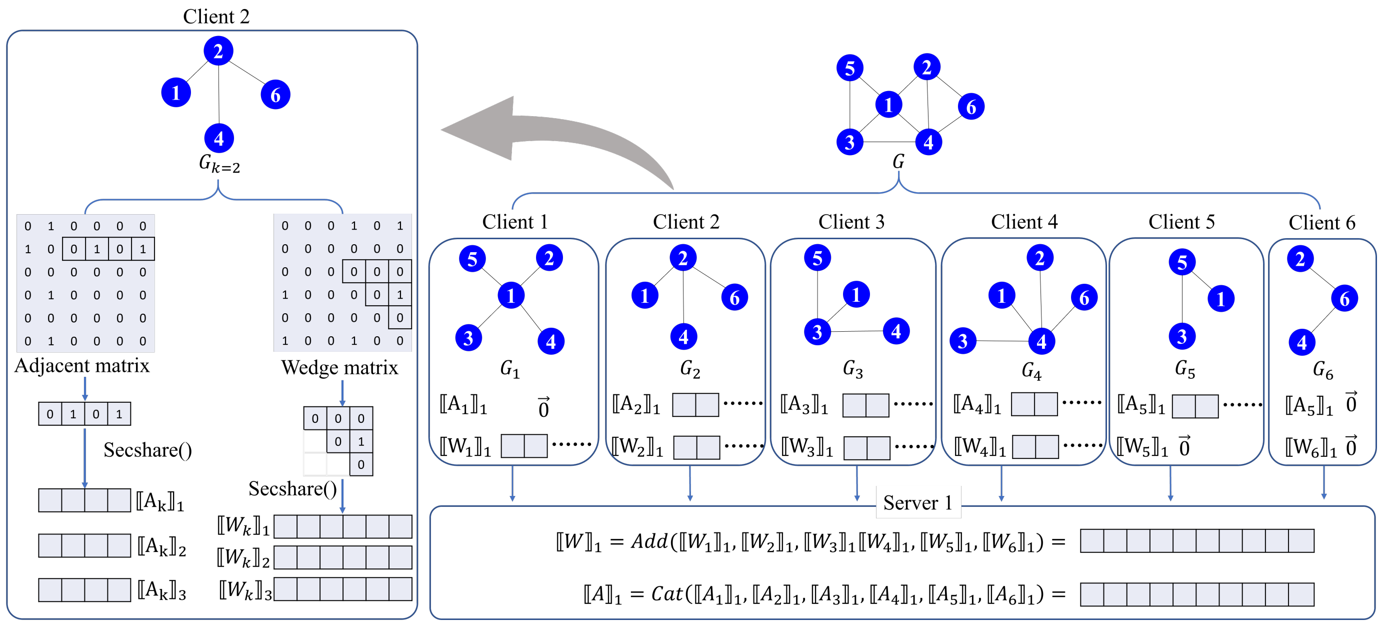


Fig. 1: In the fully distributed graph G with 6 nodes, each node represents a client k possessing a local subgraph G_k. The illustration depicts server 1 collecting $[\![A_k]\!]_1$ and $[\![W_k]\!]_1$ from clients, with other servers acting similarly. Server 1 then aggregates the $[\![W_k]\!]_1$ values to obtain $[\![W]\!]_1$, and concatenates the $[\![A_k]\!]_1$ values to derive $[\![A]\!]_1$.

the transfer of n data and the performance of n integer additions for each server. In contrast, our proposed MPC-CT and MPC-CT++ protocols primarily involve matrix and vector operations, which are $\frac{1}{2}(n-1)(n-2)$ times larger than a single integer. Thus, the following paragraphs will concentrate on the overhead and complexity of the MPC-CT and MPC-CT++ protocols.

The transmission of adjacency matrices and wedge matrices is a requirement for both the MPC-CT and MPC-CT++ protocols. In a single round of transmission, the entire adjacency matrices and wedge matrices create a $2n^3$ total communication overhead. This overhead occurs because each of the n clients sends two matrices with a size of n^2. Due to the random decomposition of the original $\mathbf{A}_k$ and $\mathbf{W}_k$ into secret-shared matrices, the sparse matrix method is not suitable for these protocols.

To reduce the overhead, our protocols decrease the communication overhead for a single $[\![A]\!]_k$ and $[\![W]\!]_k$ to $n-k$ and $\frac{1}{2}(n-k)(n-k-1)$, respectively. This reduction results in an overhead of $\frac{1}{2}n$ the size of the original $[\![A]\!]_k$ and $\frac{1}{6}$ the size of the original $[\![W]\!]_k$. The fundamental approach involves transferring only the upper triangle of the matrix instead of the entire matrix. As shown in Fig. 1, for the adjacency matrix, each client only uploads the k-th row of the upper triangle. For the wedge matrix, each client extracts upper triangular elements below the k-th row, where k represents the client's number. The server-side operation involves concatenating and re-organizing the collected adjacency vectors and wedge matrix elements to produce aggregated adjacency vector $[\![A]\!]_i$ and $[\![W]\!]_i$, both with sizes of $\frac{1}{2}(n-2)(n-1)$. An overview of the aggregation operation is depicted in Fig. 1.

The total overhead for transferring $\mathbf{A}$ and $\mathbf{W}$ amounts to $\frac{1}{2}(n-2)(n-1)$ and $\frac{1}{6}n(n-1)(n-2)$, respectively. This overhead is acceptable for moderately-sized graphs. For instance, our experimental results show that when the distributed graph contains 10,000 nodes, a server can collect all required $\mathbf{A}$ and $\mathbf{W}$ from clients within 5.98 minutes.

The MPC-CT protocol involves n participating clients and c servers and consists of an offline phase. In this phase, each server stores a secret share of a pre-generated Beaver multiplication triple, created using somewhat homomorphic encryption techniques. During the online phase, each client k generates c secret share groups for servers, and each server collaborates with the other $c-1$ servers. Both the offline and online phases necessitate numerous temporary variables for communication, the sizes of which remain undetermined. As a result, we present empirical results $\mathcal{O}(n^3)$ obtained through sufficient experimental iterations instead of analytical results.

The MPC-CT++ protocol exhibits increased efficiency in both communication and complexity compared to the MPC-CT protocol. It requires only three participating servers and fewer multiplications to be computed. Each client sends a group of secret vectors to a server, reducing the per-server communication to $\frac{1}{3}(n-1)(n-2)(n+3)$. This efficiency level is approximately 1000 times greater than that of the MPC-CT protocol, as it eliminates the need for temporary variables in interactive offline and online computing.

4 Experiments

4.1 Experimental Setup

Dataset. In our experiments, we evaluate the performance of our MPC-based protocol and compare it with state-of-the-art LDP-based and DP-based methods using synthetic datasets and twelve different real-world graphs. These graphs, such as Facebook, Twitter and Youtube, predominantly consist of small and medium-sized social networks with fewer than 4,000,000 nodes. We procured these datasets

TABLE 2: The statistics of the experimental datasets.

