Privately Counting Triangles over Coupled Graphs

Anonymous Author(s)

ABSTRACT

XXXXXXX

ACM Reference Format:
Anonymous Author(s). 2023. Privately Counting Triangles over Coupled Graphs. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

How to count triangles in a graph? For over a decade, this seemingly fundamental and easy question has been a significant keypoint in graph data mining tasks. Triangle (i.e., cliques of three nodes with three edges), as one of the most simple but essential topological structures, is widely seen in different real-world graph data, including social networks[xx], citation networks[xx], and communication networks[xx]. Accurate counting of triangles is the basis of some structural-related measures, especially the transitivity ratio[xx], the cluster coefficients[xx], and the triangle connectivity[xx]. These measures are essential for applications in various domains, including protein motif prediction[xx], spam filtering [xx], friend recommendation[xx], and web page detection[xx], to name a few.

Previous works of triangle counting algorithms[xxx] have paid considerable attention to the accurate or approximate estimation of triangles over a streaming or a large-scale graph. However, in the past few years, people are gaining more awareness of their privacy since the occurrence of severe several data breach events(e.g., Facebook[xx], Uber[xx], and Nio[xx]) has uncovered the vulnerability of personal data that an ordinary netizen almost runs "naked" in cyberspace. To alleviate the privacy concerns associated with the data mining task over ubiquitous graph data, *Differential Privacy*(DP), served as a gold standard for data privacy, has been widely adopted in triangle counting[xxx].

In most DP settings, each user holds a local view of the edges between himself and other participants, whereas he has no information on the edges among participants. A data aggregator is responsible for collecting edges from n users and building a complete graph with n nodes, from which it can count the global triangle number. Before revealing data to the aggregator, users locally add noise to their edges; hence adversaries cannot infer a specific user's private information. However, in a more practical scenario, users' private edges are usually collected and protected by different parties(e.g., institutions, APPs, and mobile carriers).;i.e., each party holds a subgraph with a more extensive view of edges than a single user.

In this paper, we investigate the triangle counting problem in the context of secure multiparty computation, where there are m parties, each possessing a portion of the whole coupled graph. A classic case in point is the mobile communication network. The entire network often includes millions of users and is maintained by several mobile carriers. Each mobile carrier possesses the phone number of its users and is capable of detecting the communication among them as well as the phone call dialed to other carriers. All carriers wish to collaborate and derive the number of triangles embedded in the coupled graph. Meanwhile, they refuse to share the internal connections in their private subgraphs since accessibility to sensitive edges will likely trigger serious legal concerns for individuals and carriers. Therefore, in the view of a particular carrier, edges inside the subgraphs of other carriers remain invisible. It's trivial to count triangles in an overall graph with the formula $\frac{1}{6}Trace(A^3)$, where A refers to the adjacent matrix. However, in a coupled graph, each party i holds a local adjacent matrix A_i of the subgraph. Adding those distributed matrices directly to compute A^3 will cause certain privacy breaches.

Up to now, far too little attention has been paid to this problem. A preliminary work done by Do and Ng [xxx] novelly introduced the definition of the coupled graph (written as "distributed-edge graph" in their paper). Their main innovation, namely *Secure Multiparty Product of Matrix Sum*, is essentially a matrix product algorithm cooperatively run by participating parties to facilitate the computation of A^3, which falls into the category of cryptology. Based on the cryptographic secure matrix computation, they realized a triangle counting protocol and extended it to the 4-cycle counting problem for small and medium size graphs. Unfortunately, applying the algorithm to a large-scale graph is unfeasible because of its $O(mn^3)$ time complexity, where m and n represent the number of participating parties and vertices, respectively. According to their experiments, one single report of triangle count in a coupled graph (600 nodes) takes roughly 1 hour. This already costs a prohibitively long time, let alone the computation over a realistic communication network with a billion or million vertices.

Contributions We proposed secure multiparty computation (MPC) frameworks to handle the problem of triangle counting over large-scale coupled graphs. Our algorithms can be adapted to undirected and directed graphs with well-designed approaches. Detailed contributions are as follows:

- We propose multiparty computation protocols for the undirected graph to count the global triangle number and the local triangle number attached to a specific vertex.
- xxxx for directed graph.
- We evaluate our protocols on xx datasets. Experimental results show that our protocols dramatically reduce the communication overhead xx times more than the previous work[xx]. For instance, for a xxx nodes coupled graph distributed over xx parties, our undirected graph algorithm can answer the triangle number within xx seconds while preserving privacy with a relatively high data utility of xxx.

