

# PrivAGM: Secure Construction of Differentially Private Directed Attributed Graph Models on Decentralized Social Graphs

Songlei Wang  
Shenzhen University  
songlei.wang@szu.edu.cn

Xiaohua Jia  
City University of Hong Kong  
csjia@cityu.edu.hk

Yifeng Zheng  
The Hong Kong Polytechnic University  
yifeng.zheng@polyu.edu.hk

Haibo Hu  
The Hong Kong Polytechnic University  
haibo.hu@polyu.edu.hk

## ABSTRACT

Decentralized social graphs, where no single entity possesses the information of the entire graph, and each user maintains only a limited view of the graph, contain great value for different applications. However, simply collecting local views for analytics raises privacy concerns due to the sensitive information of social relationships they capture. To address this, a canonical approach involves privately fitting a generative graph model to the decentralized social graph, generating a differentially private synthetic graph that serves as a proxy for analytics. Existing solutions, however, often fail to capture the inherent directionality of edges and attribute-edge correlations when dealing with decentralized directed social graphs, leading to synthetic graphs with poor utility. To bridge this gap, we present PrivAGM, a new solution that harnesses the synergies among differential privacy, secure multiparty computation, and generative graph models, enabling the secure construction of differentially private directed attributed graph models on decentralized social graphs while ensuring the privacy preservation of individuals. We evaluate PrivAGM on three real-world directed social graph datasets. The results show that PrivAGM outperforms the state-of-the-art methods, generating synthetic graphs with significantly higher utility.

## 1 INTRODUCTION

With the continuous advancements in graph analytics, a multitude of valuable applications can stem from the in-depth exploration of a social graph. However, social graph analytics becomes quite challenging when the entire graph is not available to a single entity and stored in a *decentralized* manner. In this scenario, each user possesses only a limited local view comprising its attributes and neighbor list (i.e., a list of the indices of users directly connected to them) in the social graph, and the complete information of the social graph is formed by the collective views of all users. Decentralized social graphs exist in various practical applications [19, 34, 41]. One prominent example is contact-tracing applications (e.g., Apple/Google Exposure Notification framework [2]) for infectious disease control, which are installed on users' personal devices. These applications log interactions between devices via Bluetooth, resulting in a contact-tracing network that is inherently decentralized. Each user sees only their direct contacts, and the global contact network exists only in the union of all these local datasets. Another example is emerging decentralized social networking platforms like Scuttlebutt [40], which represent a new generation of

social network architectures focused on providing strong privacy guarantees through user-controlled data distribution.

In the context of decentralized social graphs, the collection of individual users' local views for analytics may raise significant privacy concerns, particularly when these local views contain sensitive information about social interactions, political preferences, and religious affiliations [2, 19, 20, 34, 41]. If users' local views are not sufficiently protected, it would discourage them from participating in such analytics. Thus, it is crucial to incorporate security measures into analytics over decentralized social graphs from the outset, facilitating the development and deployment of valuable applications without compromising users' privacy.

Privacy-preserving analytics over decentralized social graphs has received significant attention in recent years [5, 19–21, 34, 41, 49]. Most existing methods [19–21, 41] have adopted local differential privacy (LDP) mechanisms [23] to collect certain noisy statistics from the decentralized social graph for use. However, this line of work suffers from a major deficiency that limits their practical usability. Specifically, as more statistics are collected from the decentralized social graph, the total amount of revealed information monotonically increases, eventually reaching a threshold where collecting any new statistics leads to an excessive privacy risk [11].

Alternatively, a more promising direction is to privately fit a generative graph model to the decentralized social graph, producing a differentially private generative graph model. A synthetic graph is then generated from the model, emulating key characteristics of the original decentralized graph—such as degree distribution, attribute–edge correlation distribution, and community structure—and serving as a reliable proxy for conducting social graph analytics tasks. For example, synthetic graphs built from decentralized social networking services (e.g., Scuttlebutt [40]) can be used to detect the presence of malicious users by analyzing the connection strength between vertices, or to simulate community message dissemination mechanisms and validate community dissemination algorithms. Synthetic graphs built from contact-tracing applications (such as the Apple/Google Exposure Notification framework [2]) can be used to detect the presence of disease hotspots or simulate disease propagation mechanisms. Note that based on the post-processing invariance property of differential privacy (DP) [13], the synthetic graph remains differentially private.

**Existing solutions.** In the literature, there have been a number of works considering the setting of *centralized* graphs for the construction of differentially private generative graph models (e.g., [6, 22, 52], to list a few). On the contrary, little work [5, 34, 49, 54] has

been done for constructing differentially private generative graph models over *decentralized* social graphs. The works in [5, 34, 54] concentrate solely on modeling graph structure, disregarding vertex attributes and their correlations with the graph structure. Nevertheless, real-world social graphs possess vertex attributes and exhibit such correlations. These correlations can be leveraged in various analyses; for instance, in the field of relational machine learning, they are used to predict missing or future attribute values [22]. The prior work most closely related to ours is AsgLDP [49], which aims to construct differentially private *attributed* graph models (AGMs) on decentralized social graphs.

However, AsgLDP, as well as other works [5, 34, 54], focus on idealized decentralized *undirected* social graphs, where each user holds its *complete* neighbor list. In practice, however, decentralized social graphs can also be *directed*, where each user holds the neighbor list consisting of only “one-sided relationships”. For example, in decentralized social networking platforms like Scuttlebutt [40], a user can follow another user unilaterally, without requiring any reciprocal action. Another example is the contact tracing network, which can be either directed or undirected, depending on the specific application context. For instance, in the Apple/Google Exposure Notification framework [2], each device independently records only the identifiers of nearby devices it detects, resulting in inherently one-sided contact logs that do not require mutual consent. In this case, device A may detect device B, but device B does not necessarily detect device A. However, all previous works do not consider the inherent directionality of the edges when constructing the generative graph model, leading to synthetic graphs with poor utility when dealing with directed graphs. Therefore, secure construction of differentially private *directed* AGMs (dAGMs) on decentralized directed social graphs remains to be fully explored.

**Our contributions.** We address the following three challenges when endeavoring to design PrivAGM.

*Challenge 1: How to collect neighbor lists while striking a balance between privacy, utility, and efficiency?*

To privately collect neighbor lists, a widely-used method is the randomized neighbor list (RNL) method [34], which enables users to flip each bit in their neighbor lists by a certain probability to achieve pure edge LDP. However, the RNL mechanism significantly disrupts the structure of the decentralized social graph, as it requires flipping a considerable number of bits in each neighbor list to achieve the desired level of privacy protection. Another plausible method is to let users conceal each bit of their neighbor lists using secure multiparty computation (MPC) techniques (e.g., additive secret sharing [9]). However, this incurs significant performance overhead as the neighbor lists are normally sparse [8]. To strike a balance between privacy, utility, and efficiency, we instead propose layering the concept of selective MPC [18] with RNL to protect the neighbor lists. Unlike typical secret sharing-based MPC techniques, where all computing parties receive shares for all private values, selective MPC works by randomly selecting a subset of parties to receive shares for each private value. With our sparsity-aware neighbor list secret sharing method, PrivAGM achieves pure edge LDP for the neighbor lists *without introducing any noise that would compromise their utility* or incurring significant performance overhead.

*Challenge 2: How to securely extract the differentially private characteristics essential for dAGM from the collected secret-shared social graph while ensuring their utility?*

Constructing the differentially private dAGM requires the addition of calibrated noises to the extracted characteristics from the original graph. However, for certain characteristics (e.g., triangle subgraph count), the global sensitivity of the calibrated noises is large, which could severely compromise the utility of the extracted characteristics. To overcome this challenge, we develop techniques for oblivious edge clipping, enabling the oblivious restriction of the maximum degree of the secret-shared social graph. This helps mitigate the worst-case impact of a single vertex or edge and achieve a small global sensitivity. Moreover, we develop a suite of MPC-based secure components, including degree sequence extraction, attribute-edge correlation distribution extraction, attribute distribution extraction, and triangle counts extraction. Their integration allows to securely produce the differentially private characteristics essential for constructing the dAGM.

*Challenge 3: How to construct the dAGM to effectively capture the inherent directionality of the edges in the decentralized social graph?*

The existing AGMs [6, 22, 33, 49] have mainly focused on *undirected* graphs, resulting in a lack of capability to capture the inherent directionality of edges within the original graph. This directionality is of utmost importance for various analytics tasks on directed graphs, such as the discovery of social roles [12] and prediction of disease outbreaks [22]. However, no existing AGMs can be trivially adapted to capture the inherent directionality of edges. To address this, we identify key graph characteristics—such as vertex degrees, attribute-edge correlations, and triangle subgraph counts—that can capture the directionality, filling the gap in the field of AGMs. Specifically, we begin with the widely used Chung-Lu (CL) model [7], which operates on undirected graphs, and develop a directed version of the CL model to incorporate edge directionality as reflected in vertex degrees. Building on this directed CL model, we modify the classical AGM framework [22] to capture the directionality in attribute-edge correlations and triangle subgraph counts.

**Evaluation results.** Our results clearly demonstrate that, in terms of utility of synthetic graphs, PrivAGM not only outperforms the state-of-the-art decentralized setting-based method AsgLDP [49] but also surpasses the state-of-the-art centralized setting-based method CAGM [6]. For instance, in terms of the Hellinger distance between the degree distribution of the synthetic and input social graphs, PrivAGM is 58.3% lower than AsgLDP, and also is 42.9% lower than CAGM, with the privacy budget  $\epsilon = 1$  on the Gplus dataset [28]. We also evaluate the system cost. Notably, with  $\epsilon = 1.2$ , PrivAGM’s sparsity-aware neighbor list secret sharing method achieves about 70% savings in the size of resulting shares over the simple method of secretly sharing every bit of the neighbor lists.

## 2 PRELIMINARIES

### 2.1 Attributed Graphs

This paper focuses on directed and unweighted attributed graphs. An attributed graph is represented as  $G = (\mathbf{A}, \mathbf{F})$ . Here,  $\mathbf{A} \in \{0, 1\}^{N \times N}$  is the asymmetric binary adjacency matrix of size  $N \times N$ , where  $N$  is the number of vertices. Each vertex is denoted by  $i \in [1, N]$ .  $\mathbf{A}[i, j] = 1$  indicates the presence of a directed edge

from vertex  $i$  to vertex  $j$ , while  $\mathbf{A}[i, j] = 0$  indicates the absence of such a directed edge. We exclude self-connected edges in our consideration, meaning that  $\mathbf{A}[i, i] = 0, i \in [1, N]$ . The number of edges in  $G$  is denoted as  $M$ . We use  $\mathbf{A}[i, :]$  to denote the  $i$ -th row of  $\mathbf{A}$ , which corresponds to the out-neighbors of vertex  $i$ . The number of vertex  $i$ 's out-neighbors (resp. in-neighbors) is named as its out-degree (resp. in-degree), denoted as  $d_i^+$  (resp.  $d_i^-$ ).  $\mathbf{F} \in \{0, 1\}^{N \times L}$  is the attribute matrix, where  $L$  is the dimensions of each vertex's attribute vector.  $\mathbf{F}[i, :]$  is vertex  $i$ 's attribute vector. The attribute vectors are assumed to be binary, following previous works [6, 22, 33, 49].

We use  $\{\sigma_i\}_{i \in [1, \gamma]}$  to denote the set  $\{\sigma_1, \dots, \sigma_\gamma\}$ , and omit the subscript  $i \in [1, \gamma]$  when it does not impact the presentation.

## 2.2 Attributed Graph Model (AGM)

We provide an overview of the classical and widely adopted AGM [33] as a foundational approach to demonstrate AGM's underlying principles. Given the input graph, the AGM learns:

- The degree sequence of all vertices:  $\mathcal{D} = \{d_i\}_{i \in [1, N]}$ .
- The attribute distribution  $\Theta_F : \Theta_F(\mathbf{y}) = \frac{n_{\mathbf{y}}}{N}$ , where  $n_{\mathbf{y}}$  is the number of vertices in the graph that have attribute vector  $\mathbf{y}$ .
- The attribute-edge correlation distribution  $\Theta_X : \Theta_X(\mathbf{y} - \mathbf{y}') = \frac{n_{\mathbf{y}-\mathbf{y}'}}{M}$ , where  $\mathbf{y} - \mathbf{y}'$  is named as *attribute-edge correlation* (i.e., an edge connecting two vertices with attribute vector  $\mathbf{y}$  and  $\mathbf{y}'$ , respectively),  $n_{\mathbf{y}-\mathbf{y}'}$  is the number of  $\mathbf{y} - \mathbf{y}'$  in the input graph.

To generate a graph, the AGM samples an attribute vector for each vertex from  $\Theta_F$ . It then iteratively samples edges based on the new attributes from  $\mathcal{D}$  and  $\Theta_X$ .

## 2.3 Differential Privacy for Graphs

In this paper, we consider two variants of DP in the context of attributed graphs: edge LDP [34] and edge DP [22].

**DEFINITION 1.** (Edge LDP [34]) A randomized mechanism  $\mathcal{M}$  satisfies  $\epsilon$ -edge LDP, if and only if for any two neighbor lists  $\mathbf{A}[i, :]$ ,  $\mathbf{A}'[i, :]$  that differ in one bit, we have:  $\forall \mathbf{b} \in \text{Range}(\mathcal{M})$ ,

$$\Pr[\mathcal{M}(\mathbf{A}[i, :]) = \mathbf{b}] \leq e^\epsilon \cdot \Pr[\mathcal{M}(\mathbf{A}'[i, :]) = \mathbf{b}],$$

where  $\text{Range}(\mathcal{M})$  is  $\mathcal{M}$ 's possible outputs;  $\epsilon$  is the privacy budget.

**DEFINITION 2.** (Edge DP [22]) A randomized mechanism  $\mathcal{M}$  satisfies  $(\epsilon, \delta)$ -edge DP, if and only if for any two neighboring attributed graphs  $G$  and  $G'$  that differ in the presence of a single edge or in the attribute associated with a single vertex, we have:  $\forall \mathcal{G} \in \text{Range}(\mathcal{M})$ ,

$$\Pr[\mathcal{M}(G) = \mathcal{G}] \leq e^\epsilon \cdot \Pr[\mathcal{M}(G') = \mathcal{G}] + \delta.$$

If  $\delta = 0$ , we say that  $\mathcal{M}$  provides pure edge DP. The discrete Laplace distribution  $\text{Lap}(\cdot)$  is commonly employed to draw discrete noises to provide DP.

**DEFINITION 3.** (Discrete Laplace distribution [17]) A discrete random variable follows  $\text{Lap}(\epsilon, \delta, \Delta)$  if its probability mass function:

$$\Pr[x] = \frac{e^{\frac{\epsilon}{\Delta}} - 1}{e^{\frac{\epsilon}{\Delta}} + 1} \cdot e^{\frac{-\epsilon \cdot |x - \mu|}{\Delta}}. \quad (1)$$

where  $\mu$  is the mean of the distribution and  $\Delta$  is the sensitivity.

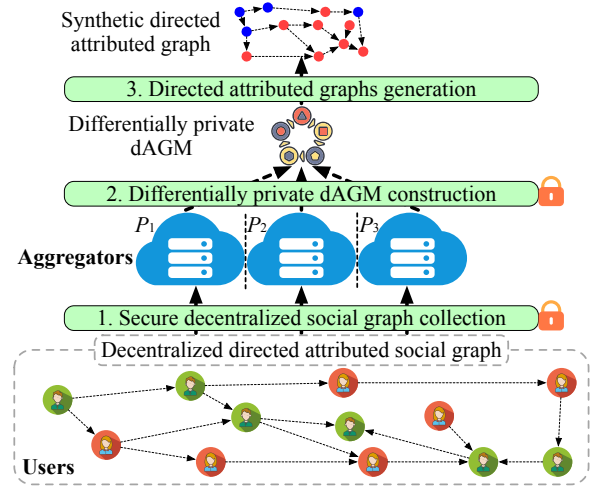


Figure 1: The system architecture of PrivAGM.