Dataset	#nodes	#edges	#wedges	#triangles
Email	1,005	16,706	1,183,216	105,461
FaceBook	4,039	88,234	9,314,849	1,612,010
Wiki-V	7,115	100,762	14,545,580	608,389
CondMat	23,133	93,497	1,967,650	173,361
SlashDot	77,360	469,180	68,516,301	551,724
Twitter	81,306	2,420,766	230,093,274	9,355,656
Stanford	281,903	1,992,636	3,944,069,093	11,329,473
Amazon	403,394	2,443,408	72,209,687	3,986,507
Google	875,713	4,322,051	727,417,224	13,391,903
YouTube	1,134,890	2,987,624	1,474,482,560	3,056,386
Wiki-T	2,394,385	4,659,565	12,593,522,422	9,203,519
Patents	3,774,768	16,518,948	335,781,273	7,515,023

from the *Stanford Network Analysis Project (SNAP)* [20]. Table 2 furnishes detailed information about the datasets.

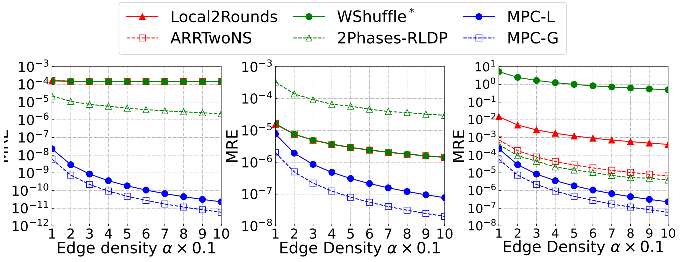
Baselines. We assess our MPC-based protocols in the context of various representative edge, wedge, and triangle counting approaches, comprising:

1) *Local2Rounds*: A 2-round communication protocol under the LDP settings, as proposed by Imola et al. [4]. In this protocol, the client incorporates noise into its local subgraph and uploads it to the server, which subsequently returns the noisy graph to each client. This process enables clients to make more precise predictions as their estimation of local connections remains unbiased. Although the original protocol does not encompass an edge-counting task, we modified the provided code to accommodate our three counting tasks, denoted as *local2Rounds*m, *local2Rounds*$\wedge$, and *local2Rounds*$_\triangle$.

2) *ARRTwoNS*: An enhanced protocol that curtails the communication overhead of Local2Rounds [1]. This approach employs edge sampling and double clipping techniques, substantially diminishing the sensitivity of each query and decreasing the communication time from 6 hours to 8 seconds. While the original code includes edge and wedge counting tasks, we adapted it to suit our tasks, resulting in *ARRTwoNS*m, *ARRTwoNS*$\wedge$, and *ARRTwoNS*$_\triangle$.

3) *WShuffle**: A one-round interaction protocol based on the shuffle model technique [21]. This method introduces wedge shuffle to augment privacy while counting triangle and 4-cycle subgraphs. It attains more precise clustering tendencies with a relatively small privacy budget (e.g., less than 1) in edge DP. We modified the code to facilitate the implementation of our tasks, yielding *WShuffle*$^*m$, *WShuffle*$^*\wedge$, and *WShuffle*$^*_\triangle$.

4) *2Phases-RLDP*: A protocol that introduces Relation Local Differential Privacy (RLDP) to ensure robust privacy protection [2]. The refined 2Phases-RLDP assesses each client's subgraph's global data correlation and necessitates clients to report their randomly sampled triangle numbers to the server. The server then computes an approximate global triangle count. The entire framework satisfies (ε, δ)-Edge-RLDP. We also modified the source code to fit our three tasks, resulting in *2Phases-RLDP*m, *2Phases-RLDP*$\wedge$, and *2Phases-RLDP*$_\triangle$.



(a) Edge Counting (b) Wedge Counting (c) Triangle Counting

Fig. 2: Relation between the MRE and edge density α in different protocols, with n fixed to 10,000 and $\varepsilon = 1$.

Parameter settings. In our experiments, we set $p = 2147483647$, which is the largest prime number within the int32 value range. This choice of parameters ensures consistency and comparability across all experiments.

4.2 Experimental Results on synthetic datasets

To conduct a thorough ablation study, we create various synthetic Erdős-Rényi graphs that exhibit a range of graph properties, such as the number of nodes n and edge density α, while also taking the privacy budget ε into account as a variable. By analyzing these parameters, we aim to demonstrate their impact on the accuracy and communication overhead of both our proposed protocols and existing baseline protocols. By default, we generate a baseline graph consisting of 1,0000 nodes and an edge density of 0.5, with the privacy budget ε set to 1.

MRE vs. edge density α. Edge density is defined as follows:

$$\alpha = \frac{2e - n + 1}{(n-1)(n-2)},$$

where e and n denote the number of edges and nodes, respectively. Unlike the raw edge counts e, the edge density metric α accounts for the number of edges and the graph size, enabling a fair comparison of graph densities. In connected undirected graphs, the edge count can range from $[n-1, \frac{n(n-1)}{2}]$. By normalizing the edge count with respect to the total possible edges, α intuitively measures graph density. To illustrate this point, consider two graphs, G_a with $n = 1,000$ and $e = 100,000$, and G_b with $n = 5,000$ and $e = 100,000$. Despite having the same number of edges, their edge densities differ $\alpha_a = 0.2$ for G_a and $\alpha_b = 0.0078$ for G_b. This demonstrates the higher density of G_a compared to G_b, highlighting the effectiveness of the α metric.

Fig. 2(a) describes the edge counting task results, accentuating the accuracy superiority in MRE across diverse edge densities. A salient observation is the nearly horizontal and overlapping trends demonstrated by the *Local2Rounds*, *ARRTwoNS*, *WShuffle** protocols, with an MRE of 1.6×10^{-4}. This finding suggests that these three protocols employ analogous edge counting techniques, exhibiting limited sensitivity to fluctuations in the parameter α. In contrast, the *2Phases-RLDP* protocol displays a modest decline in MRE, extending from 2.22×10^{-5} to 2.19×10^{-6}. The proposed MPC-based methods, *MPC-G*, and *MPC-L*, exhibit a more substantial reduction in MRE values. Specifically, *MPC-G* experiences a decrease in MRE from 2.31×10^{-8}

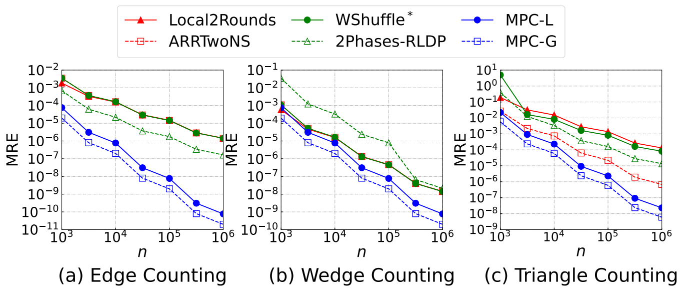


Fig. 3: Relation between the MRE and nodes n in different protocols, with α fixed to 0.5 and ε fixed to 1.

to 2.32×10^{-11}, while *MPC-L* observes a reduction from 5.97×10^{-9} to 6.00×10^{-12}.