Our novelty

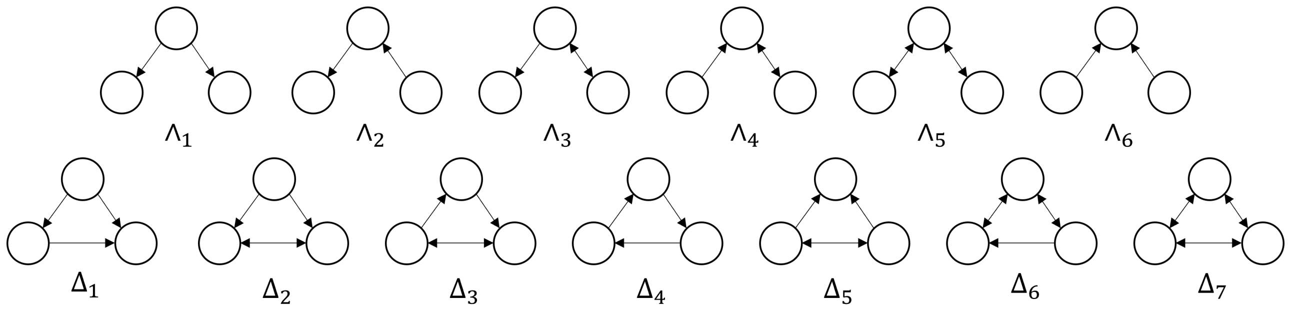

Figure 1: 13 classes of graphlets, including 7 types of directed triangles and 6 types of directed wedges.

2 FORMULATED PROBLEM

In this section, we first formally define coupled graphs. Then, we formulate the problem of privately counting triangles on coupled graphs studied in this paper.

2.1 Problem for Undirected Graphs

Definition of Undirected Coupled Graphs. An undirected graph $\mathcal{G} = (V, E)$ is distributed over m parties, and each party can only access its own data. The graph data accessible by party i is denoted by $\mathcal{G}_i = (V_i, V_i^*, E_i)$, where $V_i \subseteq V$ is the set of *internal nodes* possessed by party i, V_i^* is a set of *external border nodes* that party i's internal nodes connect to, i.e., $V_i^* = \{v^* : (v, v^*) \in E, v \in V_i, v^* \notin V_i\}$, and E_i is the set of *intra-edges* (i.e., edges between internal nodes) and *inter-edges* (i.e., edges from internal nodes to external nodes) accessible by party i. We call these subgraphs $\{\mathcal{G}_i\}_{i=1}^m$ as *coupled graphs*. For simplicity, we assume that different parties' internal nodes have no overlapping, i.e., $V_i \cap V_j = \emptyset$, $i \neq j$. Take the mobile phone network for an example. For a mobile carrier i, we denote by $\mathcal{G}_i$ represent its communication network, V_i the set of its customers, and V_i^* other carriers' customers that its customers communicated to. Clearly, mobile carrier i does not know the entire connections of any customer in V_i^*.

Problem of Private Triangle Counting. Our goal is to count triangles in graph $\{\mathcal{G}_i\}$ consisting of coupled graphs $\{\mathcal{G}_i\}_{i=1}^m$ under the constraint that party i can only access its own graph data $\mathcal{G}_i$.

2.2 Problem for Directed Graphs

Definition of Directed Coupled Graphs. We extend xxx.

3 OUR METHODS FOR UNDIRECTED GRAPHS

3.1 Global Triangle Counting

Basic Idea. Intuitively, in undirected graphs, there are only three possible triangular compositions of nodes: all three nodes belong to one party, two nodes from one party with one node belonging to another party, and all three nodes belong to three different parties (hereinafter the "overlapping triangle"). Each party can directly detect the former two cases. While the overlapping triangle appears as wedges(i.e., three nodes with two edges) in the subgraphs. For example, as shown in Fig. 2(b), party A can find $\Delta(v_4, v_7, v_5)$ as it can access the communication to the external border node $v5$. However, $\Delta(v7, v5, v8)$ is hidden towards party A because it can't observe the outlier edge $(v5, v8)$. Summation of those three cases of the triangle can correctly lead to a global triangle count.