## 2.4 Additive Secret Sharing

In 2-out-of-2 additive secret sharing (ASS) [9], a secret value  $x \in \mathbb{Z}_{2^l}$  is split into two secret shares  $\langle x \rangle_1$  and  $\langle x \rangle_2$ , which are distributed to two parties  $P_1$  and  $P_2$ , respectively. We denote the ASS of  $x$  as  $\llbracket x \rrbracket$ . If only certain values in an entity  $\Omega$  (e.g., set or table) are secret-shared, we represent the entity as  $\llbracket \Omega \rrbracket$ . The basic secure operations in the ASS domain are as follows: (1) Linear operations: Given constants  $c, c'$  and secret-shared values  $\llbracket x \rrbracket, \llbracket y \rrbracket$ , computing  $\llbracket z \rrbracket = \llbracket c \cdot x + c' \cdot y \rrbracket$  involves  $P_{i \in \{1, 2\}}$  locally computing  $\langle z \rangle_i = c \cdot \langle x \rangle_i + c' \cdot \langle y \rangle_i$ . (2) Multiplication operations: To compute  $\llbracket z \rrbracket = \llbracket x \cdot y \rrbracket$ ,  $P_{1, 2}$  prepare a Beaver triple  $(\llbracket w \rrbracket, \llbracket u \rrbracket, \llbracket v \rrbracket)$  offline, where  $w = u \cdot v$ .  $P_{i \in \{1, 2\}}$  computes  $\langle e \rangle_i = \langle x \rangle_i - \langle u \rangle_i$ ,  $\langle f \rangle_i = \langle y \rangle_i - \langle v \rangle_i$ , then communicates with each other to reveal  $e, f$ . Finally,  $P_{1, 2}$  compute  $\langle z \rangle_1 = e \cdot f + f \cdot \langle u \rangle_1 + e \cdot \langle v \rangle_1 + \langle w \rangle_1$  and  $\langle z \rangle_2 = f \cdot \langle u \rangle_2 + e \cdot \langle v \rangle_2 + \langle w \rangle_2$ , respectively.

## 3 PROBLEM STATEMENT

### 3.1 System Architecture

PrivAGM's system architecture is illustrated in Fig. 1. In the system, there are two types of entities: users and aggregators. The  $N$  users, along with the connections among them, form the decentralized social graph  $G = (\mathbf{A}, \mathbf{F})$ . Each user corresponds to a vertex  $i \in [1, N]$  and holds a local view  $(\mathbf{A}[i, :], \mathbf{F}[i, :])$ , where  $\mathbf{A}[i, :]$  represents user  $i$ 's neighbor list held by it (named as *local neighbor list*) and  $\mathbf{F}[i, :]$  represents user  $i$ 's attribute vectors. In contrast to previous studies [34, 49] that focus on idealized decentralized social graphs where each user's local neighbor list *completely* includes its out/in-neighbors, i.e.,  $\mathbf{A}$  is symmetric, our attention is directed towards a more realistic *directed* social graph scenario. Here, each user's local neighbor list only includes its out-neighbors, i.e.,  $\mathbf{A}$  is asymmetric. Such decentralized social graphs are more commonly encountered in practice (e.g., the phone networks [34] and the Google-Apple Exposure Notification framework [2]).

The aggregators collect decentralized social graph data from users to perform various graph analytics tasks, and users can remain offline after uploading their data. However, privacy concerns hinder users' participation in the tasks [34, 52]. In this paper, we leverage a distributed trust framework where three aggregators (referred to

as  $P_{\{1,2,3\}}$  from different trust domains cooperatively construct the dAGM. Such framework has recently gained significant attentions in both academia [36, 42–44, 46, 53] and industry [16, 29, 31].

### 3.2 Threat Model and Security Guarantees

**Threat model.** We consider a semi-honest and non-colluding adversary model, where each aggregator honestly follows the protocol, but may *individually* infer users' local views. In practice, such non-colluding aggregators could be cloud servers hosted by different competitive commercial cloud providers (e.g., Amazon, Microsoft, and Google). The competitive nature of these providers creates strong incentives for them to maintain independent and non-collusive operations. Such non-colluding multi-server model has also been widely adopted for building secure database applications (e.g., [36, 44, 46, 53]) and is also utilized in industry (e.g., Safeheron Wallet [37]). Additionally, we assume that the users are trusted, as they only provide their local views.

**Security guarantees.** From the view of individual aggregators, PrivAGM guarantees the following: (1) during the decentralized social graph collection phase, the local neighbor lists are differentially private (edge LDP); (2) during the dAGM construction phase, the learned dAGM is differentially private (edge DP). Based on the post-processing invariance of DP [13], the synthetic graphs generated by the differentially private dAGM are still differentially private.

## 4 SECURE SOCIAL GRAPH COLLECTION

**Motivation.** To protect  $\mathbf{A}[i, :]$ , a simple method is to directly apply lightweight ASS over the entire  $\mathbf{A}[i, :]$  of length  $N$  (i.e., dense encoding). However, this method is inefficient due to the sparsity of social graphs. According to Facebook's statistics [8], the average user has around 130 friends in a social network, which is much less than the total number of vertices, i.e.,  $N$ . As a result,  $\mathbf{A}[i, :]$  will be mostly filled with zeros, leading to high sparsity. Therefore, applying ASS over  $\mathbf{A}[i, :]$  would result in significant uplink communication cost, as well as unnecessary workload during the subsequent dAGM construction. Another simple method is to consider only the non-zero bits in  $\mathbf{A}[i, :]$  (i.e., sparse encoding) and apply ASS solely to the indices of these non-zero bits. Specifically, user  $i$  stores  $\mathbf{A}[i, :]$  in sparse encoding:  $E_i = \{(i, j) | \mathbf{A}[i, j] = 1, j \in [1, N]\}$ , and then secretly shares  $E_i$  as  $[E_i] = \{(i, \llbracket j \rrbracket) | \mathbf{A}[i, j] = 1, j \in [1, N]\}$ . However, the secret sharing of indices makes it difficult to securely access the neighbors of each user. Since accessing neighbors is a fundamental operation required to extract parameters necessary for dAGM, the method also imposes a heavy workload on the aggregators.

**Observation.** We observe that while both offering privacy protection, the two aforementioned methods represent two extremes: the first one allows efficient accessing of neighbors but sacrifices sparsity, while the second one preserves sparsity but hinders efficient neighbor accessing. This indicates that the challenge here lies in balancing the preservation of privacy and the benefits of sparsity, while facilitating efficient neighbor accessing. We note that employing edge LDP [34] is a strategic approach to address the challenge. To achieve edge LDP, our starting point is the randomized neighbor list (RNL) mechanism. RNL applies randomized response [47], allowing each user to flip each bit of its local neighbor list with a certain probability. However, to achieve the desired level of privacy

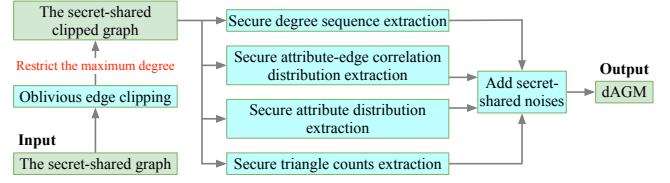


Figure 2: Running example of private dAGM construction.

protection, RNL must flip a considerable number of bits in each local neighbor list, which significantly disrupts the structure of the decentralized social graph. To address this issue, our insight is to layer the concept of selective MPC [18] with RNL to achieve pure edge LDP for the local neighbor lists *without compromising their utility* or incurring significant performance overhead. Unlike typical secret sharing-based MPC techniques [9], where all computing participants receive secret shares for all private values, selective MPC randomly selects a subset of participants to receive secret shares for each private value.

**Layering selective MPC with RNL.** Given  $\mathbf{A}[i, :]$ , user  $i$  first constructs the set  $E_i = \{(i, j, 1) | \mathbf{A}[i, j] = 1, j \in [1, N]\}$  to fully capture the nonzero bits in  $\mathbf{A}[i, :]$ . Next, for every zero bit  $\mathbf{A}[i, j] = 0, j \in [1, N]$ , user  $i$  includes  $(i, j, 0)$  to  $E_i$  with a probability of  $p$ . The resulting local neighbor list is denoted as  $E_i = \{(i, j, b_{ij})\}$ , where  $b_{ij} = 1$  or  $0$ . User  $i$  then applies ASS over each  $b_{ij}$  in  $E_i$  to obtain the secret shares  $[E_i] = \{(i, j, \llbracket b_{ij} \rrbracket)\}$ . After that, a straightforward method is to distribute  $\{(i, j, \langle b_{ij} \rangle_1)\}$  and  $\{(i, j, \langle b_{ij} \rangle_2)\}$  to  $P_1$  and  $P_2$ , respectively. However, this approach cannot achieve pure edge LDP in the view of  $P_1$  or  $P_2$  because it solely adds zero bits to  $E_i$  and does not remove nonzero bits from  $E_i$ , thereby capturing only additive noise without accounting for subtractive noise. Our insight to address this issue is to leverage the concept of selective MPC [18]. Specifically, for each  $(i, j, \llbracket b_{ij} \rrbracket) \in [E_i]$ , user  $i$  sends  $(i, j, \langle b_{ij} \rangle_1)$  and  $(i, j, \langle b_{ij} \rangle_2)$  to two randomly selected aggregators, respectively. Thus, each aggregator only has access to a random subset of  $[E_i]$ . In other words, our method simulates removing some nonzero bits from the view of each aggregator, capturing subtractive noise (similar to how adding zero bits captures additive noise), thereby achieving pure edge LDP in each aggregator's view (as proved in Section 7.2) without compromising the utility.

For each  $(i, j, \llbracket b_{ij} \rrbracket) \in [E_i]$ , user  $i$  allows the two selected aggregators to be aware of each other to facilitate subsequent computations in the ASS domain. We use  $\{[E_i]^{(1,2)}\}$ ,  $\{[E_i]^{(2,3)}\}$ , and  $\{[E_i]^{(3,1)}\}$  to represent the secret-shared local neighbor lists of all users jointly held by the pairs of aggregators  $(P_1, P_2)$ ,  $(P_2, P_3)$ , and  $(P_3, P_1)$ , respectively, and use  $\{[E_i]\} = \{[E_i]^{(1,2)} \cup [E_i]^{(2,3)} \cup [E_i]^{(3,1)}\}$  to represent the complete secret-shared neighbor lists.

**Secretly sharing the attributes.** Before encrypting the attribute vector  $\mathbf{F}[i, :]$ , user  $i$  first parses it into a value:

$$f_i = \sum_{l \in [1, L]} \mathbf{F}[i, l] \cdot 2^l. \quad (2)$$

User  $i$  secretly shares  $f_i$  into  $\llbracket f_i \rrbracket$  and distributes  $\langle f_i \rangle_1$  to  $P_1$ ;  $\langle f_i \rangle_2$  to  $P_2$ . Finally, the secret-shared graph is  $(\{[E_i]\}, \{\llbracket f_i \rrbracket\})$ .

## 5 PRIVATE DAGM CONSTRUCTION

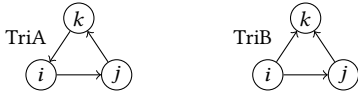
Next, we describe how the aggregators securely construct a differentially private dAGM on the collected graph. Note that during the

graph collection phase, users do not add any true noise to perturb their local views. As a result, if the aggregators directly recover the dAGM parameters extracted from the secret-shared graph, the resulting dAGM will not be differentially private. Therefore, the aggregators should add secret-shared noise to the dAGM parameters before recovering them, ensuring the construction of a differentially private dAGM. Fig. 2 provides an overview of the process.

**Parameters necessary for dAGM.** Our dAGM is built on the widely adopted AGM [33], which involves learning three sets of parameters from the input attributed graph: the degree sequence  $\mathcal{D}$ , the attribute-edge correlation distribution  $\Theta_X$ , and the attribute distribution  $\Theta_F$ . However, it focuses on undirected graphs, whereas PrivAGM specifically addresses directed graphs. Hence, we extend it by adapting its parameters, enabling to capture the directionality in the input directed graph. Since the attribute does not possess directionality, we only focus on adapting the other parameters.

When considering the degree sequence, we note that in directed graphs, each vertex has both out/in-degree. Hence, to capture the directionality by the degree sequence, we tailor PrivAGM to learn the sequence of (out-degree, in-degree) pairs:  $\mathcal{D}^\pm = \{(d_i^+, d_i^-)\}_{i \in [1, N]}$ . When considering the attribute-edge correlation distribution, we tailor our dAGM to learn  $\Theta_{\tilde{X}} : \Theta_{\tilde{X}}(f \rightarrow f') = \frac{n_{f \rightarrow f'}}{M}, f \rightarrow f' \in \mathbb{Z}_{2^L} \times \mathbb{Z}_{2^L}$ , where  $n_{f \rightarrow f'}$  is the number of edges from a vertex with attribute value  $f$  (mapped from an attribute vector by Eq. 2) to a vertex with attribute value  $f'$ . The distinction between our  $\Theta_{\tilde{X}}$  and the AGM [33]'s  $\Theta_X$  lies in our consideration of attribute-edge correlations  $f \rightarrow f'$  and  $f' \rightarrow f$  as distinct, whereas the AGM [33] treats them as equivalent. Hence,  $\Theta_{\tilde{X}}$  can capture the edge directionality.

The work [22] points out that the count of triangles in the input graph plays a pivotal role in refining the structure of synthetic graphs. However, it considers all triangles to be identical, thus also failing to capture the directionality of edges. To address the limitation, we propose counting non-isomorphic triangles separately. Specifically, we identify two non-isomorphic triangles “TriA” and “TriB”, regardless of the vertex attributes:



Our dAGM counts the number of occurrences of TriA, TriB in the input graph separately, denoted as  $n_{\Delta}^A, n_{\Delta}^B$ , respectively.

## 5.1 Oblivious Edge Clipping

**Motivation.** Edge clipping is necessary due to the significant sensitivity  $\Delta$  involved in achieving edge DP for  $\Theta_{\tilde{X}}, n_{\Delta}^A$ , and  $n_{\Delta}^B$ . For example, the addition or removal of a single edge changes  $n_{\Delta}^A$  or  $n_{\Delta}^B$  by at most  $d_{\max} - 1$ , where  $d_{\max}$  is the maximum (out or in)-degree in the input graph, which can range up to  $N - 1$ . Setting such sensitivities and adding noises will completely destroy the utility of the extracted parameters. Hence, it is crucial to mitigate the worst-case impact of a single vertex or edge. This can be accomplished by applying edge clipping on the original graph to restrict its maximum out/in-degree. A naive method would be for each user to locally perform edge clipping on their local neighbor lists. However, in decentralized graphs, each user holds an incomplete neighbor list. **Basic oblivious edge clipping.** Our (basic) oblivious edge clipping algorithm is built on the widely-used edge clipping algorithm in the

### Algorithm 1: Basic Oblivious Edge Clipping

---

**Input:** Secret-shared neighbor lists  $\{[E_i]\}$ ; clipping parameter  $k$ .  
**Output:** Secret-shared clipped local neighbor lists  $\{[\bar{E}_i]\}$ .

- 1 //  $P_{1,2}$  operate on  $\{[E_i]^{(1,2)}\}$ ;
- 2 Initialization:  $\llbracket d_i^+ \rrbracket = \llbracket 0 \rrbracket; \llbracket d_i^- \rrbracket = \llbracket 0 \rrbracket; [\bar{E}_i]^{(1,2)} = \emptyset, i \in [1, N]$ .
- 3 **for**  $(i, j, \llbracket b_{ij} \rrbracket) \in [E_i]^{(1,2)}, i \in [1, N]$  **do**
- 4      $\llbracket d_i^+ \rrbracket = \llbracket d_i^+ \rrbracket + \llbracket b_{ij} \rrbracket; \llbracket d_j^- \rrbracket = \llbracket d_j^- \rrbracket + \llbracket b_{ij} \rrbracket$ .
- 5      $\llbracket \text{isClip}_1 \rrbracket = \text{secComp}(\llbracket d_i^+ \rrbracket, k/3)$ . // Oblivious comparison.
- 6      $\llbracket \text{isClip}_2 \rrbracket = \text{secComp}(\llbracket d_j^- \rrbracket, k/3)$ . // Oblivious comparison.
- 7      $\llbracket \text{isClip} \rrbracket = \llbracket \text{isClip}_1 \rrbracket \cdot \llbracket \text{isClip}_2 \rrbracket$ . // Aggregate the results.
- 8      $\llbracket b'_{ij} \rrbracket = \llbracket b_{ij} \rrbracket \cdot \llbracket \text{isClip} \rrbracket$ . // Obviously clip the edge.
- 9      $[\bar{E}_i]^{(1,2)}.add((i, j, \llbracket b'_{ij} \rrbracket))$ . // Add the updated edge.
- 10  $P_{2,3}$  (resp.  $P_{3,1}$ ) perform the above operations on  $\{[E_i]^{(2,3)}\}$  (resp.  $\{[E_i]^{(3,1)}\}$ ) to produce  $\{[\bar{E}_i]^{(2,3)}\}$  (resp.  $\{[\bar{E}_i]^{(3,1)}\}$ ).
- 11 **return**  $\{[\bar{E}_i]\} = \{[\bar{E}_i]^{(1,2)} \cup [\bar{E}_i]^{(2,3)} \cup [\bar{E}_i]^{(3,1)}\}$ .