Fig. 2(b) provides insights into the wedge counting task. The *2Phases-RLDP* protocol demonstrates higher MRE values than other protocols, which declines from 3.30×10^{-4} to 2.98×10^{-5} as α increases, suggesting it might be less efficient for sparse data. Nonetheless, it could offer advantages in specific scenarios, such as those with resource constraints. The *Local2Rounds*, *ARRTwoNS*, and *WShuffle** protocols exhibit overlapping trends and a similar decline speed in MRE values as the *2Phases-RLDP* protocol. Specifically, their MRE values decrease approximately from 1.56×10^{-5} to 1.41×10^{-6} as α increases. This behavior indicates that these protocols share similar wedge-counting approaches and can adapt to different edge densities. In contrast, *MPC-G* and *MPC-L* show a more significant MRE reduction as α increases, reducing from 7.70×10^{-6} to 7.73×10^{-8} and 1.99×10^{-6} to 2.00×10^{-8}, respectively. These results highlight the adaptability and scalability of the MPC-based methods in handling different edge densities.

Fig. 2(c) delineates the performance of diverse protocols with respect to the triangle counting task. Although all protocols exhibit decreasing MRE values as α increases, notable variations are observed in the decline rate and MRE value range among the methods. Evidently, our MPC-based protocols consistently display the lowest MRE values, thereby attaining the highest data utility. For instance, when $\varepsilon = 6$, the MRE values of the tested protocols are 9.15×10^{-4}, 1.92×10^{-5}, 0.83, 1.06×10^{-5}, 1.0734×10^{-6}, and 2.78×10^{-07}. The *MPC-L* protocol surpasses the existing best protocol by a factor of 38 in terms of accuracy.

MPC-G and *MPC-L* not only exhibit the most promising outcomes, with MRE values decreasing from 2.31×10^{-4} to 2.32×10^{-7} and from 5.97×10^{-5} to 6.00×10^{-8}, respectively, but also display a 994-fold decline in their MRE values as the graph becomes denser. Revisiting the analysis in Section 3.5, our MPC-based methods exhibit a decline rate akin to an inversely proportional function, while alternative strategies demonstrate decline rates more characteristic of a negative exponential function. The steeper decline rates observed in the MPC-based methods indicate their potential for tackling the triangle counting task in denser networks.

MRE vs. n. A salient observation in Fig. 3 is the consistent decline in MRE as the number of nodes increases, emphasizing that estimations related to graph data tend to exhibit improved accuracy when a more extensive node set is involved. Upon further examination of Fig. 3(a), it is evident that *Local2Rounds*, *ARRTwoNS*, and *WShuffle** display similar patterns of decline. This resemblance can be attributed to their shared edge-counting techniques. Initially, *Local2Rounds* possesses a higher MRE value of approximately 3.50×10^{-3} compared to *2Phases-RLDP* with an MRE value of 6.85×10^{-4}. As the number of nodes approaches the maximum value, the MRE for *Local2Rounds* decreases to 1.42×10^{-6}, closely aligning with the other two protocols. In comparison, MPC-based protocols exhibit a lower MRE value than both *Local2Rounds* and *2Phases-RLDP*. As the number of nodes increases, the disparity between the MRE values of *MPC-L* and the other protocols widens. At the maximum node count, the MRE values for *MPC-G* and *MPC-L* reach 7.73×10^{-11} and 2.00×10^{-11}, highlighting the accuracy achieved by the MPC-based methods.

Fig. 3(b) demonstrates that our MPC-based methods exhibit favorable utility compared to alternative protocols in the context of the wedge counting task. For example, with 1,000 nodes, the *MPC-G* method achieves an MRE value of 7.47×10^{-4}, showing a modest improvement over *Local2Rounds* (by a factor of approximately 1.3) and *ARRTwoNS* (by a factor of 1.15), given that *Local2Rounds* and *ARRTwoNS* have MRE values of 5.78×10^{-4} and 1.09×10^{-3}, respectively. In addition, *MPC-L* presents a further increase in accuracy, with an MRE value of 1.93×10^{-4}, which is around 3 times more accurate than *Local2Rounds* and 5.64 times more accurate than *ARRTwoNS*. When n is set to the maximum value of 10^6, all alternative protocols achieve a similar MRE value of 1.41×10^{-8}. At this point, our *MPC-L* protocol outperforms the alternatives by a factor of 70 in terms of accuracy.

Fig. 3(c) highlights critical differences in the performance of various methods for the triangle counting task, with *MPC-G* and *MPC-L* consistently achieving superior estimation accuracy compared to alternative protocols. Notably, *MPC-L* exceeds all other methods, delivering the lowest MRE values across different node counts. For example, at 5,000 nodes, *MPC-L* achieves an MRE value of 2.37×10^{-4}, while *MPC-G* reports an MRE value of 9.18×10^{-4}, which is 2.38 time and 9.20 time more accurate than the existing best approaches *ARRTwoNS*. As the number of nodes increases, the MRE values for all protocols generally decrease. However, the rate of improvement varies considerably among the different methods. The *MPC-G* and *MPC-L* methods exhibit the most pronounced reductions in MRE values as node count increases. The *Local2Rounds* protocol displays a slower reduction line as the node count increases, leading to a widening performance gap compared to the MPC-based techniques. *ARRTwoNS*, *2Phases-RLDP* and *WShuffle** follow a similar trajectory, with their MRE values decreasing at a comparable pace, yet lagging behind our protocols.

MRE vs. privacy budget ε. Fig. 4 displays the result of three counting tasks under different privacy levels, denoted by parameter ε ranging from 1 to 10. In edge counting task, at a higher privacy level ($\varepsilon = 1$), the Mean Relative Error (MRE) values for *MPC-G* and *MPC-L* stand at 1.12×10^{-3} and 2.51×10^{-4}. In contrast, *Local2Rounds* exhibits an MRE value of 1.28×10^{-2} and *ARRTwoNS* has an MRE value of 1.43×10^{-2}, both approximately one order of magnitude higher. As the privacy budget increases to $\varepsilon = 10$, the MRE values for *MPC-G* and *MPC-L* reach 1.55×10^{-8} and 4.00×10^{-9}, In comparison, *Local2Rounds* and *ARRTwoNS*

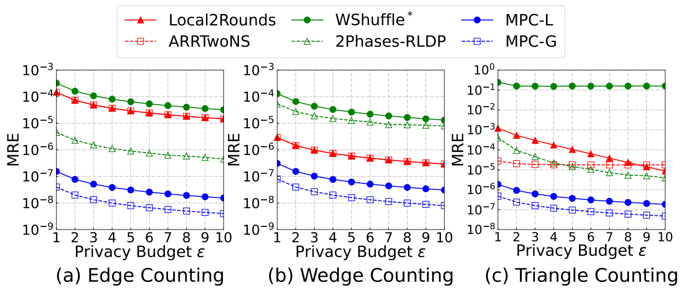


Fig. 4: Relation between the MRE and privacy budget ε of different protocols, with α fixed to 0.5, n fixed to 10,000

have MRE values of 1.47×10^{-5} and 1.46×10^{-5}, which are three orders of magnitude higher. *WShuffle** exhibits an MRE of 3.22×10^{-5}, approximately four orders of magnitude higher, while *2Phases-RLDP* performs better with an MRE of 4.51×10^{-7}, but still one to two orders of magnitude higher than our MPC-based methods.