Alg. 1: (Undirected graphs) Global triangle counting.

```
Input  : m parties with their local subgraphs $\{\mathcal{G}_i\}_{i=1}^m$.
Output : The total number of triangles in the underlying graph
         consisting of $\{\mathcal{G}_i\}_{i=1}^m$, denoted as $\Delta$.
1  foreach party $i = 1, \dots, m$ do
2      $W_i \leftarrow \emptyset$
       /* Function TriangleCnt($\mathcal{G}_i$) returns the number of
          triangles in graph $\mathcal{G}_i$.                          */
3      $\Delta_i \leftarrow$ TriangleCnt($\mathcal{G}_i$)
       /* Function WedgeCnt($\mathcal{G}_i$) returns the number of
          wedges($u, v, w$) in graph $\mathcal{G}_i$ of which nodes $u, v, w$ belong
          to three different parties respectively.                   */
4      $\wedge_i \leftarrow$ WedgeCnt($\mathcal{G}_i$)
5      foreach wedge $(u, v, w)$ in graph $\mathcal{G}_i$, where nodes $u, v, w$
         belong to three different parties respectively do
           /* Function Sort sorts input nodes according to their
              IDs in descending order.                               */
6          $(u^*, v^*, w^*) \leftarrow$ Sort($u, v, w$)
7          $W_i \leftarrow W_i \cup \{(u^*, v^*, w^*)\}$
8  All the m parties privately compute $t_1 \leftarrow \sum_{i=1}^m (\wedge_i + 2\Delta_i)$
   /* Function Cardi($S$) computes/estiamtes the cardinality of its
      input set $S$.                                                 */
9  All the m parties privately compute $t_2 \leftarrow$ Cardi($\cup_{i=1}^m W_i$)
10 Output $\Delta \leftarrow \frac{1}{2}(t_1 - t_2)$
```

We introduce a global triangle counting protocol for undirected graphs to implement the above idea. The pseudo-code is given in Algorithm 1.

Data collection. Let $\Delta_i = \texttt{TriCnt}(\mathcal{G}_i)$ denote the number of triangles locally detected in $\mathcal{G}_i = (V_i, V_i^*, E_i)$, where TriCnt is a triangle counting function based on matrix multiplication. We find that $\sum_{i=1}^m \Delta_i$ is the count of the first two cases. In Fig. 2, we have $\sum_{i=1}^3 \Delta_i = 3$. Measuring the number of overlapping triangles is the most challenging part of the algorithm. As a solution, each party first initializes an empty set W_i. Then, for each party i, formula $\wedge_i = WedgeCnt(\mathcal{G}_i)$ returns the number of wedge(u, v, w) it observes from $\mathcal{G}_i$, where its three nodes u, v, w belong to three different parties respectively. Each wedge is recorded as a triplet element (u, v, w) in set W_i. According to the input nodes' IDs, each (u, v, w) is sequenced in descending order by function Sort. For

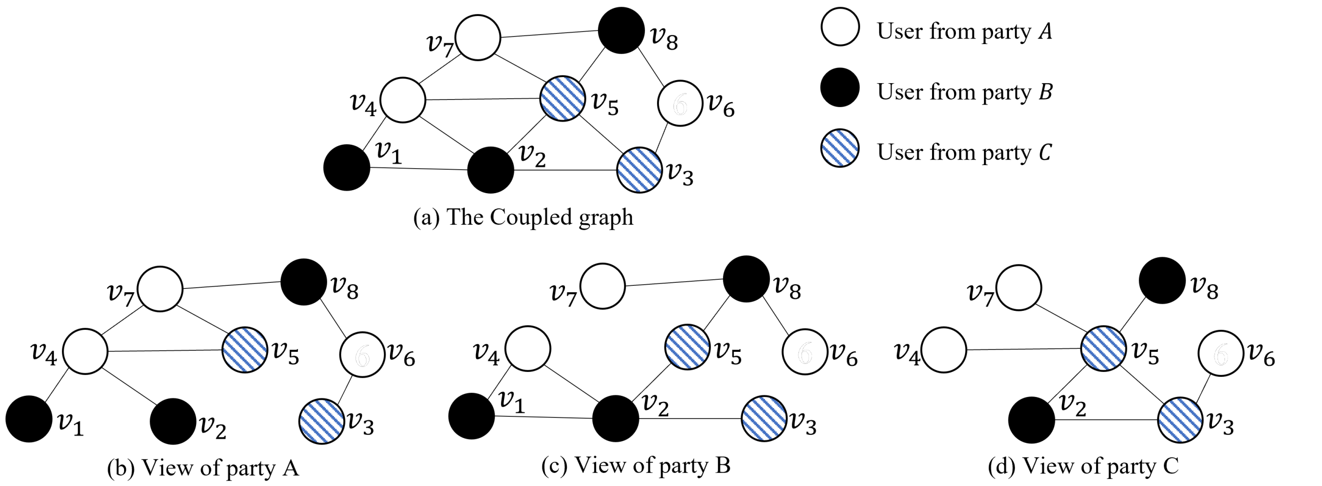

Figure 2: Overview of a simple coupled graph distributed over three parties. Nodes from different parties are labeled with different padding styles.