---

plaintext domain [3]. It projects an original graph with arbitrary degrees onto a graph where the maximum degree is  $k$ . Given a graph and a clipping parameter  $k$ , it iterates through each edge in a random order. If either of the two vertices connected by an edge currently has a degree larger than  $k$ , the edge is deleted.

However, the algorithm [3] focuses on undirected graphs, meaning it does not differentiate between out/in-degrees. We adapt it to accommodate directed graphs. Specifically, when considering an edge from vertex  $i$  to  $j$ , their degrees  $d_i^+, d_j^-$  are incremented to 1. Then, if  $d_i^+ > k$  or  $d_j^- > k$ , the edge is deleted. As stated in Section 4, each aggregator only holds a subset of each local neighbor list. Hence, they cannot simply clip the edges using the clipping parameter  $k$ . Our insight is to enable each pair of aggregators to *independently* perform oblivious edge clipping on their jointly held local neighbor lists via a clipping parameter of  $k/3$ . It ensures that the maximum out/in-degree in the subgraph held by each pair of aggregators is restricted to  $k/3$ , and the maximum out/in-degree is limited to  $k/3 + k/3 + k/3 = k$ . Algorithm 1 presents the protocol.

Note that  $\text{secComp}(\cdot, \cdot)$  (lines 5 and 6) is the oblivious comparison operation. Specifically,  $\text{secComp}(\llbracket \alpha \rrbracket, \beta)$  outputs  $\llbracket 1 \rrbracket$  if  $\alpha \leq \beta$  and  $\llbracket 0 \rrbracket$  if  $\alpha > \beta$ . We employ the efficient function secret sharing (FSS) based distributed comparison function (DCF) [4] to instantiate it. At line 7,  $\text{isClip} = 1$  means that  $\text{isClip}_1 = 1$  and  $\text{isClip}_2 = 1$ , indicating that  $d_i^+ \leq k/3$  and  $d_j^- \leq k/3$ , and thus the edge  $(i, j, \llbracket b_{ij} \rrbracket)$  does not need to be clipped.  $\text{isClip} = 0$  indicates that either  $\text{isClip}_1 = 0$  or  $\text{isClip}_2 = 0$ , meaning that either  $d_i^+ > k/3$  or  $d_j^- > k/3$ , and thus the edge  $(i, j, \llbracket b_{ij} \rrbracket)$  needs to be deleted. Finally,  $P_{1,2}$  execute the clipping operation (line 8). The design intuition is that if  $\text{isClip} = 0$ ,  $b'_{ij}$  is assigned 0, i.e., the edge is clipped; if  $\text{isClip} = 1$ ,  $b'_{ij}$  is assigned  $b_{ij}$ , i.e., the edge is unchanged.

**Efficient oblivious edge clipping.** As Algorithm 1 iterates through each edge *sequentially*, its communication rounds grow linearly with the number of edges. To address this issue, we propose an efficient protocol based on it, as given in Algorithm 2. The enhancement comes from the independence of the out-degree (resp. in-degree) of each vertex from the out-degree (resp. in-degree) of other vertices. Hence, Algorithm 2 enables the aggregators to simultaneously restrict each vertex's out/in-degree *in parallel* (lines 3 and 12). Another performance improvement from our proposed

**Algorithm 2: Efficient Oblivious Edge Clipping**


---

**Input:**  $\{[E_i]\}$ ; clipping parameters  $k$ ; batch size  $w$ .  
**Output:** Secret-shared clipped local neighbor lists  $\{[\bar{E}_i]\}$ .

- 1 //  $P_{1,2}$  operate on  $\{[E_i]^{(1,2)}\}$ ;
- 2 Initialization:  $\llbracket d_i^+ \rrbracket = \llbracket 0 \rrbracket$ ;  $\llbracket d_i^- \rrbracket = \llbracket 0 \rrbracket$ ;  $[\bar{E}_i]^{(1,2)} = \emptyset, i \in [1, N]$ .
- 3 Restrict the out-degree of vertex  $i \in [1, N]$  **in parallel**:
- 4 Initialize a queue  $Q_i = \emptyset$ .
- 5 **for**  $(i, j, \llbracket b_{ij} \rrbracket) \in [E_i]^{(1,2)}$  **do**
- 6      $\llbracket d_i^+ \rrbracket = \llbracket d_i^+ \rrbracket + \llbracket b_{ij} \rrbracket$ ;  $Q_i.\text{push}((i, j, \llbracket b_{ij} \rrbracket))$ .
- 7     **if** the size of the queue  $Q_i \geq w$  **then**
- 8          $\llbracket \text{isClip} \rrbracket = \text{secComp}(\llbracket d_i^+ \rrbracket, k/3)$ .
- 9         **while**  $Q_i \neq \emptyset$  **do**
- 10              $(i, j, \llbracket b_{ij} \rrbracket) = Q_i.\text{pop}()$ ;  $\llbracket b'_{ij} \rrbracket = \llbracket b_{ij} \rrbracket \cdot \llbracket \text{isClip} \rrbracket$ .
- 11              $[\bar{E}_i]^{(1,2)}.\text{add}((i, j, \llbracket b'_{ij} \rrbracket))$ .
- 12 Restrict the in-degrees on  $[\bar{E}_i]^{(1,2)}$  **in parallel** as above.
- 13  $P_{2,3}$  (resp.  $P_{3,1}$ ) perform the above operations on  $\{[E_i]^{(2,3)}\}$  (resp.  $\{[E_i]^{(3,1)}\}$ ) to produce  $\{[\bar{E}_i]^{(2,3)}\}$  (resp.  $\{[\bar{E}_i]^{(3,1)}\}$ ).
- 14 **return**  $\{[\bar{E}_i]\} = \{[\bar{E}_i]^{(1,2)} \cup [\bar{E}_i]^{(2,3)} \cup [\bar{E}_i]^{(3,1)}\}$ .

---

*batch edge clipping* technique. We observe that the true edges in the secret-shared local neighbor lists are sparse. This means that directly clipping the edges in a batch after a single check of the current degree (incremented by the edges in the batch), will seldom lead to clipping true edges. Hence, to reduce the number of required oblivious comparison operations, we modify the procedure to check whether the current secret-shared out-degree or in-degree is larger than  $k/3$  after iterating  $w$  edges (i.e., a batch), instead of performing the check after iterating each edge. The edges in each batch are updated without distinction based on the comparison result.

## 5.2 Secure Degree Sequence Extraction

**Design rationale.** Algorithm 3 presents how  $P_{1,2,3}$  securely extract the noisy version of the degree sequence  $\mathcal{D}^\pm$  from the secret-shared clipped local neighbor lists  $\{[\bar{E}_i]\}$ , while ensuring that the extracted noisy sequence  $\hat{\mathcal{D}}^\pm$  is differentially private. A naive method is to add secret-shared discrete Laplace noise to each secret-shared out/in-degree, followed by recovering them. However, this method introduces a significant amount of noise, particularly in low-degree vertices, which are abundant in real social graphs [22]. Our insight is to let  $P_{1,2,3}$  extract the differentially private *histogram* of (out-degree, in-degree) pairs and then derive  $\hat{\mathcal{D}}^\pm$  from the histogram.

**Secure degree histogram construction.**  $P_{1,2,3}$  first construct the histogram  $\llbracket \mathbf{H} \rrbracket$ . We observe that the primary challenge is to securely index  $(\llbracket d_i^+ \rrbracket, \llbracket d_i^- \rrbracket)$  in  $\llbracket \mathbf{H} \rrbracket$ . Our insight is to let  $P_{1,2}$  obliviously shuffle the secret-shared degree pairs by a random permutation unknown to them to break the mappings between the degree pairs and the vertices, and then securely reveal the shuffled degree pairs.

However, simply recovering the secret-shared degree pairs after the oblivious shuffle will directly expose  $\mathbf{H}$ . To address this, we let  $P_3$  secret-share a certain number of dummy entities  $d^+ \| d^- \| \rho$  (with flag  $\rho = 0$ ) for each possible degree pair  $(d^+, d^-)$  with  $P_{1,2}$  (lines 7-11).  $P_{1,2}$  then perform the oblivious row-wise shuffle [15] (denoted as  $\text{secShuffle}(\cdot)$ ) on the real degree pairs (with  $\llbracket \rho \rrbracket = \llbracket 1 \rrbracket$ ) together with the dummy degree pairs. The oblivious shuffle ensures that  $P_{1,2}$  cannot distinguish between the dummy and real degree pairs.

**Algorithm 3: Secure Degree Sequence Extraction**


---

**Input:**  $\{[\bar{E}_i]\}$ ;  $k$ ; privacy budget  $\epsilon_{\mathcal{D}}$ ; privacy parameter  $\delta_{\mathcal{D}}$ .  
**Output:** Differentially private degree sequence  $\hat{\mathcal{D}}^\pm$ .

- 1 //  $P_1, P_2$ , and  $P_3$  locally count the secret-shared degrees:
- 2 Initialization:  $\llbracket d_i^+ \rrbracket = \llbracket 0 \rrbracket$ ;  $\llbracket d_i^- \rrbracket = \llbracket 0 \rrbracket, i \in [1, N]$ .
- 3  $\llbracket d_i^+ \rrbracket = \llbracket d_i^+ \rrbracket + \llbracket b_{ij} \rrbracket, \llbracket d_i^- \rrbracket = \llbracket d_i^- \rrbracket + \llbracket b_{ij} \rrbracket, (i, j, \llbracket b_{ij} \rrbracket) \in \{[\bar{E}_i]\}$ .
- 4  $P_3$  secret-shares its shares of degrees with  $P_{1,2}$ .
- 5  $P_{1,2}$  add the received shares from  $P_3$  to their own shares to obtain  $\llbracket d_i^+ \rrbracket, \llbracket d_i^- \rrbracket, i \in [1, N]$  in 2-out-of-2 ASS.
- 6  $P_{1,2}$  arrange the secret-shared degrees into table  $\llbracket \mathbf{T} \rrbracket$ , where  $\llbracket \mathbf{T}[i, :] \rrbracket = \llbracket d_i^+ \rrbracket \| \llbracket d_i^- \rrbracket \| \llbracket \rho \rrbracket, i \in [1, N]$  and  $\rho = 1$ .
- 7 //  $P_3$  constructs dummy degree pair table  $\mathbf{T}'$ :  $\mathbf{T}' = []$  // Empty table.
- 8 **for**  $(d^+, d^-), d^+ \in [0, k], d^- \in [0, k]$  **do**
- 9      $n = \max(0, \text{Lap}(\epsilon_{\mathcal{D}}, \delta_{\mathcal{D}}, 4))$ . // Draw a noise.
- 10     Append  $n$  rows of  $d^+ \| d^- \| \rho$  to  $\mathbf{T}'$ , where  $\rho = 0$ .
- 11  $P_3$  secret-shares  $\mathbf{T}'$  with  $P_{1,2}$  to produce  $\llbracket \mathbf{T}' \rrbracket$ .
- 12 //  $P_{1,2}$  perform the remaining operations:
- 13  $\llbracket \hat{\mathbf{T}} \rrbracket = \frac{\llbracket \mathbf{T} \rrbracket}{\llbracket \mathbf{T}' \rrbracket}$ . // Vertical concatenation.
- 14  $\llbracket \hat{\mathbf{T}}' \rrbracket = \text{secShuffle}(\llbracket \hat{\mathbf{T}} \rrbracket)$ . // Oblivious row-wise shuffle.
- 15 Safely reveal columns 1 and 2 of  $\llbracket \hat{\mathbf{T}}' \rrbracket$  to obtain  $\llbracket \hat{\mathbf{T}}' \rrbracket$ .
- 16 Initialize degree histogram  $\llbracket \mathbf{H} \rrbracket = \llbracket 0 \rrbracket^{(k+1) \times (k+1)}$ .
- 17 **for**  $d^+ \| d^- \| \llbracket \rho \rrbracket$  in  $\llbracket \hat{\mathbf{T}}' \rrbracket$  **do**
- 18      $\llbracket \mathbf{H}[d^+, d^-] \rrbracket = \llbracket \mathbf{H}[d^+, d^-] \rrbracket + \llbracket \rho \rrbracket$ . // Index edge to histogram.
- 19  $\llbracket \hat{\mathbf{H}}[d^+, d^-] \rrbracket = \llbracket \mathbf{H}[d^+, d^-] \rrbracket + \llbracket \text{Lap}(\epsilon_{\mathcal{D}}, 0, 4) \rrbracket, d^+, d^- \in [0, k]$ .
- 20 Safely reveal the differentially private histogram  $\hat{\mathbf{H}}$ .
- 21  $\hat{\mathbf{H}}'[d^+, d^-] = \max(0, \hat{\mathbf{H}}[d^+, d^-])$ ,  $d^+, d^- \in [0, k]$ .
- 22  $\hat{\mathbf{H}}''[d^+, d^-] = \hat{\mathbf{H}}'[d^+, d^-] \cdot \frac{N}{\text{sum}(\hat{\mathbf{H}}')}, d^+, d^- \in [0, k]$ .
- 23 Initialize degree sequence  $\hat{\mathcal{D}}^\pm = \emptyset$ .
- 24  $\forall d^+, d^- \in [0, k]$ , add  $\hat{\mathbf{H}}''[d^+, d^-]$  tuples  $(d^+, d^-)$  to  $\hat{\mathcal{D}}^\pm$ .
- 25 **return** Differentially private degree sequence  $\hat{\mathcal{D}}^\pm$ .

---

To ensure the revealed frequency of degree pairs in the views of  $P_{1,2}$  is differentially private, the number of dummy entities for each degree pair is drawn from  $\text{Lap}(\epsilon_{\mathcal{D}}, \delta_{\mathcal{D}}, 4)$  and truncated to 0 (line 9). Here,  $\Delta$  is set to 4, as the addition/removal of an edge changes the frequency of at most four degree pairs. We will prove in Section 7.2 that the revealed degree pairs are differentially private in the individual view of  $P_1, P_2$ , even if the noises may be truncated to 0. Note that we cannot use the revealed frequency of degree pairs as  $\mathbf{H}$  due to  $P_3$  knowing the number of dummy entities, i.e., the frequency is not differentially private in  $P_3$ 's view. Instead,  $P_{1,2}$  add the secret-shared flag of each degree pair to the corresponding element of  $\llbracket \mathbf{H} \rrbracket$  (lines 16-18).

**Secure degree sequence construction.** To construct  $\hat{\mathcal{D}}^\pm$  based on  $\llbracket \mathbf{H} \rrbracket$ ,  $P_{1,2}$  first add a secret-shared noise drawn from  $\text{Lap}(\epsilon_{\mathcal{D}}, 0, 4)$  to each element of  $\llbracket \mathbf{H} \rrbracket$  (line 19). The noises can be generated offline by the protocol in [14].  $P_{1,2}$  reveal the differentially private histogram  $\llbracket \hat{\mathbf{H}} \rrbracket$  to obtain  $\hat{\mathbf{H}}$ .  $\hat{\mathcal{D}}^\pm$  can be constructed based on  $\hat{\mathbf{H}}$  (lines 21-24).

## 5.3 Secure Attribute-Edge Correlation Distribution Extraction

$P_{1,2,3}$  then securely extract the noisy version of attribute-edge correlation distribution  $\Theta_{\bar{X}}$ . PrivAGM ensures that the extracted distribution (denoted as  $\tilde{\Theta}_{\bar{X}}$ ) is differentially private. Algorithm 4 gives



**Algorithm 4:** Secure Attribute-Edge Distribution Extraction

**Input:** Local neighbor lists  $\{\{\bar{E}_i\}\}$ ; secret-shared attributes  $\{\{f_i\}\}$ ; maximum degree  $k$ ; privacy budget, parameter  $\epsilon_X, \delta_X$ .