The outcomes of the wedge counting task are presented in Fig. 4(b). At a high privacy level ($\varepsilon = 1$), the Mean Relative Error (MRE) values for *MPC-G* and *MPC-L* stand at 3.09×10^{-7} and 8.00×10^{-8}, respectively. In comparison, both *ARRTwoNS* and *Local2Rounds* have MRE values approximately one order of magnitude higher than the MPC-based methods, with 2.91×10^{-6} and 2.93×10^{-6}, respectively. Upon increasing the privacy budget to $\varepsilon = 10$, a general decrease in MRE values is observed for all protocols. However, *MPC-G* and *MPC-L*, maintain higher utility, with MRE values of 3.09×10^{-8} and 8.00×10^{-9}, respectively. In contrast, *2Phases-RLDP* remains two orders of magnitude higher than the MPC-based methods. Meanwhile, *WShuffle** has an MRE of 1.32×10^{-5}, approximately three orders of magnitude higher, and both *ARRTwoNS* and *Local2Rounds* still exhibit MRE values 10 times higher.

In Fig. 4(c), we assess the outcomes of the triangle counting task. At a low privacy budget ($\varepsilon = 1$), the MRE values for *MPC-G* and *MPC-L* are 1.85×10^{-6} and 4.80×10^{-7}, respectively, which is 12 times more accurate than the existing best protocol (*ARRTwoNS*). When the privacy level increases to $\varepsilon = 10$, the MRE values of *MPC-G* and *MPC-L* decrease rapidly to 1.85×10^{-7} and 4.80×10^{-8}, respectively, marking their scalability in maintaining high accuracy in more privacy-sensitive scenarios. In comparison, the MRE of *ARRTwoNS* and *WShuffle** remains almost constant with increasing privacy budget, implying a limited improvement in accuracy for loose privacy demands. *Local2Rounds* records an MRE of 8.71×10^{-6} at $\varepsilon = 10$, lower than before but still higher than the MPC-based approaches.

This evaluation emphasizes the relative advantages of *MPC-G* and *MPC-L* regarding estimation accuracy for different subgraph counting tasks, signifying that these approaches are optimally suited for privacy-sensitive situations involving this type of graph analysis. The trend analysis reveals that while some protocols show improvement in accuracy with a higher privacy budget, the MPC-based methods consistently outperform the others.

***n* vs. communication overhead and runtime**. In Fig. 5, we investigate the communication overhead and runtime of various edge and wedge protocols under different n settings, as depicted in the first row of Fig. 5. It is noteworthy

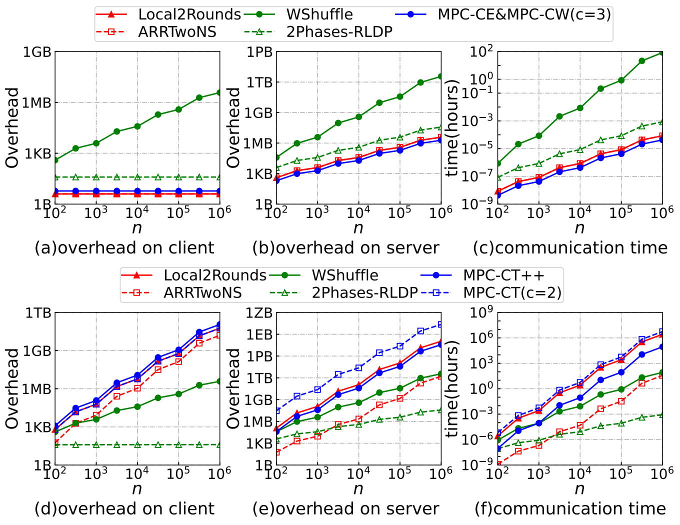


Fig. 5: Communication overhead and time on synthetic datasets including various n for edge counting, wedge counting (first row), and triangle counting (second row). The bandwidth limit is 1000Mbps between clients and servers, and 10Gbps for servers' interaction.

that our MPC-CE and MPC-CW protocols exhibit identical communication overheads, with each server receiving a single float (16 bits) number. As demonstrated in Fig. 5(a), the communication overhead per client is 48 bits when $c = 3$. In contrast, *WShuffle** incurs significantly higher overhead, as the protocol necessitates each client to download $\frac{n}{2}$ user pairs from the server during every counting procedure.

Fig. 5(b) and (c) present the communication overhead for a single server and the corresponding transmission time, given a bandwidth constraint of 1000 Mbps. For $n = 10^6$, the costs are as follows: 30.52 Mbits (0.03 s) for *Local2Rounds*; 30.52 Mbits (0.03 s) for *ARRTwoNS*; 29.10 Tbits (8.47 hours) for *WShuffle**; 305.18 Mbits (0.305 s) for *2Phases-RLDP*; and 15.25 Mbits (0.015 s) for MPC-based methods. As illustrated in Figs. 2 to 4, MPC-based methods outperform other protocols in terms of accuracy and efficiency for edge and wedge counting tasks. The second row of Fig. 5 reveals that MPC-based methods are communication-intensive due to the requirement to transmit secret share matrices, which have a size of $\mathcal{O}(n^2)$ and are stored in integer format (32 bits). Consequently, our protocols' overhead grows at the rate of the square (for the client side) and the cube (for the server side) as n increases. *Local2Rounds* follows a similar growth pattern, given its two-round communication of noisy graph data.

In particular, when $n = 10,000$, the MPC-CT++ protocol requires each server to collect 38.80 GB of data from all participating clients, resulting in each client transmitting 3.87 MB to the server. Assuming a bandwidth limit of 1000 Mbps, each server takes 5.29 minutes to receive all necessary secret shares. However, as n increases to $50,000$, the overhead grows to 4850.63 GB for a server, necessitating almost 11 hours for data transfer. Considering that our MPC-based protocols outperform other existing protocols by an average of 100 times in terms of utility, this level of efficiency is highly competitive for the triangle counting task.

TABLE 3: MRE result on real datasets when $\varepsilon = 1$. Each row represents a tested protocol, containing edge counting result (the first sub-row) and wedge counting result (the second sub-row). The lowest MRE on each dataset for different counting tasks is highlighted in bold.