Alg. 2: (Undirected graphs) Local triangle counting for a node v.

```
Input  : node $v$, $m$ parties with their local subgraphs $\{\mathcal{G}_i\}_{i=1}^m$.
Output : The total number of triangles that include node $v$ in the
         underlying graph consisting of $\{\mathcal{G}_i\}_{i=1}^m$, denoted as $\Delta(v)$.
1 foreach party $i = 1, \ldots, m$ do
2     $W_i(v) \leftarrow \emptyset$
      /* Function TriCnt($\mathcal{G}_i, v$) returns the number of triangles
         that include node $v$ in graph $\mathcal{G}_i$.  */
3     $\Delta_i(v) \leftarrow$ TriCnt($\mathcal{G}_i, v$)
      /* Function WedgeCnt($\mathcal{G}_i, v$) returns the number of
         wedges($u, v, w$) in graph $\mathcal{G}_i$ of which nodes $u, v, w$ belong
         to three different parties respectively.  */
4     $\wedge_i \leftarrow$ WedgeCnt($\mathcal{G}_i, v$)
5     foreach wedge($u, v, w$) that includes node $v$ in $\mathcal{G}_i$, where its
      three nodes $u, v, w$ belong to three different parties respectively
      do
6         $(u^*, v^*, w^*) \leftarrow$ Sort($u, v, w$)
7         $W_i(v) \leftarrow W_i(v) \cup \{(u^*, v^*, w^*)\}$
8 All the $m$ parties privately compute
  $t_1(v) \leftarrow \sum_{i=1}^m (\wedge_i(v) + 2\Delta_i(v))$;   $t_2(v) \leftarrow$ Cardi($\cup_{i=1}^m W_i(v)$)
9 Output $\Delta(v) \leftarrow \frac{1}{2}(t_1(v) - t_2(v))$
```

example, in Fig. 2, party A reports

$$\wedge_A = 4, W_A = \{(5,4,1), \mathbf{(5,4,2)}, \mathbf{(8,7,5)}, (8,6,3)\}$$

party B reports

$$\wedge_B = 4, W_B = \{\mathbf{(5,4,2)}, (4,3,2), (8,6,5), \mathbf{(8,7,5)}\}$$

and party C reports

$$\wedge_C = 5, W_C = \{\mathbf{(8,7,5)}, (8,5,4), (7,5,2), \mathbf{(5,4,2)}, (6,3,2)\}$$

The intersection of the three sets shows that only $(5,4,2)$ and $(8,7,5)$ are reported at the same time, thus $\Delta(5,4,2)$ and $\Delta(8,7,5)$ must exist in the whole coupled graph. The global triangle number of Fig. 2(a) would be 5 in total. However, sharing of W_i will lead to data breaches.

Data aggregation. We refer to cardinality estimation to compute the number of intersections over m parties. To begin this process, each party directly publishes their triangle count Δ_i and wedge count $\wedge_i$ to a data curator, which will not leak any information about private edges. Curator then computes t_1 and t_2 as:

$$t1 = \sum_{i=1}^{m} (\wedge_i + 2\Delta_i), \quad t_2 = \texttt{Cardi}(\cup_{i=1}^m W_i),$$

where Cardi(S) represents a function collaboratively and securely estimating the cardinality of its input set S. In example Fig. 2, we have $t_1 = 19$ and $t_2 = 9$.

Global triangle estimation. Finally, we compute

$$\Delta = \frac{1}{2}(t_1 - t_2),$$

and reveal it as the overall triangle count. Essentially, intersection elements (e.g., $(5,2,4)$ and $(8,7,5)$) are accumulated three times by term $\sum_{i=1}^m \wedge_i$ in the computation of t_1, while the other elements are accumulated only once. t_2 privately estimate the cardinality of the union set gathered from m party that each distinct element will be calculated once. So $t_1 - t_2$ virtually contains twice the number of intersection elements, i.e., as we mentioned, the latter case of triangles.