**Output:** Differentially private attribute-edge distribution  $\tilde{\Theta}_{\bar{X}}$ .

- 1 //  $P_{1,2}$  operate on  $\{\{\bar{E}_i\}^{(1,2)}\}$  and  $\{\{f_i\}\}$ :  $\llbracket \mathbf{T} \rrbracket = []$ ;  $\mathbf{T}' = []$ .
- 2 **for**  $(i, j, \llbracket b_{ij} \rrbracket) \in \{\{\bar{E}_i\}^{(1,2)}\}, i \in [1, N]$  **do**
- 3   Append row  $\llbracket f_i \rrbracket \parallel \llbracket f_j \rrbracket \parallel \llbracket \rho \rrbracket$  to  $\llbracket \mathbf{T} \rrbracket$ , where  $\llbracket \rho \rrbracket = \llbracket b_{ij} \rrbracket$ .
- 4 //  $P_3$  constructs dummy attribute-edge correlation table  $\mathbf{T}'$ :
- 5 **for**  $f \rightarrow f' \in \mathbb{Z}_{2^L} \times \mathbb{Z}_{2^L}$  **do**
- 6    $n = \max(0, \text{Lap}(\epsilon_X, \delta_X, 2 \cdot k/3))$ . // Draw a noise.
- 7   Append  $n$  rows of  $f \parallel f' \parallel \rho$  to  $\mathbf{T}'$ , where  $\rho = 0$ .
- 8  $P_3$  secret-shares  $\mathbf{T}'$  with  $P_{1,2}$  to produce  $\llbracket \mathbf{T}' \rrbracket$ .
- 9 //  $P_{1,2}$  perform the remaining operations:
- 10  $\llbracket \hat{\mathbf{T}} \rrbracket = \llbracket \mathbf{T} \rrbracket \parallel \llbracket \mathbf{T}' \rrbracket$ . // Vertical concatenation.
- 11  $\llbracket \hat{\mathbf{T}}' \rrbracket = \text{secShuffle}(\llbracket \hat{\mathbf{T}} \rrbracket)$ . // Oblivious row-wise shuffle
- 12 Safely reveal columns 1 and 2 of  $\llbracket \hat{\mathbf{T}}' \rrbracket$  to obtain  $\llbracket \hat{\mathbf{T}}' \rrbracket$ .
- 13 Initialize secret-shared histogram  $\llbracket \mathbf{H} \rrbracket = \llbracket 0 \rrbracket^{2^L \times 2^L}$ .
- 14 **for row**  $f \parallel f' \parallel \llbracket \rho \rrbracket$  **in**  $\llbracket \hat{\mathbf{T}}' \rrbracket$  **do**
- 15    $\llbracket \mathbf{H}[f, f'] \rrbracket = \llbracket \mathbf{H}[f, f'] \rrbracket + \llbracket \rho \rrbracket$ . // Index correlation to  $\llbracket \mathbf{H} \rrbracket$ .
- 16  $\llbracket \tilde{\mathbf{H}}[f, f'] \rrbracket = \llbracket \text{Lap}(\epsilon_X, \delta_X, 0, 2 \cdot k/3) \rrbracket, f \rightarrow f' \in \mathbb{Z}_{2^L} \times \mathbb{Z}_{2^L}$ .
- 17 Safely reveal the differentially private histogram  $\tilde{\mathbf{H}}$ .
- 18  $\tilde{\mathbf{H}}'[f, f'] = \max(0, \tilde{\mathbf{H}}[f, f'])$ ,  $f \rightarrow f' \in \mathbb{Z}_{2^L} \times \mathbb{Z}_{2^L}$ . // Truncation.
- 19  $\tilde{\mathbf{H}}''[f, f'] = \frac{\tilde{\mathbf{H}}'[f, f']}{\text{sum}(\tilde{\mathbf{H}}')}$ ,  $f \rightarrow f' \in \mathbb{Z}_{2^L} \times \mathbb{Z}_{2^L}$ . // Normalization.
- 20 Initialize attribute-edge distribution  $\tilde{\Theta}_{\bar{X}} = \{0\}^{2^L \times 2^L}$ .
- 21  $\tilde{\Theta}_{\bar{X}}(f \rightarrow f') = \tilde{\mathbf{H}}''[f, f']$ ,  $f \rightarrow f' \in \mathbb{Z}_{2^L} \times \mathbb{Z}_{2^L}$ .
- 22 **return** Differentially private attribute-edge distribution  $\tilde{\Theta}_{\bar{X}}$ .

the protocol. Its design intuition shares similarities with Algorithm 3. Since the edges held by each aggregator are incomplete, we let a pair of aggregators to compute  $\tilde{\Theta}_{\bar{X}}$  on their jointly held local neighbor lists. The sensitivity  $\Delta$  is set to  $2 \cdot k/3$  (lines 6 and 15), as modifying the attribute value of a vertex can affect, at most,  $k/3$  edges, resulting in changes to the frequency of  $2 \cdot k/3$  attribute-edge correlations. Appendix A of the full version [45] details how to extract the differentially private attribute distribution (denoted as  $\tilde{\Theta}_F$ ). The privacy budget and parameter are  $\epsilon_F$  and  $\delta_F$ , respectively.

## 5.4 Secure Triangle Counts Extraction

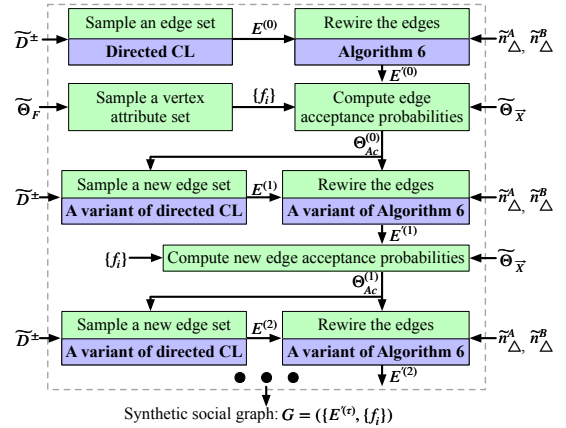
Algorithm 5 presents how to extract the noisy version of triangle counts  $n_{\Delta}^A, n_{\Delta}^B$ , while ensuring that the extracted noisy counts (denoted as  $\tilde{n}_{\Delta}^A, \tilde{n}_{\Delta}^B$ ) are differentially private. The idea is to iterate over every possible set of three connected edges in pairs, incrementing the secret-shared product of flags for the three edges to the triangle count. However, since multiplication in the secret sharing domain requires communication between the involved parties, the method can only count triangles whose edges are held by the *same* pair of aggregators. To address this issue, our solution is based on the observation: the probability of each triangle's edges being distributed to the same pair of aggregators is  $3 \cdot \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{1}{3} = \frac{1}{9}$ . Hence, PrivAGM enables each pair of aggregators to count triangles over the edges they jointly hold. The counts are then multiplied by 9 to obtain approximate triangle counts. The sensitivity is set to  $k-1$  as adding or removing an edge change the triangle count by at most  $k-1$ .

**Algorithm 5:** Secure Triangle Counts Extraction

**Input:**  $\{\{\bar{E}_i\}\}$ ; privacy budget  $\epsilon_T$ ; maximum in/out-degree  $k$ .

**Output:** Differentially private triangle counts  $\tilde{n}_{\Delta}^A$  and  $\tilde{n}_{\Delta}^B$ .

- 1 Initialize  $\llbracket n_{\Delta}^A \rrbracket = \llbracket 0 \rrbracket$ ;  $\llbracket n_{\Delta}^B \rrbracket = \llbracket 0 \rrbracket$  in 3-out-of-3 ASS.
- 2 //  $P_{1,2}$  operate on  $\{\{\bar{E}_i\}^{(1,2)}\}$ :
- 3 **for**  $i : i \in [1, N]$  **in parallel do**
- 4   **for**  $j : j < i$  and  $(i, j, \llbracket b_{ij} \rrbracket) \in \{\{\bar{E}_i\}^{(1,2)}\}$  **do**
- 5     **for**  $k : k < i$  and  $(j, k, \llbracket b_{jk} \rrbracket) \in \{\{\bar{E}_j\}^{(1,2)}\}$  **do**
- 6       **if**  $(k, i, \llbracket b_{ki} \rrbracket) \in \{\{\bar{E}_k\}^{(1,2)}\}$  **then**
- 7          $\llbracket n_{\Delta}^A \rrbracket = \llbracket n_{\Delta}^A \rrbracket + \llbracket b_{ij} \rrbracket \cdot \llbracket b_{jk} \rrbracket \cdot \llbracket b_{ki} \rrbracket$ .
- 8 **for**  $i : i \in [1, N]$  **in parallel do**
- 9   **for**  $j, k : j \neq k, (i, j, \llbracket b_{ij} \rrbracket), (i, k, \llbracket b_{ik} \rrbracket) \in \{\{\bar{E}_i\}^{(1,2)}\}$  **do**
- 10     **if**  $(j, k, \llbracket b_{jk} \rrbracket) \in \{\{\bar{E}_j\}^{(1,2)}\}$  **then**
- 11        $\llbracket n_{\Delta}^B \rrbracket = \llbracket n_{\Delta}^B \rrbracket + \llbracket b_{ij} \rrbracket \cdot \llbracket b_{ik} \rrbracket \cdot \llbracket b_{jk} \rrbracket$ .
- 12  $P_{2,3}$  (resp.  $P_{3,1}$ ) count triangles on  $\{\{\bar{E}_i\}^{(2,3)}\}$  (resp.  $\{\{\bar{E}_i\}^{(3,1)}\}$ ) as above and aggregate the triangle counts to  $\llbracket n_{\Delta}^A \rrbracket$  and  $\llbracket n_{\Delta}^B \rrbracket$ .
- 13  $\llbracket n_{\Delta}^A \rrbracket = 9 \cdot \llbracket n_{\Delta}^A \rrbracket$ ;  $\llbracket n_{\Delta}^B \rrbracket = 9 \cdot \llbracket n_{\Delta}^B \rrbracket$ .
- 14  $\llbracket \tilde{n}_{\Delta}^A \rrbracket = \llbracket n_{\Delta}^A \rrbracket + \llbracket \text{Lap}(\epsilon_T, 0, k-1) \rrbracket$ . // Add secret-shared noise.
- 15  $\llbracket \tilde{n}_{\Delta}^B \rrbracket = \llbracket n_{\Delta}^B \rrbracket + \llbracket \text{Lap}(\epsilon_T, 0, k-1) \rrbracket$ . // Add secret-shared noise.
- 16 Safely reveal the differentially private triangle counts  $\tilde{n}_{\Delta}^A, \tilde{n}_{\Delta}^B$ .
- 17 **return** Differentially private triangle counts  $\tilde{n}_{\Delta}^A$  and  $\tilde{n}_{\Delta}^B$ .



**Figure 3: Overview of directed attributed graphs generation.**

**Remark.** We note that performing Algorithm 5 sequentially would entail a significant number of communication rounds. As the iterations in the two outermost loops (i.e., lines 3 and 8) are independent of each other, we can execute them in parallel. Moreover, we observe that the primary cost arises from the secure 3-input multiplication gates in lines 7 and 11. To speed up Algorithm 5, we use the technique in [32] to reduce the communication cost of each secure 3-input multiplication gate from 2 rounds and 8 elements to 1 round and 6 elements. Additionally, we use a random sampling strategy to further reduce the cost. Specifically, instead of iterating over each vertex, we only iterate over  $\eta$  randomly sampled vertices, and then multiply the secret-shared triangle counts by  $\frac{N}{\eta}$ .

## 6 ATTRIBUTED GRAPHS GENERATION

Hence, our contribution is to adapt it to cater for the requirements of generating directed graphs.

**Overview.** Fig. 3 gives an overview. We note that the CL model is a widely used generative graph model. However, it is designed

exclusively for undirected graphs. Therefore, we propose a directed version of it, which is used to sample an initial *directed* edge set  $E^{(0)} = \{(i, j)\}$  based on  $\tilde{\mathcal{D}}^\pm$ . Next, Algorithm 6 is used to rewire the edges to ensure that the counts of TriA and TriB formed by  $E^{(0)}$  are no less than  $\tilde{n}_\Delta^A$  and  $\tilde{n}_\Delta^B$ , respectively. The rewired edges are denoted as  $E'^{(0)}$ . In addition, the vertex attribute set  $\{f_i\}_{i \in [1, N]}$  (where  $f_i$  for vertex  $i \in [1, N]$ ) is sampled from  $\tilde{\Theta}_F$ . The accept/reject sampling technique is applied to generate new edge sets. Specifically, the edge acceptance probabilities  $\Theta_{Ac}^{(0)}$  are computed based on the current graph  $(E'^{(0)}, \{f_i\})$  and  $\tilde{\Theta}_X$ . Then a new edge set  $E^{(1)}$  is sampled by the directed CL model and a variant of Algorithm 6 considering  $\Theta_{Ac}^{(0)}$ . This process iterates until  $\Theta_{Ac}$  converges.

### 6.1 Directed CL Model

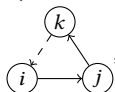
In the CL model [7], each vertex is assigned a degree based on the degree sequence  $\{d_i\}_{i \in [1, N]}$  of the input graph. Then,  $\sum_{i \in [1, N]} d_i$  edges are sampled by a probability that is proportional to the desired degrees of the vertices. Specifically, an edge between vertices  $i, j$  is sampled with a probability of  $\frac{d_i \cdot d_j}{\sum_{i \neq j, i, j \in [1, N]} d_i \cdot d_j}$ . However, this cannot capture the directionality of edges. Hence, we propose the *directed CL model* (denoted as directedCL( $\cdot$ )).

In our directed CL model, we first assign each vertex  $i$  a desired out/in-degree pair  $(d_i^+, d_i^-)$  based on  $\tilde{\mathcal{D}}^\pm$ . Then, for each directed edge from  $i$  to  $j$ , we add it into the edge set  $E^{(0)}$  with a probability proportional to  $i$ 's out-degree and  $j$ 's in-degree, i.e.,  $\frac{d_i^+ \cdot d_j^-}{\sum_{i \neq j, i, j \in [1, N]} d_i^+ \cdot d_j^-}$ . To maintain the expected degree distribution, we sample  $\max(\sum_{i \in [1, N]} d_i^+, \sum_{i \in [1, N]} d_i^-)$  edges. Clearly, our model can capture the edge directionality in the original graph by sampling each edge based on the out-degree of the edge's starting vertex and the in-degree of the edge's ending vertex.

### 6.2 Refining Graph Structure

The (unattributed) graphs generated by the directed CL model only capture the degree information of the original graph. Therefore, the AGM [22] proposes iterative edge rewiring to ensure that the number of triangles in the generated graph is no smaller than that in the original graph. However, the AGM overlooks the directionality of edges in the original graph and treats all triangles as identical. Therefore, we propose Algorithm 6 (denoted as Rewire( $\cdot$ )).

Algorithm 6 operates on the edge set  $E$  generated by the directed CL model. It iteratively rewires the edges until the counts of TriA and TriB formed by the edges surpass  $\tilde{n}_\Delta^A$  and  $\tilde{n}_\Delta^B$ , respectively. Specifically, to create a new edge for constructing TriA (lines 4-12), the algorithm proceeds as follows: (1) sample a vertex  $i$  from  $[1, N]$  where each vertex  $i \in [1, N]$  is sampled with a probability of  $\frac{d_i^+}{\sum_{i \in [1, N]} d_i^+}$ ; (2) randomly sample a vertex  $j$  from vertex  $i$ 's out-neighbors; (3) randomly sample a vertex  $k$  from vertex  $j$ 's out-neighbors. If  $(k, i) \notin E$ , indicating that vertices  $i, j$ , and  $k$  form an

incomplete TriA, i.e., , the algorithm attempts to replace

the oldest edge in  $E$  by edge  $(k, i)$  to construct TriA (lines 7-12). Specifically, if replacing the oldest edge  $(q, r)$  in  $E$  with edge  $(k, i)$  would increase both the counts of TriA and TriB, edge  $(q, r)$  is

#### Algorithm 6: Refining Graph Structure by Edge Rewiring

---

**Input:** Edge set  $E$ ; degree sequence  $\tilde{\mathcal{D}}^\pm$ ; triangle counts  $\tilde{n}_\Delta^A, \tilde{n}_\Delta^B$ .  
**Output:** The new edge set  $E'$  after rewiring.

```

1  $n_\Delta^A, n_\Delta^B = \text{countTriangle}(E)$ . // Count triangles on  $E$ .
2 while  $n_\Delta^A < \tilde{n}_\Delta^A$  and  $n_\Delta^B < \tilde{n}_\Delta^B$  do
3   // Construct TriA:
4    $i = \text{Sample}([1, N], \tilde{\mathcal{D}}^\pm)$ . // Sample a vertex based on  $\tilde{\mathcal{D}}^\pm$ .
5    $j = \text{SampleNbr}(E, i)$ ;  $k = \text{SampleNbr}(E, j)$ .
6   if  $(k, i) \notin E$  then
7      $(q, r) = \text{oldestEdge}(E)$ . // Get the oldest edge from  $E$ .
8     if replacing  $(q, r)$  by  $(k, i)$  boosts triangle counts then
9        $E.\text{replace}((q, r), (k, i))$ . // Replace  $(q, r)$  by  $(k, i)$ .
10      Make edge  $(k, i)$  the youngest edge in  $E$ .
11       $n_\Delta^A, n_\Delta^B = \text{countTriangle}(E)$ .
12    else Make edge  $(q, r)$  the youngest edge in  $E$ .
13   // Construct TriB:
14    $i = \text{Sample}([1, N], \tilde{\mathcal{D}}^\pm)$ ;  $j, k = \text{Sample2Nbr}(E, i)$ .
15   If  $(j, k) \notin E$ , replacing the oldest edge by  $(j, k)$  as above.
16 return The edge set  $E' = E$  after rewiring.
```

---

replaced by edge  $(k, i)$  and  $(k, i)$  is set as the youngest edge in  $E$ . Otherwise, edge  $(q, r)$  is set as the youngest edge in  $E$ .