dataset	Email	Facebook	Wiki-V	CondMat	Slashdot	Twitter	Stanford	Amazon	Google	Youtube	Wiki-T	Patents
local2	0.0147	0.0084	0.0075	0.0021	0.0038	0.001	0.0137	0.0008	0.001	0.0068	0.0152	3E-05
Rounds	3.2261	7.4506	6.5774	6.1035	26.21	14.183	200.84	66.614	51.58	597.36	1229.4	3.6386
ARR	0.0078	0.0016	0.0013	0.0012	0.0002	6E-05	6E-05	5E-05	3E-05	4E-05	2E-05	7E-06
TwoNS	0.8997	0.256	0.1844	1.9317	0.1067	0.0415	0.0036	0.2312	0.0329	0.0174	0.0029	0.1458
WShuffle	0.0078	0.0016	0.0013	0.0012	0.0002	6E-05	6E-05	5E-05	3E-05	4E-05	2E-05	7E-06
	0.8997	0.256	0.1844	1.9317	0.1067	0.0415	0.0036	0.2312	0.0329	0.0174	0.0029	0.1458
2Phases	0.0022	0.0008	0.0009	0.0018	0.0007	0.001	0.0003	0.0003	0.0002	0.0004	0.0004	0.0001
-RLDP	0.1668	0.0644	0.0512	0.4707	0.0667	0.0245	0.0111	0.152	0.0367	0.0459	0.0142	0.049
MPC-G	0.0002	4E-05	4E-05	4E-05	8E-06	2E-06	2E-06	2E-06	9E-07	1E-06	8E-07	2E-07
	0.0033	0.0017	0.0019	0.0454	0.0044	0.0014	0.0003	0.0216	0.0047	0.003	0.0007	0.0435
MPC-L	**6E-05**	**1E-05**	**1E-05**	**1E-05**	**2E-06**	**4E-07**	**5E-07**	**4E-07**	**2E-07**	**3E-07**	**2E-07**	**6E-08**
	0.0008	**0.0004**	**0.0005**	**0.0118**	**0.0011**	**0.0004**	**7E-05**	**0.0056**	**0.0012**	**0.0008**	**0.0002**	**0.0042**

4.3 Experimental Results on Real datasets

Edge and wedge counting tasks. In Table 3, we present the Mean Relative Error (MRE) results for edge and wedge counting tasks on twelve real-world datasets, as described in Section 4.1. We set the global value of ε to 1. Each row in the table represents a tested protocol, with the first sub-row illustrating the MRE for edge counting, and the second sub-row depicting the MRE for wedge counting.

Considering the Facebook dataset, the MRE values for edge counting include: *MPC-L* (1E-05), *MPC-G* (4E-05), *2Phases-RLDP* (0.0008), *ARRTwoNS* (0.0016), *WShuffle** (0.0016), and *Local2Rounds* (0.0084). For the same dataset, the MRE values for wedge counting are: *MPC-L* (0.0004), *MPC-G* (0.0017), *2Phases-RLDP* (0.0644), *ARRTwoNS* (0.256), *WShuffle** (0.256), and *Local2Rounds* (7.4506). A detailed examination of the Facebook dataset shows that the MPC-based protocols outperform other protocols on average by 271 times for edge counting and 4,630 times for wedge counting. When extending the comparison across all datasets, the average utility gap increases to 3,016 times for edge counting and 212,758 times for wedge counting. This evaluation highlights that the MPC-based protocols consistently perform better than other tested protocols for both edge and wedge counting tasks. In particular, the *MPC-L* protocol reaches the lowest MRE values, indicating the highest accuracy among the analyzed protocols.

Nonetheless, there are instances where other protocols exhibit competitive performance. For example, in the Patents dataset, the *2Phases-RLDP* protocol achieves an MRE value of 0.049 for wedge counting, which is relatively close to the performance of *MPC-G* (0.0145) and *MPC-L* (0.0042), about 3 times and 11 times less accurate, respectively. This observation suggests that the *2Phases-RLDP* protocol may be effective in counting wedges for certain datasets with specific structural properties, such as sparsity or unique connectivity patterns. Another interesting finding emerges from the Amazon dataset for wedge counting, where the *2Phases-RLDP* protocol attains an MRE value of 0.1520. This value is only marginally higher than the *MPC-G* protocol (0.0216) and the *MPC-L* protocol (0.0055). Considering communication overhead, this result implies that the *2Phases-RLDP* protocol could be a viable choice for certain datasets where the data distribution and structure align well with its underlying assumptions. These exceptional cases underline the importance of understanding the specific properties and characteristics of the datasets when selecting an appropriate protocol for a given task.

In conclusion, the careful comparison of MRE values emphasizes the benefits of the MPC-based protocols, *MPC-G* and *MPC-L*, in terms of utility, consistency, privacy preservation, and scalability. These factors position them as the preferred protocols for edge and wedge counting tasks among the evaluated alternatives.

Triangle counting task. In Fig. 6, we illustrate the MRE values associated with the triangle counting task, spanning ε values from 1 to 10. Globally, the MRE values for all protocols exhibit a decreasing trend with increasing ε values, irrespective of the specific real-world dataset being assessed. It is important to note that distinct protocols display unique patterns in their response to fluctuations in ε.

Among these protocols, $ARRTwoNS_\triangle$ demonstrates notable robustness against fluctuations in ε. To illustrate, within the Twitter dataset, the MRE of the $ARRTwoNS_\triangle$ protocol experiences a decrement from 0.0824 at $\varepsilon = 1$ to 0.0549 at $\varepsilon = 3$. Subsequent to this interval, the MRE reduction rate exhibits a diminishing trend, ultimately converging to 0.0521 when ε is assigned a value of 10. The $WShuffle^*_\triangle$ protocol is also of particular interest, as it is characterized by a decline curve that resembles a negative exponential function when the graph data comprises a relatively small node count. For instance, in the Facebook dataset, as depicted in Fig. 6(2), the MRE of $WShuffle^*_\triangle$ undergoes a decrease as ε initially increases. As the value of ε continues to rise, the reduction rate exhibits a decelerating pattern, suggesting that the MRE could approach a plateau given a sufficiently large privacy budget ε. This distinct behavior can also be observed in the proposed MPC-based protocols and the $2Phases\text{-}RLDP_\triangle$ protocol. However, when analyzing graph data containing a greater number of nodes, such as the Stanford dataset and the Wiki-T dataset, the decline curve of