3.2 Local Triangle Counting

Based on the above protocol, we further concentrate on counting the local triangle number of a specific node v. The pseudo-code of our local triangle counting protocol is given in 2.

Data collection. Much similar to 1, the first step for each party i is to initialize an empty set $W_i(v)$ to store the wedge triplets involving node v. Then, each party derives the local triangle number that includes node v with the below formula:

$$\Delta_i(v) = \texttt{TriCnt}(\mathcal{G}_i, v),$$

where $\mathcal{G}_i$ represents the local subgraph of party i. After computing $\Delta_i(v)$, we let each party i itemise the wedges(u, v, w) in graph $\mathcal{G}_i$

whose nodes u,v,w belong to three different parties respectively. This procedure could be illustrated as follows:

$$\wedge_i = \texttt{WedgeCnt}(\mathcal{G}_i, v).$$

Also, for each wedge(u, v, w) correlated to node v in $\mathcal{G}_i$, we sort its triplet element in descending order with the function Sort and preserve it in set $W_i(v)$.

Take node $v5$ in Fig. 2 for example, who belongs to party C and its corresponding triangle number should be 4. In subgraph $\mathcal{G}_A$, party A enumerates $\Delta_A(v5) = 1$ and $\wedge_A(v5) = 2$. Set $W_A(v5)$ records

$$W_A(v5) = \{\mathbf{(5,4,2)}, \mathbf{(8,7,5)}\}.$$

In subgraph $\mathcal{G}_B$, party B enumerates $\Delta_B(v5) = 0$ and $\wedge_B(v5) = 3$. Set $W_B(v5)$ records

$$W_B(v5) = \{\mathbf{(5,4,2)}, (8, 6, 5), \mathbf{(8,7,5)}\}.$$

In subgraph $\mathcal{G}_C$, party C enumerates $\Delta_C(v5) = 1$ and $\wedge_C(v5) = 4$. Set $W_C(v5)$ records

$$W_C(v5) = \{\mathbf{(5,4,2)}, (8, 5, 4), \mathbf{(8,7,5)}, (7, 5, 2)\}.$$

Data aggregation. Following the similar process in 3.1, we leverage cardinality estimation to count the intersection element of W. Let $\Delta(v)$ denote the number of triangles that include node v. Prior to the computation of $\Delta(v)$, we generate two variables $t_1(v)$ and $t_2(v)$ from formula:

$$t_1(v) = \sum_{i=1}^{m} (\wedge_i(v) + 2\Delta_i(v)),$$
$$t_2(v) = \texttt{Cardi}(\cup_{i=1}^{m} W_i(v)).$$

Then we construct our output as $\Delta(v) = \frac{1}{2}(t_1 - t_2)$ in local triangle counting protocol. More concretely, in the given example Fig. 2, we have

$$t_1(v5) = 13, \quad t_2(v5) = 5.$$

The final estimation is $\Delta(v5) = \frac{1}{2}(13 - 5) = 4$, which perfectly matches the ground truth value.

4 OUR METHODS FOR DIRECTED GRAPHS

Unlike undirected triangles, there are seven types of directed triangles and six types of directed wedges, as shown in Fig. 1. Following the analysis in Section 3.1, directed overlapping triangles in the coupled graph, whose nodes belong to three different parties, are undetectable in the distributed subgraphs, since each party only has access to one of the isomorphic directed wedge.

To give you an idea, we post a simple directed coupled graph in Fig. 3 as an intuition and will deduce our protocol over the given example. Each directed triangle has its own isomorphic directed wedge type. Detailed isomorphic relation is given in Table 1. We can obtain the number of directed overlapping triangles by gathering statistical wedge information from different parties.

In particular, assume that we have one overlapping Δ_2 ((v_8, v_7, v_5) in Fig. 3(a)) in the whole coupled graph, then there must exist two $\wedge_3$ ((v_8, v_7, v_5) in both Fig. 3(b) and Fig. 3(c)) and one $\wedge_6$ ((v_8, v_7, v_5) in Fig. 3(d)) in three distributed subgraphs. Nevertheless, this rule does not always hold conversely. That is, the existence of a directed wedge in a participating party does not necessarily result in an overlapping directed triangle in the coupled graph. To illustrate, although party A can detect $\wedge_6(v_5, v4, v1)$ in $\mathcal{G}_A$, there is no corresponding directed triangle in Fig. 3(a).