Similarly, Algorithm 6 creates a new edge to construct TriB as follows (lines 14-15). First, a vertex  $i$  is sampled from  $[1, N]$  where each vertex  $i \in [1, N]$  is chosen with a probability of  $\frac{d_i^+}{\sum_{i \in [1, N]} d_i^+}$ . Then, two vertices  $j$  and  $k$  are randomly sampled from vertex  $i$ 's out-neighbors. If  $(j, k) \notin E$ , indicating that vertices  $i, j$ , and  $k$

form an incomplete TriB, i.e., , the algorithm checks if

replacing the oldest edge  $(q, r)$  in  $E$  with edge  $(j, k)$  would increase both the counts of TriA and TriB. If it does, the algorithm replaces edge  $(q, r)$  with edge  $(j, k)$  and makes  $(j, k)$  the youngest edge in  $E$ . Otherwise, it makes edge  $(q, r)$  the youngest edge in  $E$ .

We present how to integrate the above algorithms in Appendix B of the full version [45].

## 7 PRIVACY AND SECURITY ANALYSIS

In Section 7.1, we parametrize the leakage function  $\mathcal{L}$ . In Section 7.2, we conduct a formal analysis to prove that the output of  $\mathcal{L}$  is differentially private. In Section 7.3, we prove that the adversary's view can be simulated solely based on the output of  $\mathcal{L}$ .

### 7.1 Leakage Function

We define the leakage function as  $\mathcal{L} = (\mathcal{L}_E, \mathcal{L}_\mathcal{D}, \mathcal{L}_X, \mathcal{L}_F, \mathcal{L}_\Delta)$ :

- $\mathcal{L}_E = \{[E_i]\}$ , where  $[E_i] = \{(i, j, \langle b_{ij} \rangle)\}$  denotes the local neighbor list held by the adversary for user  $i$ .
- $\mathcal{L}_\mathcal{D} = (\tilde{\mathbf{T}}'_\mathcal{D}, \tilde{\mathbf{H}}_\mathcal{D})$ , where  $[\tilde{\mathbf{T}}'_\mathcal{D}]$  and  $\tilde{\mathbf{H}}_\mathcal{D}$  are the table and histogram revealed in lines 15 and 20 of Algorithm 3, respectively. While the adversary learns additional information during the execution of other operations in Algorithm 3, since the information is derived from  $\tilde{\mathbf{H}}_\mathcal{D}$ , it is not included in  $\mathcal{L}_\mathcal{D}$ .
- $\mathcal{L}_X = (\tilde{\mathbf{T}}'_X, \tilde{\mathbf{H}}_X)$ , where  $[\tilde{\mathbf{T}}'_X]$ ,  $\tilde{\mathbf{H}}_X$  are the table and histogram revealed in lines 12, 17 of Algorithm 4, respectively.



- $\mathcal{L}_F = ([\hat{\mathbf{T}}'_F], \tilde{\mathbf{H}}_F)$ , where  $[\hat{\mathbf{T}}'_F]$  and  $\tilde{\mathbf{H}}_F$  are the table and histogram revealed in secure attribute distribution extraction.
- $\mathcal{L}_\Delta = (\tilde{n}_\Delta^A, \tilde{n}_\Delta^B)$ :  $\tilde{n}_\Delta^A, \tilde{n}_\Delta^B$  are the counts revealed in line 16, Algo. 5.

## 7.2 Privacy Analysis

**THEOREM 1.** *The neighbor list  $[E_i]$  of user  $i$  held by the adversary satisfies pure  $\varepsilon_E$ -edge LDP with  $\varepsilon_E = \max\{\ln(1/p), \ln(3-2p)\}$ .*

PROOF. Note that  $[E_i]$  can be described as an  $N$ -dimensional bit vector  $\mathbf{b} = (b_1, \dots, b_N)$  for the adversary, where  $b_j = 1$  if  $(i, j, \langle b_{ij} \rangle) \in [E_i]$ , and otherwise  $b_j = 0$ . We use  $Pr[\mathbf{A}[i, j] \rightarrow b_j]$  to denote the probability that, given  $\mathbf{A}[i, j]$ , the adversary observes  $b_j \in \{0, 1\}$ .  $Pr[\mathbf{A}[i, j] \rightarrow b_j]$  can be categorized into four cases:

- $\mathbf{A}[i, j] = 0$  and  $b_j = 0$ : This case corresponds to the scenario where either user  $i$  does not include the zero bit  $\mathbf{A}[i, j]$  in  $E_i$  (with a probability of  $1-p$ ), or user  $i$  includes  $\mathbf{A}[i, j]$  in  $E_i$  (with a probability of  $p$ ), but does not choose the adversary to receive  $(i, j, \langle 0 \rangle)$  (with a probability of  $1/3$ ). Therefore,  $Pr[0 \rightarrow 0] = 1-p+p \cdot 1/3 = 1-2p/3$ .
- $\mathbf{A}[i, j] = 1$  and  $b_j = 0$ : This case corresponds to the scenario where user  $i$  includes the nonzero bit  $\mathbf{A}[i, j]$  in  $E_i$  (with a probability of 1), but does not choose the adversary to receive  $(i, j, \langle 1 \rangle)$  (with a probability of  $1/3$ ). Therefore,  $Pr[1 \rightarrow 0] = 1 \cdot 1/3 = 1/3$ .
- $\mathbf{A}[i, j] = 1$  and  $b_j = 1$ : This case corresponds to the scenario where user  $i$  includes the nonzero bit  $\mathbf{A}[i, j]$  in  $E_i$  (with a probability of 1) and also chooses the adversary to receive  $(i, j, \langle 1 \rangle)$  (with a probability of  $2/3$ ). Therefore,  $Pr[1 \rightarrow 1] = 1 \cdot 2/3 = 2/3$ .
- $\mathbf{A}[i, j] = 0$  and  $b_j = 1$ : This case corresponds to the scenario where user  $i$  includes the zero bit  $\mathbf{A}[i, j]$  in  $E_i$  (with a probability of  $p$ ) and also chooses the adversary to receive  $(i, j, \langle 0 \rangle)$  (with a probability of  $2/3$ ). Therefore,  $Pr[0 \rightarrow 1] = p \cdot 2/3 = 2p/3$ .

Given any two local neighbor lists  $\mathbf{A}[i, :]$  and  $\mathbf{A}'[i, :]$  that differ in one bit, we need to prove that  $\frac{Pr[\mathcal{M}(\mathbf{A}[i, :])=\mathbf{b}]}{Pr[\mathcal{M}(\mathbf{A}'[i, :])=\mathbf{b}]} \leq e^{\varepsilon_E}$ , where  $\varepsilon_E = \max\{\ln(1/p), \ln(3-2p)\}$ . Without loss of generality, let us assume that  $\mathbf{A}[i, 1]$  and  $\mathbf{A}'[i, 1]$  are not identical. Then we have:

$$\begin{aligned} \frac{Pr[\mathcal{M}(\mathbf{A}[i, :])=\mathbf{b}]}{Pr[\mathcal{M}(\mathbf{A}'[i, :])=\mathbf{b}]} &= \frac{Pr[\mathbf{A}[i, 1] \rightarrow b_1] \cdots Pr[\mathbf{A}[i, N] \rightarrow b_N]}{Pr[\mathbf{A}'[i, 1] \rightarrow b_1] \cdots Pr[\mathbf{A}'[i, N] \rightarrow b_N]} \\ &= \frac{Pr[\mathbf{A}[i, 1] \rightarrow b_1]}{Pr[\mathbf{A}'[i, 1] \rightarrow b_1]}. \end{aligned} \quad (3)$$

We analyze Eq. 3 in four cases:

- 1)  $\frac{Pr[0 \rightarrow 0]}{Pr[1 \rightarrow 0]} = \frac{1-2p/3}{1/3} = 3-2p$ ; 2)  $\frac{Pr[1 \rightarrow 0]}{Pr[0 \rightarrow 0]} = \frac{1/3}{1-2p/3} = \frac{1}{3-2p}$ ;
- 3)  $\frac{Pr[0 \rightarrow 1]}{Pr[1 \rightarrow 1]} = \frac{2p/3}{2/3} = p$ ; 4)  $\frac{Pr[1 \rightarrow 1]}{Pr[0 \rightarrow 1]} = \frac{2/3}{2p/3} = 1/p$ .

We note that the upper bound of the set  $\{3-2p, \frac{1}{3-2p}, p, 1/p\}$ . Since  $0 \leq p \leq 1$ , we have  $1/p \geq p$  and  $3-2p \geq \frac{1}{3-2p}$ . Therefore,  $\{3-2p, \frac{1}{3-2p}, p, 1/p\} \leq \max\{1/p, 3-2p\} = e^{\varepsilon_E}$ , and  $\varepsilon_E = \max\{\ln(1/p), \ln(3-2p)\}$ .  $\square$

**THEOREM 2.** *Given  $\mathcal{L}_\mathcal{D}$ , in the view of adversary,  $[\hat{\mathbf{T}}'_\mathcal{D}]$  adheres to  $(\varepsilon_\mathcal{D}, \delta_\mathcal{D})$ -edge DP and  $\tilde{\mathbf{H}}_\mathcal{D}$  adheres to pure  $\varepsilon_\mathcal{D}$ -edge DP.*

PROOF. Note that only  $P_{1,2}$  have access to  $[\hat{\mathbf{T}}'_\mathcal{D}]$ , and thus we only analyze the view of the adversary controlling  $P_1$  or  $P_2$ . Since the shares (i.e., the 3rd column of table  $[\hat{\mathbf{T}}'_\mathcal{D}]$ ) reveal nothing about the corresponding values,  $[\hat{\mathbf{T}}'_\mathcal{D}]$  can be described as a histogram (denoted as  $\mathbf{H}_\mathcal{D}$ ) of degree pairs. We then prove that with probability  $1 - \delta_\mathcal{D}$ , the probability for the adversary viewing the same

$\mathbf{H}_\mathcal{D}$  from two neighboring graphs  $G$  and  $G'$  is bounded by  $e^{\varepsilon_\mathcal{D}}$ . If all the noise (drawn in line 9 of Algorithm 3) are non-negative, the probability to output the same noisy histogram  $\mathbf{H}_\mathcal{D}$  from  $G, G'$  is

$$\frac{Pr[\mathcal{M}(G)=\mathbf{H}_\mathcal{D}]}{Pr[\mathcal{M}(G')=\mathbf{H}_\mathcal{D}]} = \frac{\prod_{d^+, d^- \in [0, k]} Pr[\mathbf{H}_\mathcal{D}[d^+, d^-] - \mathbf{H}_G[d^+, d^-]]}{\prod_{d^+, d^- \in [0, k]} Pr[\mathbf{H}_\mathcal{D}[d^+, d^-] - \mathbf{H}_{G'}[d^+, d^-]]}. \quad (4)$$

$Pr[\mathbf{H}_\mathcal{D}[d^+, d^-] - \mathbf{H}_G[d^+, d^-]]$  is the probability drawing the noise  $\mathbf{H}_\mathcal{D}[d^+, d^-] - \mathbf{H}_G[d^+, d^-]$  from  $Lap(\varepsilon_\mathcal{D}, \delta_\mathcal{D}, 4)$ . Let  $D_\Delta$  be the set of bins where  $\mathbf{H}_G \neq \mathbf{H}_{G'}$ . We can rewrite Eq. 4 as:

$$\begin{aligned} &\frac{\prod_{(d^+, d^-) \in D_\Delta} e^{-\frac{\varepsilon_\mathcal{D}}{4} \cdot |\mathbf{H}_\mathcal{D}[d^+, d^-] - \mathbf{H}_G[d^+, d^-] - \mu|}}{\prod_{(d^+, d^-) \in D_\Delta} e^{-\frac{\varepsilon_\mathcal{D}}{4} \cdot |\mathbf{H}_\mathcal{D}[d^+, d^-] - \mathbf{H}_{G'}[d^+, d^-] - \mu|}} \\ &= \prod_{(d^+, d^-) \in D_\Delta} e^{\frac{\varepsilon_\mathcal{D}}{4} \cdot (|\mathbf{H}_\mathcal{D}[d^+, d^-] - \mathbf{H}_{G'}[d^+, d^-] - \mu| - |\mathbf{H}_\mathcal{D}[d^+, d^-] - \mathbf{H}_G[d^+, d^-] - \mu|)} \\ &= e^{\frac{\varepsilon_\mathcal{D}}{4} \cdot \sum_{(d^+, d^-) \in D_\Delta} (|\mathbf{H}_\mathcal{D}[d^+, d^-] - \mathbf{H}_{G'}[d^+, d^-] - \mu| - |\mathbf{H}_\mathcal{D}[d^+, d^-] - \mathbf{H}_G[d^+, d^-] - \mu|)} \\ &\leq e^{\frac{\varepsilon_\mathcal{D}}{4} \cdot \sum_{(d^+, d^-) \in D_\Delta} |\mathbf{H}_G[d^+, d^-] - \mathbf{H}_{G'}[d^+, d^-]|} \leq e^{\frac{\varepsilon_\mathcal{D}}{4} \cdot 4} = e^{\varepsilon_\mathcal{D}}. \end{aligned} \quad (5)$$

The proof is based on: as  $G, G'$  differ by the presence/absence of an edge, the maximum discrepancy in the values of their degree pair histograms is 4, i.e.,  $\sum_{(d^+, d^-) \in D_\Delta} |\mathbf{H}_G[d^+, d^-] - \mathbf{H}_{G'}[d^+, d^-]| \leq 4$ .

The probability to draw a negative noise from  $Lap(\varepsilon_\mathcal{D}, \delta_\mathcal{D}, 4)$  is [17]:  $Pr[n < 0] = \sum_{x=-1}^{-\infty} \frac{e^{\frac{\varepsilon_\mathcal{D}}{4} \cdot x - 1}}{e^{\frac{\varepsilon_\mathcal{D}}{4} \cdot x + 1}} \cdot e^{-\frac{\varepsilon_\mathcal{D}}{4} \cdot |x - \mu|} = \frac{e^{-\frac{\mu \cdot \varepsilon_\mathcal{D}}{4}}}{e^{\frac{\varepsilon_\mathcal{D}}{4} \cdot 1 + 1}}$ . Given  $\mu = -\frac{4}{\varepsilon_\mathcal{D}} \cdot \ln((\frac{\varepsilon_\mathcal{D}}{4} + 1) \cdot (1 - (1 - \delta_\mathcal{D})^{\frac{1}{4}}))$ , we have  $Pr[n < 0] = 1 - (1 - \delta_\mathcal{D})^{\frac{1}{4}}$ . As there are at most 4 bins in  $\mathbf{H}_G, \mathbf{H}_{G'}$  that are not equal. Hence, the overall failing probability is  $1 - (1 - Pr[n < 0])^4 = \delta_\mathcal{D}$ . Since the noises added to  $\tilde{\mathbf{H}}_\mathcal{D}$  (line 19 of Algorithm 3) are drawn from  $Lap(\varepsilon_\mathcal{D}, 0, 4)$  (i.e.,  $\delta = 0$ ),  $\mathcal{M}$  satisfies pure  $\varepsilon_\mathcal{D}$ -edge DP.  $\square$

We can apply the same method for other leakage.

**THEOREM 3.** *Given  $\mathcal{L}_\Delta = (\tilde{n}_\Delta^A, \tilde{n}_\Delta^B)$ , in the view of the adversary, both  $\tilde{n}_\Delta^A$  and  $\tilde{n}_\Delta^B$  adhere to pure  $\varepsilon_\Delta$ -edge DP.*

PROOF. First, we establish the  $\varepsilon_\Delta$ -edge DP of  $\tilde{n}_\Delta^A$ , and the same conclusion holds for  $\tilde{n}_\Delta^B$ . Given  $G, G'$ , we have  $|n_\Delta^A - n_\Delta'^A| \leq k - 1$ , where  $n_\Delta^A, n_\Delta'^A$  represent the count of triangle  $\text{Tri}_\Delta$  in  $G, G'$  respectively, after edge clipping with parameter  $k$ . The probability to output the same noisy count  $\tilde{n}_\Delta^A$  is bounded by

$$\begin{aligned} \frac{Pr[\mathcal{M}(G) = \tilde{n}_\Delta^A]}{Pr[\mathcal{M}(G') = \tilde{n}_\Delta^A]} &= \frac{Pr[n_\Delta^A - \tilde{n}_\Delta^A]}{Pr[n_\Delta'^A - \tilde{n}_\Delta^A]} = \frac{e^{-\frac{\varepsilon_\Delta \cdot |n_\Delta^A - \tilde{n}_\Delta^A|}{k-1}}}{e^{-\frac{\varepsilon_\Delta \cdot |n_\Delta'^A - \tilde{n}_\Delta^A|}{k-1}}} \\ &= e^{\frac{\varepsilon_\Delta}{k-1} \cdot (|n_\Delta'^A - \tilde{n}_\Delta^A| - |n_\Delta^A - \tilde{n}_\Delta^A|)} \leq e^{\frac{\varepsilon_\Delta}{k-1} \cdot (|n_\Delta^A - n_\Delta'^A|)} \leq e^{\varepsilon_\Delta}. \end{aligned}$$

$\mathcal{M}$  satisfies  $\varepsilon_\Delta$ -edge DP for  $\tilde{n}_\Delta^A$  based on Definition 2.  $\square$

## 7.3 Security Analysis

We use the simulation paradigm [26] to analyze the security.