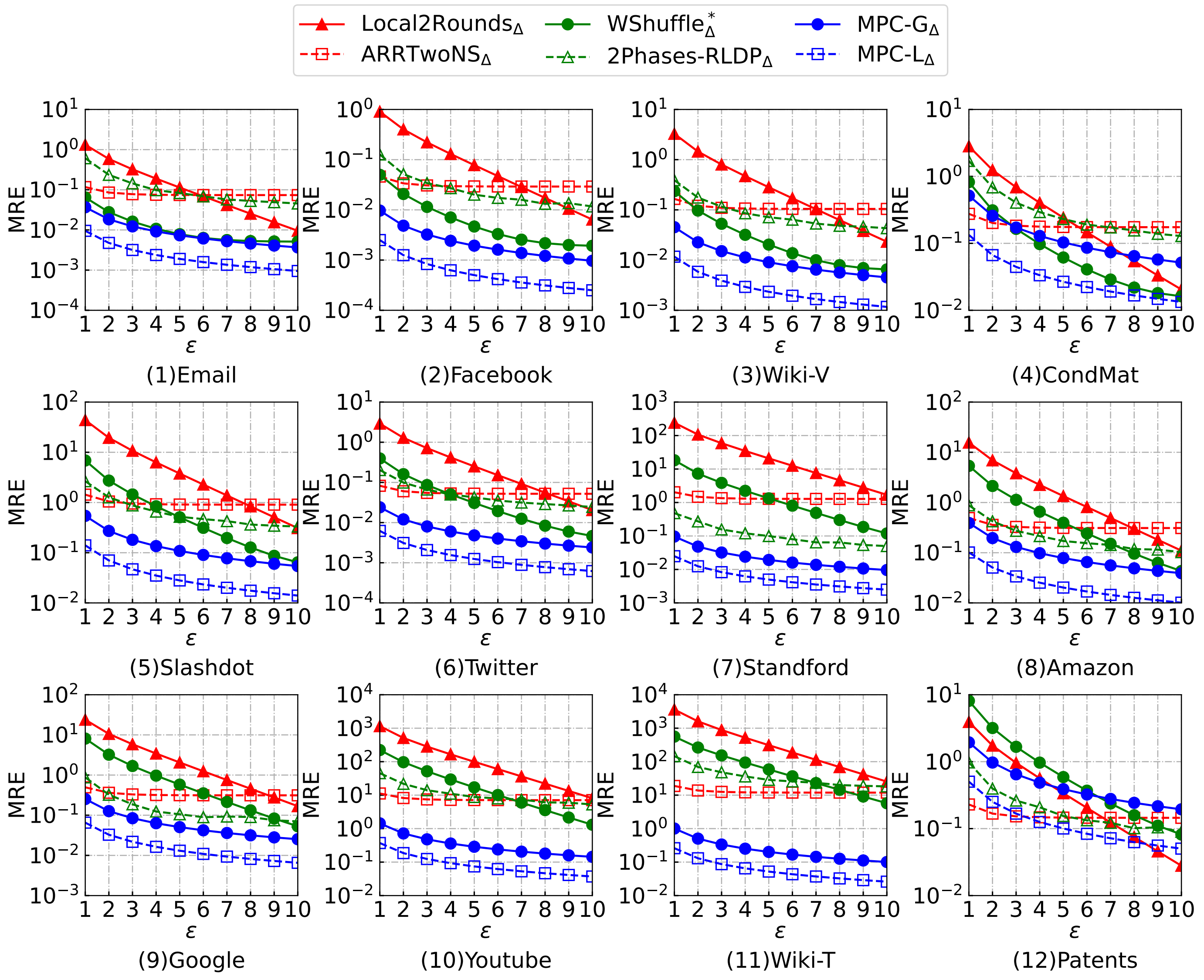


Fig. 6: MRE result for the triangle counting task.

WShuffle$^*_\Delta$ is nearly linear with respect to the increment of ε. This pattern is comparable to the *Local2Rounds*$_\Delta$ protocol, which displays an almost linear decline curve across all examined datasets.

In an analysis encompassing all investigated datasets, our proposed *MPC-G* and *MPC-L* protocols consistently attain the lowest MRE values, resulting in the highest utility. Within each subgraph illustrating the outcomes of the examined datasets, the MRE curve associated with our proposed MPC-based methods invariably occupies the most favorable position. This finding underscores the superiority of our proposed protocols in terms of utility, adaptability, and scalability. For example, with ε set to 6, our approach outperforms the best existing strategies by a factor of 40 (1.59 times for the smallest dataset and 275 times for the largest dataset).

An exception to this trend can be observed in the data presented in Fig. 6(12). Specifically, for $\varepsilon = 1, 2$ and $\varepsilon = 9, 10$, the *ARRTwoNS*$_\Delta$ and *Local2Rounds*$_\Delta$ protocols exhibit lower MRE values compared to our *MPC-L* protocol. This deviation may be ascribed to factors such as node count, graph sparsity, and the clustering coefficient of the graph data. Based on empirical evidence, when the node count is larger, the graph denser, and the clustering coefficient higher, the competing protocols tend to achieve lower MRE values. These observed limitations also suggest potential avenues for refining and optimizing our protocol in future developments, further expanding its applicability to diverse graph data scenarios.

For graph data with node counts below 4,000,000, our protocols remain a more advantageous choice in comparison to other existing protocols. The consistently low MRE values achieved by our protocols suggest that they can provide more accurate results in the triangle counting task, thereby enhancing the quality of the obtained data. Furthermore, our protocols are scalable and can be effectively applied to various types of graph data. The combination of utility, adaptability, and scalability renders our proposed methods an attractive option for graph data analysis.

5 Related Work

• **Subgraph Counting** The purpose of subgraph counting is to identify the occurrences of a specific pattern (or a query graph) within a labeled graph. In this related work section, we first present an overview of traditional subgraph counting methods, followed by a discussion of recent deep learning-based approaches. Lastly, we address the privacy-preserving problem and its significance in the context of subgraph counting tasks.

Traditional studies on subgraph counting primarily fall into two categories: summary-based methods [22], [23], [24] and sampling-based methods [25], [26], [27], [28], [29]. Summary-based methods involve decomposing the query graph into smaller subgraphs and estimating the final result by aggregating the counts of these smaller subgraphs. Conversely, sampling-based methods generate sampled subgraphs and match them to the query subgraphs.

With the advent of deep learning, recent studies [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43] have proposed the use of deep neural networks in subgraph analysis tasks, subject to specific estimation error bounds. These works explore various aspects of subgraph counting, such as network community search in multiplex networks, subgraph containment, benchmarking frameworks, techniques for reducing search space and improving efficiency, and scaling up subgraph matching on single machines using FPGAs. A major focus of these deep learning-based approaches is the development of improved GNN models for subgraph counting, a fundamental aspect of subgraph analysis [29], [30], [31], [34], [35].

Despite the high-performance potential of deep learning-based methods in subgraph counting, few works have addressed the increasingly important privacy-preserving problem. To date, two primary method paradigms have demonstrated effectiveness in tackling this problem: central differential privacy (DP) and local differential privacy (LDP). These paradigms cater to different application scenarios and provide a foundation for further exploration in privacy-preserving subgraph counting.

• **CDP-based Methods for Subgraph Counting.** Subgraph counting under the context of Central Differential Privacy (CDP) has emerged as a significant area of investigation in the field of cybersecurity. This paradigm entails the existence of a reliable third-party server, which is responsible for computing diverse graph metrics from cooperating clients, all the while maintaining the confidentiality of their private data from external inquiries.

In CDP, a trusted centralized curator collects and maintains graph data from individual clients. The curator applies a differential privacy strategy to the consolidated graph, safeguarding sensitive information when releasing different types of graph statistics, including connections, subgraph counts, and degree distribution. A majority of prior research considers two graphs as neighbors if they differ by a single edge, referred to as the edge-differential privacy model. Several studies have focused on developing algorithms that ensure privacy under this model while providing accurate graph statistics.

In contrast to edge-differential privacy, node-differential privacy pertains to cases where one graph is derived from another by merely adding or removing a node. This model has been explored by various researchers. Liu et al. [44] introduced the concept of dependent differential privacy, where nodes in graph data are correlated with one another. They subsequently proposed a dependent perturbation mechanism to ensure privacy against adversaries with prior knowledge of node dependence. Borgs et al. [45] employed a novel variation of Lipschitz extensions on large, sparse networks under node-differential privacy. They later improved upon their work [46] by conducting a more advanced analysis of the upper bound, thus reducing their error rate approximately quadratically. In a subsequent study, Sealfon [47] developed an edge counting estimator constrained by node-differential privacy for instances where Lipschitz extensions are unknown or unavailable. Their results exhibited greater efficiency than those of Borgs' works. Existing literature on centralized graph settings, encompassing both edge and node differential privacy, relies on a trusted curator with comprehensive knowledge of the entire graph.