**DEFINITION 4.** *Let  $[\ ]$  denote the protocol for securely constructing dAGM. Let  $\mathcal{A}$  be an adversary who statically corrupts one of  $P_{\{1,2,3\}}$*

**Table 1: Overall<sub>diff</sub> under different privacy budget allocations**

| $[\epsilon_D, \epsilon_X, \epsilon_F, \epsilon_\Delta]$ | MOOC         | Twitter      | Gplus        |
|---|--------------|--------------|--------------|
| [0.3, 0.1, 0.2, 0.2]                                    | 0.191        | 0.185        | 0.221        |
| [0.3, 0.3, 0.2, 0.1]                                    | 0.171        | 0.159        | 0.194        |
| [0.4, 0.1, 0.1, 0.2]                                    | 0.163        | 0.132        | 0.154        |
| [0.4, 0.3, 0.1, 0.1]                                    | 0.159        | 0.135        | 0.155        |
| [0.5, 0.2, 0.1, 0.1]                                    | 0.126        | 0.119        | 0.125        |
| [0.6, 0.1, 0.1, 0.1]                                    | <b>0.108</b> | <b>0.096</b> | <b>0.113</b> |

or one of the users, and let  $\text{View}_{\text{Real}}^\Pi$  be the view of the corrupted parties during the protocol run. In the ideal world, a simulator  $S$  generates a simulated view  $\text{View}_{\text{Ideal}}^{S, \mathcal{L}}$  given only the leakage  $\mathcal{L}$ . We say that  $\Pi$  is secure, if there exists a probabilistic polynomial time simulator  $S$  such that  $\text{View}_{\text{Ideal}}^{S, \mathcal{L}}$  is indistinguishable from  $\text{View}_{\text{Real}}^\Pi$ .

**THEOREM 4.** Based on Definition 4, PrivAGM is secure.

We present the proof in Appendix C of the full version [45].

## 8 PERFORMANCE EVALUATION

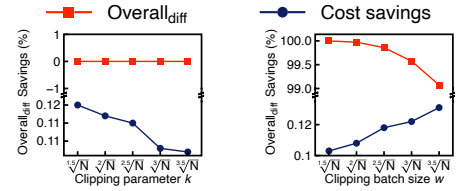
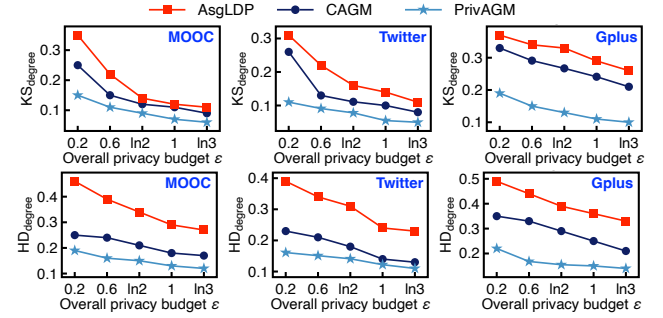
### 8.1 Setup

We develop a prototype implementation of PrivAGM using a combination of Python and C++. All experiments are conducted on a workstation equipped with 24 Intel Xeon Gold 6240R CPU cores, a NVIDIA RTX A6000 GPU, 128 GB of RAM, and 2 TB of external SSD storage, running Ubuntu 20.04.3 LTS. Following the setup of prior MPC-based studies [24, 30], each aggregator is executed as an independent process on the workstation. To emulate the network environment, we use the Linux tc command to simulate a 1 Gbps bandwidth with 1 ms latency. In addition, we use a MacBook Air with 8 GB of RAM to simulate users, secretly share the local views. All public and private values are encoded in  $\mathbb{Z}_{2^{32}}$ . In addition, we set the privacy parameters  $\delta_D = \delta_X = \delta_F = 10^{-5}$ .

**Datasets.** We use three directed social graph datasets: MOOC (7047 vertices; 411749 edges; 3.7 MB) [25], Twitter (81306 vertices; 1768149 edges; 21.1 MB) [28], and Gplus (107614 vertices; 13673453 edges; 156.5 MB) [28]. For each dataset, we treat each vertex as a user. Therefore, for the three datasets, the average amount of data held by each user is about 0.54 KB, 0.27 KB, and 1.49 KB, respectively.

**Baselines.** We use two state-of-the-art methods as baselines for utility comparison: CAGM, which operates in a centralized setting with edge DP guarantees [6], and AsgLDP, which operates in a decentralized setting with edge LDP guarantees [49].

**Utility metrics.** We evaluate the utility using four widely-used metrics [6, 22, 34, 49]. (1) *Degree distribution:* We use the Kolmogorov-Smirnov statistic to evaluate how well a synthetic graph captures the degree distribution of the original graph ( $\text{KS}_{\text{degree}}$ ). As  $\text{KS}_{\text{degree}}$  is less sensitive to differences in the tails of the distributions, we also report the Hellinger distance ( $\text{HD}_{\text{degree}}$ ) between the degree distributions of the synthetic and original graphs. (2) *Attribute-edge correlation distribution:* We report the mean relative error (MRE) ( $\text{MRE}_{\text{corr}}$ ) and the Hellinger distance  $\text{HD}_{\text{corr}}$  between the attribute-edge correlation distribution of the synthetic and original graphs. (3) *Local clustering coefficient:* We report the MRE between the average of the local clustering coefficient of the synthetic and original graphs ( $\text{MRE}_{\text{cluster}}$ ). (4) *Triangle counts:* We report the average MRE in TriA and TriB ( $\text{MRE}_{\text{Tri}}$ ).

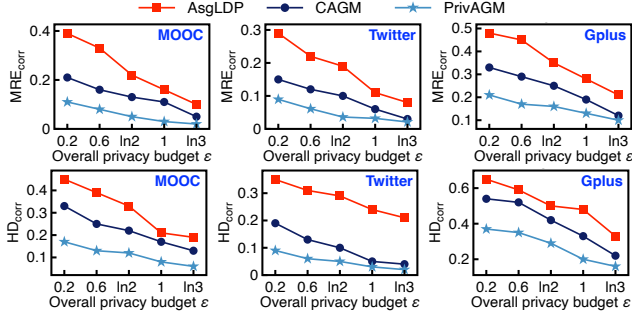
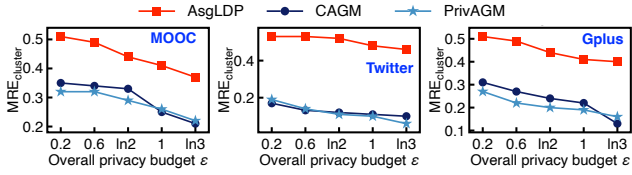
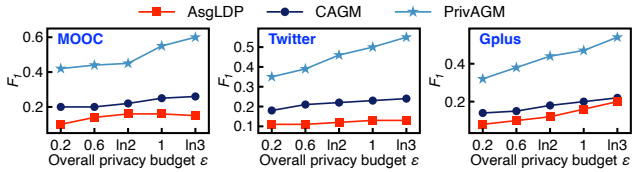
**Figure 4: Overall<sub>diff</sub> and cost savings under different  $k$  and  $w$ .****Figure 5: Comparison in terms of  $\text{KS}_{\text{degree}}$  and  $\text{HD}_{\text{degree}}$ .**

**Downstream task.** We also implement a widely used downstream task—truss-based community detection—which aims to identify groups or clusters of vertices in a graph that are more densely connected to each other than to the rest of the graph [27]. We use the  $F_1$  score as the metric to evaluate the differences between the truss communities detected on the synthetic graphs and those detected on the original graphs. The detailed computation method of the above metrics is given in Appendix D of the full version [45].

### 8.2 Effect of Key Parameters

**Privacy budget allocation.** We first determine the best privacy budget allocation of PrivAGM by empirically measure the overall difference  $\text{Overall}_{\text{diff}}$ .  $\text{Overall}_{\text{diff}}$  is calculated as the average of the above utility metrics, excluding the  $F_1$  score of the truss community structures. Recall that only four privacy budgets impact the utility of the synthetic graphs:  $\epsilon_D$  for the degree sequence,  $\epsilon_X$  for the attribute-edge correlation distribution,  $\epsilon_F$  for the attribute distribution, and two  $\epsilon_\Delta$  for the two types of triangle counts. Hence, the overall privacy budget allocation can be represented as  $\epsilon = \epsilon_D + \epsilon_X + \epsilon_F + 2 \cdot \epsilon_\Delta$ . The results are shown in Table 1 ( $\epsilon = 1$ ). We can observe that a larger  $\epsilon_D$  corresponds to a smaller  $\text{Overall}_{\text{diff}}$ , and the optimal privacy budget allocation is  $\epsilon_D = 0.6, \epsilon_X = 0.1, \epsilon_F = 0.1, \epsilon_\Delta = 0.1$ . This is due to the degree sequence is crucial for capturing the input graph's structure [22].

**Effect of clipping parameters.** We next evaluate how the clipping parameter  $k$  and batch size  $w$  affect  $\text{Overall}_{\text{diff}}$  and the system cost. Fig. 4 presents the experimental results on the Gplus dataset. We observe that  $k$  does not impact the cost, as the edge clipping is performed in an oblivious manner, incurring a fixed overhead regardless of the value of  $k$ . However, a smaller  $k$  results in lower utility loss, as it reduces the sensitivity during noise generation for DP guarantees, thereby decreasing the magnitude of the noise and better preserving the graph structure. We observe that larger values of  $w$  yield greater cost savings. However, this comes at the expense

Figure 6: Comparison in terms of  $MRE_{corr}$  and  $HD_{corr}$ .Figure 7: Comparison in terms of  $MRE_{cluster}$ .Figure 8: Comparison in terms of  $F_1$  score.

of higher  $Overall_{diff}$ , indicating a trade-off between efficiency and utility. In the following experiments, we set  $k = \sqrt[3]{N}$  and  $w = \sqrt{N}$ .

### 8.3 Utility Comparison

**Degree distribution.** Fig. 5 presents a comparison between PrivAGM and the baselines in terms of  $KS_{degree}$  and  $HD_{degree}$ . It is evident that AsgLDP shows the largest  $KS_{degree}$  and  $HD_{degree}$  since it is based on LDP, which introduces larger noise than central DP under the same privacy budget. In contrast, our PrivAGM achieves the smallest  $KS_{degree}$  and  $HD_{degree}$ .

**Attribute-edge correlation distribution.** Fig. 6 shows a comparison between PrivAGM and the baselines in terms of  $MRE_{corr}$  and  $HD_{corr}$ . PrivAGM consistently achieves the smallest  $MRE_{corr}$  and  $HD_{corr}$  across all privacy budgets and datasets. Moreover, as  $\epsilon$  decreases, the difference in  $MRE_{corr}$ ,  $HD_{corr}$  between PrivAGM and the baselines becomes more significant.

**Local clustering coefficient.** Fig. 7 presents a comparison between PrivAGM and the baselines in terms of  $MRE_{cluster}$ . We can observe that  $MRE_{cluster}$  of PrivAGM and CAGM is approximately similar. In certain cases (e.g.,  $\epsilon = \ln 3$  on Gplus), the  $MRE_{cluster}$  of CAGM is even lower than that of PrivAGM. This disparity is because  $MRE_{cluster}$  fails to capture the difference in edge directionality between the synthetic and the original graphs. Hence, the advantage of PrivAGM in capturing the directionality of the original graphs cannot be fully reflected by  $MRE_{cluster}$ . We compare the utility of triangle counting in Appendix E of the full version [45].

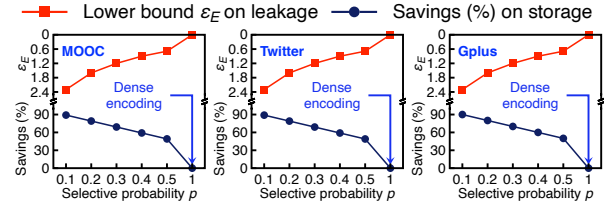
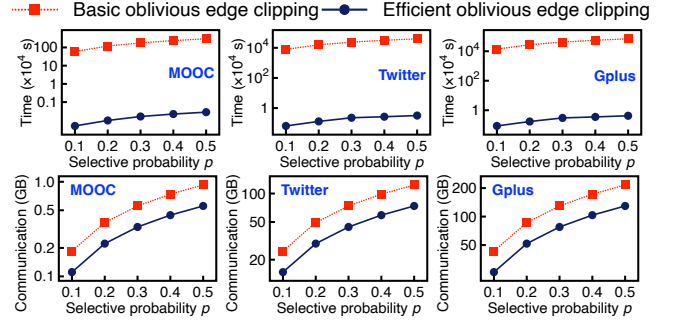
Figure 9: Evaluation of  $\epsilon_E$  and the savings on storage.

Figure 10: Comparison of oblivious edge clipping methods.

**Downstream task: truss community detection.** Fig. 8 compares PrivAGM with the baselines in terms of the  $F_1$  score of the truss community structures. The results show that PrivAGM consistently achieves the highest  $F_1$  scores across all privacy budgets and datasets. Moreover, PrivAGM's  $F_1$  score is comparable to that of non-private methods [1, 27, 50], highlighting the practical utility of the synthetic graphs it generates for real-world downstream tasks.

### 8.4 System Cost Evaluation

**Local neighbor list secret sharing.** The amount of data each user sent to each server is approximately  $32 \cdot N \cdot \frac{3 \cdot p}{4}$ . For example, when  $p = 0.1$ , the amount of data sent to each server is approximately 2.1 KB, 23.8 KB, and 31.5 KB for the MOOC, Twitter, and Gplus datasets, respectively. The ratio of user-held data to the data transmitted to each server is 0.25, 0.01, and 0.05 for the three datasets, respectively. Fig. 9 presents the lower bound  $\epsilon_E$  and the savings on the resulting size. The results show that PrivAGM achieves a significant reduction in the resulting ciphertext size compared to direct secret sharing of the complete local neighbor list (up to 90% under  $p = 0.1$ ,  $\epsilon_E \approx 2.3$ ).

**Oblivious edge clipping.** Fig. 10 compares the basic and efficient oblivious edge clipping protocols. The results for the basic protocol are estimated from processing 1000 edges, as it is significantly slow (taking a few days). The comparison shows that the efficient protocol achieves a substantial speedup compared to the basic protocol (up to 10000×). Additionally, the efficient protocol achieves approximately 50% savings in communication cost compared to the basic protocol. For instance, with  $p = 0.1$  on the Twitter dataset, the basic protocol incurs a communication cost of 0.19GB, requires 7,449,031 rounds of communication, and takes 5.4 hours for offline preparation. In contrast, the efficient protocol reduces the communication cost to 0.1GB, requires only 89,032 rounds of communication, and takes just 0.06 hours for offline preparation.

**Differentially private dAGM construction.** Fig. 11 shows the time and communication costs, along with their breakdown. We

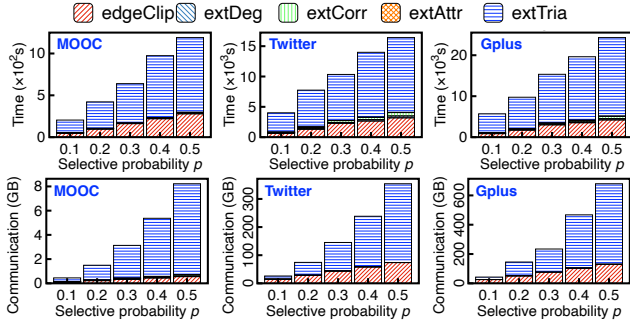


Figure 11: Time/communication cost of dAGM construction.

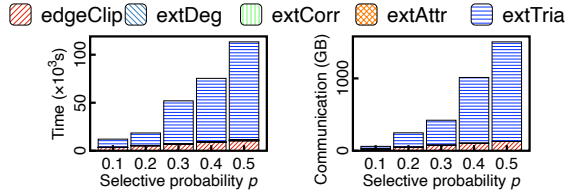


Figure 12: Time/communication cost on the Weibo dataset.

can observe that the majority of the cost is attributed to the secure triangle counting. Its high time cost is due to the communication required by the 3-input multiplication gates (i.e., lines 7, 11 of Algorithm 5). Additionally, the results further demonstrate that our local neighbor list secret sharing can significantly save the cost. While the cost of dAGM construction is relatively high, it is a *one-off* process. In addition, the system’s cost is sensitive to the selective probability  $p$ . For example, with  $p = 0.1$  on the Twitter dataset, PrivAGM requires approximately 1 hour of computation, 801,288 rounds of communication, and 0.7 hours for offline preparation.

**Scalability.** To evaluate the scalability of PrivAGM, we conduct experiments on the significantly larger Weibo social graph dataset<sup>1</sup>. This dataset contains 1,776,950 vertices and 308,489,739 edges. Fig. 12 presents the results, which demonstrate the good scalability of PrivAGM. Even on a graph with  $\sim 1.7$  million vertices, the running time increases by only  $2.12\times$  to  $4.66\times$  and the communication cost increases by just  $1.43\times$  to  $2.2\times$ , compared to the case of Gplus dataset with  $\sim 107k$  vertices.

## 8.5 Remark

Based on the experimental results, we observe a clear trade-off among privacy, utility, and efficiency in PrivAGM. As stronger privacy guarantees (i.e., smaller  $\epsilon$ ) are enforced, the utility of the generated synthetic graphs tends to diminish. Nonetheless, PrivAGM consistently outperforms baselines [6, 49] in utility preservation, demonstrating its robustness and adaptability. The superior performance of PrivAGM compared to the centralized method [6] can be attributed to two main factors. Firstly, while PrivAGM is a decentralized method, the noise is added to the extracted parameters only after the social graph has been collected. This means that, similar to the centralized method, the noise is injected at the same point in the process, i.e., during the post-collection phase. Secondly,

<sup>1</sup>Weibo dataset is available at <https://www.aminer.cn/influencelocality>.

PrivAGM is the first approach that explicitly considers the inherent directionality of edges in the input graph. This allows PrivAGM to preserve the structural characteristics of the graph more effectively than centralized method. This improvement in utility, however, inherently comes at a moderate computational cost. Overall, the results validate that PrivAGM achieves an effective balance among privacy protection, utility preservation, and computational efficiency, making it well-suited for practical deployment in realistic decentralized social graph scenarios.

## 9 RELATED WORK

**Private centralized graph analytics.** In the *centralized* setting, there is a substantial body of work on private graph analysis. In this setting, a curator holds the complete graph and publishes some differentially private information. Some works publish differentially private statistics of the original graph, e.g., degree distribution [35] and subgraph counting [10]. Other works [6, 22, 52] develop differentially private generative graph models. Chen *et al.* [6] propose CAGM, an extension of [22], which additionally captures the community structure. However, since they are designed for the centralized setting, they cannot be directly applied to the decentralized setting. Moreover, they focus solely on *undirected* graphs.

**Private decentralized graph analytics.** Another line of work considers the more challenging *decentralized* setting. Several works [19, 20, 41, 51] focus on subgraph counting. Other works [5, 34, 49, 54] focus on constructing differentially private generative graph models. However, these methods [5, 34, 54] fail to capture the attributed information of the graph. Wei *et al.* [49] propose AseLDP, which considers decentralized *attributed* graphs by incorporating the attribute-edge correlation into the framework of [34]. AseLDP is the work most related to ours. However, since it does not consider the directionality of the edges in the input graph, it has poor utility when dealing with directed graphs, as shown in our experiments. **Graph neural network training with privacy awareness.** There exist other works on training differentially private graph neural networks (GNNs) in the centralized setting [39] or decentralized setting [38]. However, the work in [39] is designed for the centralized setting and is not applicable to our decentralized setting. For the work in [38], their techniques are specifically developed for training GNNs with DP guarantees over decentralized social graphs. Since their approach is centered around adapting the GNN model parameters based on the loss function, it does not address the generation of synthetic graphs. As a result, the problems tackled and the techniques used in these studies are fundamentally different from those explored in this paper, and their methodologies and objectives remain distinct.

## 10 CONCLUSION

This paper presents PrivAGM, the first solution for the secure construction of differentially private dAGM on decentralized social graphs. By bridging the gap among edge DP, edge LDP, MPC, and generative graph models, PrivAGM enables aggregators to effectively construct differentially private dAGM on decentralized social graphs without compromising the individual privacy of the users. The evaluation results on three real-world graph datasets demonstrate that PrivAGM outperforms the state-of-the-art baselines.

## REFERENCES

- [1] Wei Ai, Canhao Xie, Tao Meng, Jayi Du, and Keqin Li. 2024. A D-truss-equivalence Based Index for Community Search over Large Directed Graphs. *IEEE Transactions on Knowledge and Data Engineering* (2024).
- [2] Apple and Google. 2020. Privacy-Preserving Contact Tracing. <https://covid19.apple.com/contacttracing>.
- [3] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. 2013. Differentially private data analysis of social networks via restricted sensitivity. In *Proc. of ACM ITCS*.
- [4] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. 2021. Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation. In *Proc. of EUROCRYPT*.
- [5] Felipe T Brito, Victor AE Farias, Cheryl Flynn, Subhabrata Majumdar, Javam C Machado, and Divesh Srivastava. 2023. Global and Local Differentially Private Release of Count-Weighted Graphs. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–25.
- [6] Xihui Chen, Sjouke Mauw, and Yuniior Ramirez-Cruz. 2020. Publishing community-preserving attributed social graphs with a differential privacy guarantee. *Proceedings on Privacy Enhancing Technologies* 2020, 4 (2020), 131–152.
- [7] Fan Chung and Linyuan Lu. 2002. The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences* 99, 25 (2002), 15879–15882.
- [8] Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko, Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Søren B. Lassen, Philip Pronin, Sriram Sankar, Guanghao Shen, Gintaras Woss, Chao Yang, and Ning Zhang. 2013. Unicorn: A System for Searching the Social Graph. *Proceedings of the VLDB Endowment* 6, 11 (2013), 1150–1161.
- [9] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *Proc. of NDSS*.
- [10] Xiaofeng Ding, Shujun Sheng, Huajian Zhou, Xiaodong Zhang, Zhifeng Bao, Pan Zhou, and Hai Jin. 2021. Differentially private triangle counting in large graphs. *IEEE Transactions on Knowledge and Data Engineering* 34, 11 (2021), 5278–5292.
- [11] Irit Dinur and Kobbi Nissim. 2003. Revealing information while preserving privacy. In *Proc. of ACM PODS*.
- [12] Derek Doran. 2014. Triad-based role discovery for large social systems. In *Proc. of SocInfo*.
- [13] Cynthia Dwork. 2006. Differential Privacy. In *Proc. of ICALP*.
- [14] Reo Eriguchi, Atsunori Ichikawa, Noboru Kunihiro, and Koji Nuida. 2021. Efficient Noise Generation to Achieve Differential Privacy with Applications to Secure Multiparty Computation. In *Proc. of FC*.
- [15] Saba Eskandarian and Dan Boneh. 2022. Clarion: Anonymous communication from multiparty shuffling protocols. In *Proc. of NDSS*.
- [16] Fireblocks. 2023. Remove the complexity of working with digital assets. online at <https://www.fireblocks.com>. [Online; Accessed 1-Jan-2024].
- [17] Xi He, Ashwin Machanavajjhala, Cheryl J. Flynn, and Divesh Srivastava. 2017. Composing Differential Privacy and Secure Computation: A Case Study on Scaling Private Record Linkage. In *Proc. of ACM CCS*.
- [18] Thomas Humphries, Rasoul Akhavan Mahdavi, Shannon Veitch, and Florian Kerschbaum. 2022. Selective MPC: Distributed Computation of Differentially Private Key-Value Statistics. In *Proc. of ACM CCS*.
- [19] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2021. Locally Differentially Private Analysis of Graph Statistics. In *Proc. of USENIX Security*.
- [20] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2022. Communication-efficient triangle counting under local differential privacy. In *Proc. of USENIX Security*.
- [21] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2022. Differentially Private Triangle and 4-Cycle Counting in the Shuffle Model. In *Proc. of ACM CCS*.
- [22] Zach Jorgensen, Ting Yu, and Graham Cormode. 2016. Publishing attributed social graphs with formal privacy guarantees. In *Proc. of ACM SIGMOD*.
- [23] Shiva Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam D. Smith. 2011. What Can We Learn Privately? *SIAM J. Comput.* 40, 3 (2011), 793–826.
- [24] Nishat Koti, Varsha Bhat Kukkala, Arpita Patra, and Bhavish Raj Gopal. 2024. Graphiti: Secure Graph Computation Made More Scalable. In *Proc. of ACM CCS*.
- [25] Srikanth Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proc. of ACM KDD*.
- [26] Yehuda Lindell. 2017. How to Simulate It - A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*. 277–346.
- [27] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based community search over large directed graphs. In *Proc. of ACM SIGMOD*.
- [28] Julian J. McAuley and Jure Leskovec. 2012. Learning to Discover Social Circles in Ego Networks. In *Proc. of NeurIPS*.
- [29] Meta. 2023. The value of secure multi-party computation. online at <https://privacytech.fb.com/multi-party-computation/>. [Online; Accessed 1-Jan-2024].
- [30] Payman Mohassel and Peter Rindal. 2018. ABY<sup>3</sup>: A Mixed Protocol Framework for Machine Learning. In *Proc. of ACM CCS*.
- [31] MPCVault. 2023. Multisig crypto wallet for business. online at <https://mpcvault.com>. [Online; Accessed 1-Jan-2024].
- [32] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In *Proc. of USENIX Security*.
- [33] Joseph J Pfeiffer III, Sebastian Moreno, Timothy La Fond, Jennifer Neville, and Brian Gallagher. 2014. Attributed graph models: Modeling network structure with correlated attributes. In *Proc. of ACM WWW*.
- [34] Zhan Qin, Ting Yu, Yin Yang, Issa Khalil, Xiaokui Xiao, and Kui Ren. 2017. Generating Synthetic Decentralized Social Graphs with Local Differential Privacy. In *Proc. of ACM CCS*.
- [35] Sofya Raskhodnikova and Adam Smith. 2016. Lipschitz extensions for node-private graph statistics and the generalized exponential mechanism. In *Proc. of IEEE FOCS*.
- [36] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. 2020. Crypte: Crypto-assisted differential privacy on untrusted servers. In *Proc. of ACM SIGMOD*.
- [37] Safeheron. 2025. Safeheron MPC Wallet. <https://safeheron.com>.
- [38] Sina Sajadmanesh and Daniel Gatica-Perez. 2021. Locally private graph neural networks. In *Proc. of ACM CCS*.
- [39] Sina Sajadmanesh, Ali Shahin Shamsabadi, Aurélien Bellet, and Daniel Gatica-Perez. 2023. GAP: Differentially private graph neural networks with aggregation perturbation. In *Proc. of USENIX Security*.
- [40] Scuttlebutt. 2025. P2P social network. <https://covid19.apple.com/contacttracing>.
- [41] Haipei Sun, Xiaokui Xiao, Issa Khalil, Yin Yang, Zhan Qin, Wendy Hui Wang, and Ting Yu. 2019. Analyzing Subgraph Statistics from Extended Local Views with Decentralized Differential Privacy. In *Proc. of ACM CCS*.
- [42] Sijun Tan, Weikeng Chen, Ryan Deng, and Raluca Ada Popa. 2023. MPCAuth: Multi-factor Authentication for Distributed-trust Systems. In *Proc. of IEEE S&P*.
- [43] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. 2021. CryptGPU: Fast Privacy-Preserving Machine Learning on the GPU. In *Proc. of IEEE S&P*.
- [44] Chenghong Wang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2022. IncShrink: architecting efficient outsourced databases using incremental mpc and differential privacy. In *Proc. of ACM SIGMOD*.
- [45] Songlei Wang, Yifeng Zheng, Xiaohua Jia, and Haibo Hu. 2025. The full version of PrivAGM. [https://github.com/songleiW/PrivAGM/blob/main/PrivAGM\\_full\\_version.pdf](https://github.com/songleiW/PrivAGM/blob/main/PrivAGM_full_version.pdf).
- [46] Zuan Wang, Xiaofeng Ding, Hai Jin, and Pan Zhou. 2022. Efficient secure and verifiable location-based skyline queries over encrypted data. *Proceedings of the VLDB Endowment* 15, 9 (2022), 1822–1834.
- [47] Stanley L Warner. 1965. Randomized response: A survey technique for eliminating evasive answer bias. *J. Amer. Statist. Assoc.* 60, 309 (1965), 63–69.
- [48] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of ‘small-world’ networks. *Nature* 393, 6684 (1998), 440–442.
- [49] Chengkun Wei, Shouling Ji, Changchang Liu, Wenzhi Chen, and Ting Wang. 2020. AsgLDP: collecting and generating decentralized attributed graphs with local differential privacy. *IEEE Transactions on Information Forensics and Security* 15 (2020), 3239–3254.
- [50] Jaewon Yang, Julian McAuley, and Jure Leskovec. 2013. Community detection in networks with node attributes. In *Proc. of IEEE ICDM*.
- [51] Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. 2022. LF-GDPR: A framework for estimating graph metrics with local differential privacy. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (2022), 4905–4920.
- [52] Quan Yuan, Zhikun Zhang, Linkang Du, Min Chen, Peng Cheng, and Mingyang Sun. 2023. PrivGraph: Differentially Private Graph Data Publication by Exploiting Community Information. In *Proc. of USENIX Security*.
- [53] Yanping Zhang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2023. Longshot: Indexing growing databases using MPC and differential privacy. *Proceedings of the VLDB Endowment* 16, 8 (2023), 2005–2018.
- [54] Yuxuan Zhang, Jianghong Wei, Xiaojian Zhang, Xuexian Hu, and Wenfen Liu. 2018. A two-phase algorithm for generating synthetic graph under local differential privacy. In *Proc. of ACM ICCNS*.

## A SECURE ATTRIBUTE DISTRIBUTION EXTRACTION

We present how  $P_{1,2,3}$  securely extract the noisy version of attribute distribution  $\Theta_F$  from  $\{\llbracket f_i \rrbracket\}$ , while ensuring that the extracted noisy distribution (denoted as  $\tilde{\Theta}_F$ ) is differentially private in the individual view of  $P_{1,2,3}$ . The design intuition is the same as that of Algorithm 4. Specifically,  $P_{1,2,3}$  first securely extract the differentially private histogram of attributes, then derive  $\tilde{\Theta}_F$  from the histogram. Algorithm 7 gives our secure attribute extraction protocol. Here, the



**Algorithm 8: Directed Attributed Graphs Generation**


---

**Input:** The dAGM parameters  $\tilde{\mathcal{D}}^\pm, \tilde{\Theta}_X, \tilde{\Theta}_F, \tilde{n}_\Delta^A$ , and  $\tilde{n}_\Delta^B$ .  
**Output:** The synthetic directed social graph  $G = (E, \{f_i\})$ .

- 1  $\{f_i\}_{i \in [1, N]} = \text{Sample}(\tilde{\Theta}_F)$ . // Sample a vertex attribute set.
- 2  $E^{(0)} = \text{directedCL}(\tilde{\mathcal{D}}^\pm)$ . // Sample the initial edge set.
- 3  $E'^{(0)} = \text{Rewire}(E^{(0)}, \tilde{\mathcal{D}}^\pm, \tilde{n}_\Delta^A, \tilde{n}_\Delta^B)$ . // Rewire edges by Algo. 6.
- 4 Initialize the round  $\tau = 0$ .
- 5 **while** Edge acceptance probability  $\Theta_{Ac}$  does not converage **do**
- 6     Compute  $\Theta_X^{(\tau)}$  of the current graph  $(E'^{(\tau)}, \{f_i\})$ .
- 7      $R(f \rightarrow f') = \frac{\tilde{\Theta}_X(f \rightarrow f')}{\Theta_X^{(\tau)}(f \rightarrow f')}, f \rightarrow f' \in \mathbb{Z}_{2L} \times \mathbb{Z}_{2L}$ .
- 8     **if**  $\tau > 0$  **then**
- 9          $R(f \rightarrow f') = R(f \rightarrow f') \cdot \Theta_{Ac}^{(\tau-1)}(f \rightarrow f'), f \rightarrow f' \in \mathbb{Z}_{2L} \times \mathbb{Z}_{2L}$ .
- 10      $\Theta_{Ac}^{(\tau)}(f \rightarrow f') = \frac{R(f \rightarrow f')}{\text{SUP}(R)}, f \rightarrow f' \in \mathbb{Z}_{2L} \times \mathbb{Z}_{2L}$ .
- 11     // Sample a new edge set  $E'^{(\tau+1)}$  based on  $\Theta_{Ac}^{(\tau)}$ :
- 12      $E'^{(\tau+1)} = \text{directedCL}(\tilde{\mathcal{D}}^\pm, \{f_i\}, \Theta_{Ac}^{(\tau)})$ .
- 13      $E'^{(\tau+1)} = \text{Rewire}(E'^{(\tau+1)}, \tilde{\mathcal{D}}^\pm, \tilde{n}_\Delta^A, \tilde{n}_\Delta^B, \{f_i\}, \Theta_{Ac}^{(\tau)})$ .
- 14      $\tau = \tau + 1$ .
- 15 **return**  $E = E'^{(\tau)}$ ; The synthetic social graph  $G = (E, \{f_i\})$ .