However, our method focuses on a different scenario in which each subgraph owner applies DP algorithms locally before uploading data to the curator. This approach mitigates the risk of data breaches, as it can be challenging to find a reliable third-party data aggregator in real-world situations.

• **LDP-based Methods for Subgraph Counting.** In the context of local differential privacy (LDP), the entire graph is distributed among participating clients, with the curator lacking access to their subgraphs. Numerous studies have examined LDP's general procedure in various network security contexts [1], [2], [3], [4], [21], [48], [49], [50], [51], [52]. These approaches require clients to generate carefully designed random coefficients, which they incorporate as noise terms within local query results before transmission to the untrusted data curator. Subsequently, the curator conducts post-processing on the noisy data, estimating final query outputs and disseminating them to the participants.

Qin et al. [48] pioneered work on decentralized graphs employing local differential privacy, devising LDPGen, a multi-phase protocol that generates synthetic graphs and streamlines subsequent graph analysis tasks. Sun et al. [3] enhanced LDPGen by positing that each node maintains a 2-hop extended local view, thereby capitalizing on clients' correlation knowledge and yielding superior experimental results. Liu et al. [2] unveiled Edge-RLDP, a novel method enabling graph curators to allocate disparate local privacy budgets to participants based on nodes' mutuality within the noisy graph. However, these techniques fall short of meeting stringent privacy-preserving stipulations in certain network security contexts, such as online social networks.

Addressing this limitation, Wei et al. [49] tailored a local privacy-preserving strategy for attributed graphs, unveiling AsgLDP and attaining an optimal privacy-utility trade-off while preserving innate graph properties. Ye et al. [50] concentrated on computing the clustering coefficient and community detection in LF-GDPR, harnessing Warner's RR (Randomized Response) for subgraph adjacency matrices. Regrettably, this method introduced significant bias in the triangle count. Although their approximation technique [51] aimed to produce more accurate results, estimations remained biased. Imola et al. [4] observed that two-round communication outperformed LF-GDPR in terms of accuracy and refined their algorithm in [1] by integrating edge sampling and double clipping, concurrently reducing transmission overhead and Laplacian noise. In [21], they proposed one-round triangle and 4-cycle counting algorithms, leveraging a wedge shuffle technique to further expand the local differential privacy budget.

Despite the efficacy of differential privacy strategies in tackling large-scale graph pattern mining tasks in network security, applying LDP to small and medium-sized graphs results in unacceptably high estimation errors. To surmount this challenge, our work innovatively employs multi-party computation (MPC) for counting graph statistics such as edges, wedges, and triangles, taking into account its stringent cryptographic constraints.

6 Conclusion

In this study, we investigate the problem of accurately counting edges, wedges, and triangles in fully distributed graphs, wherein each node retains its connections locally and privately. We note that existing Local Differential Privacy (LDP)-based protocols exhibit substantial estimation errors, particularly for small and medium-sized graphs. To address this limitation, we introduce Multi-Party Computation MPC-based protocols for securely and precisely computing these statistics, ensuring no information about individual node connections is leaked. Through a series of experiments using real-world and simulated datasets, we demonstrate that our proposed protocols achieve up to 100 times greater accuracy compared to state-of-the-art protocols. In the current implementation of our protocol, we assume that each server faithfully executes the protocol. Recent research [53], [54], [55] indicates that data poisoning attacks have the potential to cause the protocol to yield significantly inaccurate results.

As future work, we aim to adapt our protocols to address malicious models. Additionally, we plan to develop MPC-based protocols for counting subgraphs in more complex graph structures, such as directed graphs, signed graphs, and heterogeneous graphs, as well as subgraphs containing more than three nodes.

Acknowledgments

The authors would like to thank the anonymous reviewers for their comments and suggestions. This work was supported in part by the National Key R&D Program of China (2021YFB1715600), National Natural Science Foundation of China (U22B2019, 62272372, 62272379).

References

[1] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. {Communication-Efficient} triangle counting under local differential privacy. In *Proc. USENIX Security'22*, pages 537–554, 2022.
[2] Yuhan Liu, Suyun Zhao, Yixuan Liu, Dan Zhao, Hong Chen, and Cuiping Li. Collecting triangle counts with edge relationship local differential privacy. In *Proc. ICDE'22*, pages 2008–2020. IEEE, 2022.
[3] 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 *Proc. CCS'19*, pages 703–717, 2019.
[4] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. Locally differentially private analysis of graph statistics. In *Proc. USENIX Security'21*, pages 983–1000, 2021.
[5] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In *Security and Cryptography for Networks: 13th International Conference, SCN 2022, Amalfi (SA), Italy, September 12–14, 2022, Proceedings*, pages 516–539. Springer, 2022.
[6] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776. IEEE, 2021.
[7] Weikeng Chen and Raluca Ada Popa. Metal: A metadata-hiding file-sharing system. In *NDSS Symposium 2020*, 2020.
[8] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI*, pages 259–282, 2017.
[9] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.
[10] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, Dan Boneh, et al. Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In *USENIX Security Symposium*, pages 1775–1792, 2021.
[11] Dmitry Kogan, Henry Corrigan-Gibbs, et al. Private blocklist lookups with checklist. In *30th USENIX Security Symposium*, 2021. ISBN 978-1-939133-24-3.
[12] Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. Spectrum: High-bandwidth anonymous broadcast. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 229–248, 2022.
[13] Sacha Servan-Schreiber, Simon Langowski, and Srinivas Devadas. Private approximate nearest neighbor search with sublinear communication. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 911–929. IEEE, 2022.
[14] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. Dory: An encrypted search system with distributed trust. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 1101–1119, 2020.
[15] Internet Security Research Group et al. Prio services: Privacy-respecting application metrics. https://www.abetterinternet.org/post/introducing-prio-services/, 2021. Accessed June 5, 2023.
[16] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.
[17] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *SIGSAC'16*, 2016.
[18] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT'06*, 2006.
[19] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.
[20] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.
[21] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. Differentially private subgraph counting in the shuffle model. *arXiv preprint arXiv:2205.01429*, 2022.
[22] Walter Cai, Magdalena Balazinska, and Dan Suciu. Pessimistic cardinality estimation: Tighter upper bounds for intermediate join cardinalities. In *Proc. SIGMOD'19*, pages 18–35, 2019.
[23] Thomas Neumann and Guido Moerkotte. Characteristic sets: Accurate cardinality estimation for rdf queries with multiple joins. In *Proc. ICDE'11*, pages 984–994. IEEE, 2011.
[24] Giorgio Stefanoni, Boris Motik, and Egor V Kostylev. Estimating the cardinality of conjunctive queries over rdf data using graph summarisation. In *Proc. WWW'18*, pages 1043–1052, 2018.
[25] Xiaowei Chen and John CS Lui. Mining graphlet counts in online social networks. *ACM Transactions on Knowledge Discovery from Data*, 12(4):1–38, 2018.
[26] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join: Online aggregation via random walks. In *Proc. SIGMOD'16*, pages 615–629, 2016.
[27] David Vengerov, Andre Cavalheiro Menck, Mohamed Zait, and Sunil P Chakkappen. Join size estimation subject to filter conditions. *Proceedings of the VLDB Endowment*, 8(12):1530–1541, 2015.