---

**Algorithm 7: Secure Attribute Distribution Extraction**


---

**Input:**  $S$   
 1 ecret-shared attributes  $\{\llbracket f_i \rrbracket\}$ ; privacy budget, parameter  $\epsilon_F, \delta_F$ .  
**Output:**  $D$

- 2 differentially private attribute distribution  $\tilde{\Theta}_F$ .
- 3 //  $P_{1,2}$  operate on  $\{\llbracket f_i \rrbracket\}$ :
- 4  $\llbracket \mathbf{T} \rrbracket = []$ . // Empty table.
- 5 **for**  $\llbracket f_i \rrbracket \in \{\llbracket f_i \rrbracket\}$  **do**
- 6     Append row  $\llbracket f_i \rrbracket \parallel \llbracket \rho \rrbracket$  to  $\llbracket \mathbf{T} \rrbracket$ , where  $\llbracket \rho \rrbracket = \llbracket 1 \rrbracket$ .
- 7 //  $P_3$  constructs a dummy attribute value table  $\mathbf{T}'$ :
- 8  $\mathbf{T}' = []$ . // Empty table.
- 9 **for**  $f \in \mathbb{Z}_{2L}$  **do**
- 10      $n = \max(0, \text{Lap}(\epsilon_F, \delta_F, 2))$ . // Draw a noise. Append  $n$  rows of  $f \parallel \rho$  to  $\mathbf{T}'$ , where  $\rho = 0$ .
- 11  $P_3$  secret-shares  $\mathbf{T}'$  with  $P_{1,2}$  to produce  $\llbracket \mathbf{T}' \rrbracket$ .
- 12 //  $P_1, P_2$  perform the remaining operations:
- 13  $\llbracket \hat{\mathbf{T}} \rrbracket = \begin{bmatrix} \llbracket \mathbf{T} \rrbracket \\ \llbracket \mathbf{T}' \rrbracket \end{bmatrix}$ . // Vertical concatenation.
- 14  $\llbracket \hat{\mathbf{T}}' \rrbracket = \text{secShuffle}(\llbracket \hat{\mathbf{T}} \rrbracket)$ . // Oblivious row-wise shuffle.
- 15 Safely reveal column 1 of  $\llbracket \hat{\mathbf{T}}' \rrbracket$  to obtain  $\llbracket \hat{\mathbf{T}}' \rrbracket$ .
- 16 Initialize secret-shared histogram  $\llbracket \mathbf{H} \rrbracket = \llbracket 0 \rrbracket^{2^L}$ .
- 17 **for** row  $f \parallel \llbracket \rho \rrbracket$  in  $\llbracket \hat{\mathbf{T}}' \rrbracket$  **do**
- 18      $\llbracket \mathbf{H}[f] \rrbracket = \llbracket \mathbf{H}[f] \rrbracket + \llbracket \rho \rrbracket$ . // Index attribute value to histogram.
- 19  $\llbracket \tilde{\mathbf{H}}[f] \rrbracket = \llbracket \text{Lap}(\epsilon_F, 0, 2) \rrbracket, f \in \mathbb{Z}_{2L}$ .
- 20 Safely reveal the differentially private histogram  $\tilde{\mathbf{H}}$ .
- 21  $\tilde{\mathbf{H}}'[f] = \max(0, \tilde{\mathbf{H}}[f]), f \in \mathbb{Z}_{2L}$ . // Truncation.
- 22  $\tilde{\mathbf{H}}''[f] = \frac{\tilde{\mathbf{H}}'[f]}{\text{sum}(\tilde{\mathbf{H}}')}, f \in \mathbb{Z}_{2L}$ . // Normalization
- 23 Initialize the attribute distribution  $\tilde{\Theta}_F = \{0\}^{2^L}$ .
- 24  $\tilde{\Theta}_F(f) = \tilde{\mathbf{H}}''[f], f \in \mathbb{Z}_{2L}$ .
- 25 **return** Differentially private attribute distribution  $\tilde{\Theta}_F$ .

---

sensitivity  $\Delta$  is set to 2, as modifying the attribute of a single vertex leads to changes in the count of two attributes in the social graph.

**B ATTRIBUTED GRAPHS GENERATION**

Algorithm 8 gives the workflow for synthesizing directed graphs. Lines 6-10 calculate the new acceptance probabilities  $\Theta_{Ac}^{(\tau)}$  based on the attribute-edge correlation  $\Theta_X^{(\tau)}$  in the current graph  $(E'^{(\tau)}, \{f_i\})$  and  $\tilde{\Theta}_X$  derived from the original graph.  $\text{SUP}(R)$  is the least upper bound of set  $R$ . The new edge set  $E'^{(\tau+1)}$  is sampled (lines 12-13). Specifically, when the directed CL model or Algorithm 6 add an edge  $(i, j)$  to the new edge set, they first sample a value uniformly from the range of 0 to 1 ( $\text{Sample}(0, 1)$ ). The edge  $(i, j)$  is included in the new edge set (i.e., “accepted”) only when  $\text{Sample}(0, 1) \leq \Theta_{Ac}^{(\tau+1)}(f_i \rightarrow f_j)$ , where  $f_i$  and  $f_j$  are the attributes sampled for vertices  $i$  and  $j$  at line 1. If  $\text{Sample}(0, 1) > \Theta_{Ac}^{(\tau+1)}(f_i \rightarrow f_j)$ , the edge  $(i, j)$  is not included in the new edge set (i.e., “rejected”).

**C PROOF OF THEOREM 4**

**PROOF.** As no user receives any values from other parties, the simulator for the users must exist. Therefore, we only analyze the existence of the simulator for  $P_{\{1,2,3\}}$ . Since the roles of  $P_{\{1,2,3\}}$  in PrivAGM are symmetric, with the exception that  $P_{1,2}$  receive the secret shares of dummy entities (i.e.,  $\llbracket \mathbf{T}' \rrbracket$ ) from  $P_3$ , it is enough to analyze the existence of  $\mathcal{S}$  under the condition that  $\mathcal{A}$  corrupts  $P_1$ .  
**Simulator  $\mathcal{S}$  in Colle.** During this phase,  $P_1$  exclusively receives the set of tuples  $\{(i, j, \langle b_{ij} \rangle)\}$ . Therefore, we can trivially construct the simulator  $\mathcal{S}$  by invoking the simulator of ASS [9] and providing it with the output of  $\mathcal{L}_E$  as input.

**Simulator  $\mathcal{S}$  in edgeClip.** The efficient oblivious edge clipping protocol is presented in Algorithm 2. Initially,  $\mathcal{A}$  possesses the clipped local neighbor lists  $\{(i, j, \langle b_{ij} \rangle)\}$ ,  $k$ , and  $w$ .  $\mathcal{A}$  only receives messages from other honest aggregators during the execution of lines 8 and 10. The operations in line 8 is based on the FSS-based DCF from [4]. Hence, the simulator can be trivially constructed by invoking the simulator of FSS-based DCF [4] and providing it with  $k/3$  as input. The operation in line 10 relies on the basic secure multiplication operation in the ASS domain. Hence, the simulator can be trivially constructed by invoking the simulator of ASS [9].  
**Simulator  $\mathcal{S}$  in extDeg.** The secure degree sequence extraction protocol is given in Algorithm 3. Initially,  $\mathcal{A}$  possesses the clipped local neighbor lists  $\{(i, j, \langle b_{ij} \rangle)\}$ ,  $k, \epsilon_{\mathcal{D}}, \delta_{\mathcal{D}}$ , and the secret shares of Laplacian noises.  $\mathcal{A}$  only receives messages from other honest aggregators during the execution of lines 4, 11, 14, 15, and 20. The operations in lines 4 and 11 rely on the secret sharing operation in the ASS domain. Hence, the simulator can be trivially constructed by invoking the simulator of ASS [9]. The operation in line 14 is based on the oblivious shuffle protocol from [15]. Therefore, the simulator can be trivially constructed by invoking the simulator of oblivious shuffle [15]. During the execution of lines 15 and 20,  $\mathcal{A}$  receives secret shares from  $P_2$  to reveal  $\llbracket \hat{\mathbf{T}}' \rrbracket$  and  $\tilde{\mathbf{H}}$ . Based on the security of ASS, the secret shares from  $P_2$  are uniformly random in



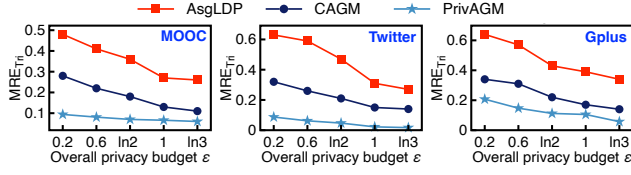


Figure 13: Comparison in terms of  $MRE_{Tri}$ .

the view of  $\mathcal{A}$ . For revealed  $[\hat{T}']$  and  $\tilde{H}$ ,  $\mathcal{S}$  adjusts  $P_2'$  shares such that the revealed values are indeed the coefficient vector it received from  $\mathcal{F}$  (i.e.,  $\mathcal{L}_{\mathcal{D}}$ ).

**Simulator  $\mathcal{S}$  in extCorr and extAttr.** As the analysis of the existence of  $\mathcal{S}$  in extCorr and extAttr follows the same idea as that in extDeg, we omit the analysis here.

**Simulator  $\mathcal{S}$  in extTria.** The secure triangle counts extraction protocol is given in Algorithm 5. Initially,  $\mathcal{A}$  possesses  $\{(i, j, (b_{ij}))\}$ ,  $k$ ,  $\varepsilon_{\mathcal{D}}$ ,  $\delta_{\mathcal{D}}$ , and the secret shares of Laplacian noises.  $\mathcal{A}$  only receives messages from  $P_{2,3}$  during the execution of lines 7, 11, and 16. The operations in lines 7 and 11 are based on the secure 3-input multiplication gate from [32]. Hence, the simulator can be trivially constructed by invoking the simulator of the secure 3-input multiplication gate [32]. During the execution of line 16,  $\mathcal{A}$  receives secret shares from  $P_2$  and  $P_3$  to reveal  $\tilde{n}_{\Delta}^A$  and  $\tilde{n}_{\Delta}^B$ . Based on the security of ASS, the secret shares from  $P_{2,3}$  are uniformly random in the view of  $\mathcal{A}$ . For revealed  $\tilde{n}_{\Delta}^A, \tilde{n}_{\Delta}^B$ ,  $\mathcal{S}$  adjusts the shares of  $P_{2,3}$  such that the revealed values are indeed the coefficient vector it received from  $\mathcal{F}$  (i.e.,  $\mathcal{L}_{\Delta}$ ).  $\square$

## D UTILITY METRICS

We evaluate the utility of the synthetic graphs using four widely-used metrics [6, 22, 34, 49].

- **Degree distribution:** We utilize the Kolmogorov-Smirnov statistic to evaluate how well a synthetic graph captures the degree distribution of the original graph (denoted as  $KS_{degree}$ ), which quantifies the maximum distance between two distributions. Let  $c_{\mathcal{D}^+}$  and  $c'_{\mathcal{D}^+}$  represent the cumulative distribution functions estimated from the sorted degree pair sequences of the input and synthetic graphs, respectively. Then  $KS_{degree} = \max_{d^+=0}^{d_{max}^+} \max_{d^-=0}^{d_{max}^-} |c_{\mathcal{D}^+}(d^+, d^-) - c'_{\mathcal{D}^+}(d^+, d^-)|$ , where  $d_{max}^+/d_{max}^-$  are the maximum out/in-degree in the input and synthetic graphs.  $c_{\mathcal{D}^+}(d^+, d^-)$  (and  $c'_{\mathcal{D}^+}(d^+, d^-)$ ) represents the percentage of vertices in the input (and synthetic) graph whose out-degrees  $\leq d^+$  and in-degrees  $\leq d^-$ . As  $KS_{degree}$  is less sensitive to differences in the tails of the degree distributions, we additionally report the Hellinger distance between two degree distributions  $\Theta_{\mathcal{D}^+}$  and  $\Theta'_{\mathcal{D}^+}$ :

$$HD_{degree} = \frac{1}{\sqrt{2}} \cdot \sqrt{\sum_{d^+=0}^{d_{max}^+} \sum_{d^-=0}^{d_{max}^-} (\sqrt{\Theta_{\mathcal{D}^+}(d^+, d^-)} - \sqrt{\Theta'_{\mathcal{D}^+}(d^+, d^-)})^2},$$

where  $\Theta_{\mathcal{D}^+}(d^+, d^-)$  is the percentage of vertices having a degree pair  $(d^+, d^-)$  in the graph. A smaller  $KS_{degree}$  and  $HD_{degree}$  indicate

a closer alignment between the degree distributions of the synthetic and original graphs.

- **Attribute-edge correlation distribution:** To evaluate how well a synthetic graph captures the attribute-edge correlations of the original graph, we report the mean relative error (MRE) between their attribute-edge correlation distributions (denoted as  $MRE_{corr}$ ), and the Hellinger distance:

$$HD_{corr} = \frac{1}{\sqrt{2}} \cdot \sqrt{\sum_{f=0}^{2^L} \sum_{f'=0}^{2^L} (\sqrt{\Theta_{\tilde{X}}(f \rightarrow f')} - \sqrt{\Theta'_{\tilde{X}}(f \rightarrow f')})^2},$$

where  $\Theta_{\tilde{X}}(f \rightarrow f')$  is the proportion of edges that connect a vertex with attribute value  $f$  to a vertex with attribute value  $f'$  in a graph.

- **Local clustering coefficient:** The local clustering coefficient of a vertex in a graph quantifies how close its neighbors are to forming a clique. The local clustering coefficient  $C_i$  for a vertex  $i$  is determined by the proportion of edges between the vertices within its neighborhood divided by the number of edges that could potentially exist between them. In directed graphs,  $C_i = \frac{|\{(j,k): j,k \in Ne_i, (j,k) \in E\}|}{(d_i^+ + d_i^-) \cdot (d_i^+ + d_i^- - 1)}$ , where  $Ne_i$  is the set of vertex  $i$ 's out/in-neighbors,  $E$  is the edge set, and  $(j, k)$  represents the edge from vertex  $j$  to vertex  $k$  [48]. The average of the local clustering coefficient is  $C = \frac{\sum_{i \in [1, N]} C_i}{N}$ . We report the MRE between the average of the local clustering coefficient of the synthetic and original graphs (denoted as  $MRE_{cluster}$ ).

- **Triangle counts:** We report the average MRE in the number of triangles,  $TriA$  and  $TriB$ , between the synthetic and original graphs (denoted as  $MRE_{Tri}$ ).

- **Community structures:** We report the  $F_1$  score of the truss community structures [27]. The truss community is a recently developed and widely adopted community model for directed social graphs. It aims to identify a subgraph that satisfies the query's connectivity requirements. Given two communities  $C_1$  and  $C_2$ , the  $F_1$  score is defined as

$$F_1 = \frac{2 \cdot \text{prec}(C_1, C_2) \cdot \text{recall}(C_1, C_2)}{\text{prec}(C_1, C_2) + \text{recall}(C_1, C_2)},$$

where  $\text{prec}(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1|}$  and  $\text{recall}(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_2|}$ .

## E UTILITY OF TRIANGLE COUNTING

Fig. 13 presents a comparison between PrivAGM and the baselines in terms of  $MRE_{TriA}$  and  $MRE_{TriB}$ . The results demonstrate that PrivAGM consistently achieves a significantly small  $MRE_{TriA}$ ,  $MRE_{TriB}$  (even smaller than 0.1 in most instances) compared to the baselines and validate our sampling-based secure triangle counting scheme. In contrast, AsgLDP consistently exhibits the highest  $MRE_{TriA}$  and  $MRE_{TriB}$  (even exceeding 0.5 in most instances). This is because AsgLDP does not optimize the structure of the synthetic graphs by incorporating triangle counts. While CAGM optimizes the structure of synthetic graphs by triangle counts, it still cannot achieve lower  $MRE_{TriA}$  and  $MRE_{TriB}$  than PrivAGM.