[28] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. Random sampling over joins revisited. In *Proc. SIGMOD'18*, pages 1525–1539, 2018.

[29] Hanchen Wang, Rong Hu, Ying Zhang, Lu Qin, Wei Wang, and Wenjie Zhang. Neural subgraph counting with wasserstein estimator. In *Proc. SIGMOD'22*, pages 160–175, 2022.

[30] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? In *Proc. NeurIPS'20*, pages 10383–10395, 2020.

[31] Xin Liu, Haojie Pan, Mutian He, Yangqiu Song, Xin Jiang, and Lifeng Shang. Neural subgraph isomorphism counting. In *Proc. KDD'20*, pages 1959–1969, 2020.

[32] Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, Jure Leskovec, et al. Neural subgraph matching. *arXiv preprint arXiv:2007.03092*, 2020.

[33] Angjela Davitkova, Damjan Gjurovski, and Sebastian Michel. Lmkg: Learned models for cardinality estimation in knowledge graphs. *arXiv preprint arXiv:2102.10588*, 2021.

[34] Kangfei Zhao, Jeffrey Xu Yu, Hao Zhang, Qiyan Li, and Yu Rong. A learned sketch for subgraph counting. In *Proc. SIGMOD'21*, pages 2142–2155, 2021.

[35] Hao Zhang, Qiyan Li, Kangfei Zhao, Jeffrey Xu Yu, and Yuanyuan Zhu. How learning can help complex cyclic join decomposition. In *Proc. ICDE'22*, pages 3138–3141. IEEE, 2022.

[36] Yuli Jiang, Yu Rong, Hong Cheng, Xin Huang, Kangfei Zhao, and Junzhou Huang. Query driven-graph neural networks for community search: from non-attributed, attributed, to interactive attributed. *Proceedings of the VLDB Endowment*, 15(6):1243–1255, 2022.

[37] Ali Behrouz and Farnoosh Hashemi. Cs-mlgcn: Multiplex graph convolutional networks for community search in multiplex networks. In *Proc. CIKM'22*, pages 3828–3832, 2022.

[38] Ali Behrouz, Farnoosh Hashemi, and Laks VS Lakshmanan. Firmtruss community search in multilayer networks. *arXiv preprint arXiv:2205.00742*, 2022.

[39] Hyunjoon Kim, Yunyoung Choi, Kunsoo Park, Xuemin Lin, Seok-Hee Hong, and Wook-Shin Han. Versatile equivalences: Speeding up subgraph query processing and subgraph matching. In *Proc. SIGMOD'21*, pages 925–937, 2021.

[40] Xin Jin, Zhengyi Yang, Xuemin Lin, Shiyu Yang, Lu Qin, and You Peng. Fast: Fpga-based subgraph matching on massive graphs. In *Proc. ICDE'21*, pages 1452–1463. IEEE, 2021.

[41] 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.

[42] Wilco Van Leeuwen, George Fletcher, and Nikolay Yakovets. A general cardinality estimation framework for subgraph matching in property graphs. *IEEE Transactions on Knowledge & Data Engineering*, (01):1–1, 2022.

[43] Yeonsu Park, Seongyun Ko, Sourav S Bhowmick, Kyoungmin Kim, Kijae Hong, and Wook-Shin Han. G-care: A framework for performance benchmarking of cardinality estimation techniques for subgraph matching. In *Proc. SIGMOD'20*, pages 1099–1114, 2020.

[44] Changchang Liu, Supriyo Chakraborty, and Prateek Mittal. Dependence makes you vulnberable: Differential privacy under dependent tuples. In *Proc. NDSS'16*, volume 16, pages 21–24, 2016.

[45] Christian Borgs, Jennifer T Chayes, and Adam Smith. Private graphon estimation for sparse graphs. In *Proc. NeurIPS'15*, volume 1, pages 1369–1377, 2015.

[46] Christian Borgs, Jennifer Chayes, Adam Smith, and Ilias Zadik. Revealing network structure, confidentially: Improved rates for node-private graphon estimation. In *Proc. FOCS'18*, pages 533–543. IEEE, 2018.

[47] Adam Sealfon and Jonathan Ullman. Efficiently estimating erdős-rényi graphs with node differential privacy. In *Proc. NeurIPS'19*, pages 3770–3780, 2019.

[48] Zhan Qin, Ting Yu, Yin Yang, Issa Khalil, Xiaokui Xiao, and Kui Ren. Generating synthetic decentralized social graphs with local differential privacy. In *Proc. CCS'17*, pages 425–438, 2017.

[49] 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.

[50] Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. Towards locally differentially private generic graph metric estimation. In *Proc. ICDE'20*, pages 1922–1925. IEEE, 2020.

[51] 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.

[52] Hailong Zhang, Sufian Latif, Raef Bassily, and Atanas Rountev. Differentially-private control-flow node coverage for software usage analysis. In *Proc. USENIX Security'20*, pages 1021–1038, 2020.

[53] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Data poisoning attacks to local differential privacy protocols. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[54] Albert Cheu, Adam Smith, and Jonathan Ullman. Manipulation attacks in local differential privacy. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 883–900. IEEE, 2021.

[55] Yongji Wu, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Poisoning attacks to local differential privacy protocols for {Key-Value} data. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 519–536, 2022.

Pinghui Wang (Senior Member, IEEE) is currently a Professor with the MOE Key Laboratory for Intelligent Networks and Network Security, Xi'an Jiaotong University, Xi'an, China, and also with the Shenzhen Research Institute, Xi'an Jiaotong University, Shenzhen, China. His research interests include internet traffic measurement and modeling, traffic classification, abnormal detection, and online social network measurement.

Dongxu Zeng received B.S. in computer science from Zhejiang University of Technology, Hangzhou, China, in 2022. He is currently a Ph.D. student with MOE Key Laboratory for Intelligent Networks and Network Security at Xi'an Jiaotong University. His research interests include big data, data mining, privacy-preserving computation, and secure computation.

Shiqi Lou is currently a junior student with the MOE Key Laboratory for Intelligent Networks and Network Security, Xi'an Jiaotong University, Xi'an, China, and also with Luo lab Undergraduate Division, Xi'an Jiaotong University, Xi'an, China. His research interests include graph neural networks, spatiotemporal neural networks, anomaly detection, social network analysis, and network security.

Haoxin Yang received B.S. in automation engineering from Xi'an Jiaotong University, Xi'an, China, in 2018. He is currently a graduate student with MOE Key Laboratory for Intelligent Networks and Network Security at Xi'an Jiaotong University. His research interests include big data, data mining, privacy-preserving computation, and secure computation.

Tao Qin received B.S. degree in information engineering and Ph.D. degree in computer science and technology from Xi'an Jiaotong University, Xi'an, China, in 2004 and 2010 respectively. He is a visiting scholar at the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst from 2017 to 2018. He is currently a Professor at the Department of Computer Science and Technology, Xi'an Jiaotong University. His research focuses on Data Mining, Complex Network Analysis.