Alg. 3: (Directed graphs) Counting global 3-node graphlets.

```
Input  : $m$ parties with their local subgraphs $\{\mathcal{G}_i\}_{i=1}^m$.
Output : The total number of triangles in the underlying graph
         consisting of $\{\mathcal{G}_i\}_{i=1}^m$, denoted as $\Delta$.
1  foreach party $i = 1, \ldots, m$ do
2      $(W_{i,1}, \ldots, W_{i,7}) \leftarrow (\emptyset, \ldots, \emptyset)$
       /* Function DirTriCnt($\mathcal{G}_i$) returns the frequencies of 7
          different types of directed triangles appearing in
          graph $\mathcal{G}_i$.                                         */
3      $(\Delta_{i,1}, \ldots, \Delta_{i,7}) \leftarrow \texttt{DirTriCnt}(\mathcal{G}_i)$
       /* Function DirWedgeCnt($\mathcal{G}_i$) returns the frequencies of 6
          different types of directed wedges appearing in graph
          $\mathcal{G}_i$, whose nodes belong to three different parties,
          respectively.                                                  */
4      $(\wedge_{i,1}, \ldots, \wedge_{i,6}) \leftarrow \texttt{DirWedgeCnt}(\mathcal{G}_i)$
5      foreach wedge $(u, v, w)$ in graph $\mathcal{G}_i$, where its three nodes
         $u, v, w$ belong to three different parties respectively do
           /* Function WedgeType($\mathcal{G}_i, u, v, w$) returns the type of
              the wedge consisting of nodes $u, v, w$ in $\mathcal{G}_i$.  */
6          $t \leftarrow \texttt{WedgeType}(\mathcal{G}_i, u, v, w)$
           /* $S_t$ the set of directed triangles that include an
              induced subgraph isomorphic to the $t$-th type of
              wedge.                                                     */
7          foreach $j \in S_t$ do
8              $(u^*, v^*, w^*) \leftarrow \texttt{Sort}(u, v, w)$
9              $W_{i,j} \leftarrow W_{i,j} \cup \{(u^*, v^*, w^*)\}$
10 foreach $j = 1, \ldots, 6$ do
11     All the $m$ parties privately compute $y_j \leftarrow \sum_{i=1}^m \wedge_{i,j}$
12 foreach $j = 1, \ldots, 7$ do
13     All the $m$ parties privately compute $y_{j+6} \leftarrow \texttt{Cardi}(\cup_{i=1}^m W_{i,j})$
14 Solve the equation system by substituting $Y = (y_1, \ldots, y_{13})$ and
     yield the unknown variables $X = (x_1, \ldots, x_{13})$
15 $t_1 \leftarrow \sum_{j=7}^{13} x_j$
16 All the $m$ parties privately compute $t_2 \leftarrow \sum_{j=1}^{7} \sum_{i=1}^{m} \Delta_{i,j}$;
17 Output $\Delta \leftarrow t_1 + t_2$
```

Based on the linear correlation between directed triangle types and wedge types, we propose a novel protocol performing collaborative computation among different parties to count the directed triangle. Pseudo-code is shown in Algorithm 3.

Data collection. Prior to commencing the algorithm, each participating party i initializes a series of empty sets $(W_{i,1}, \ldots, W_{i,7})$ to record the directed wedge triplets it observed. We begin with the more accessible part. The number of 7 types of directed triangles in each subgraph $\mathcal{G}_i$, whose nodes belong to up to 2 parties, can be computed by the DirTriCnt function:

$$(\Delta_{i,1}, \ldots, \Delta_{i,7}) = \texttt{DirTriCnt}(\mathcal{G}_i),$$

In Fig. 3, each party records

$$(\Delta_{A,1}, \ldots, \Delta_{A,7}) = (0, 1, 0, 0, 0, 0, 0),$$
$$(\Delta_{B,1}, \ldots, \Delta_{B,7}) = (0, 0, 0, 0, 1, 0, 0),$$
$$(\Delta_{C,1}, \ldots, \Delta_{C,7}) = (0, 0, 0, 0, 0, 1, 0).$$

Then, we exploit function WedgeCnt to estimate the number of 6 different types of directed wedges:

$$(\wedge_{i,1}, \ldots, \wedge_{i,6}) = \texttt{WedgeCnt}(\mathcal{G}_i),$$

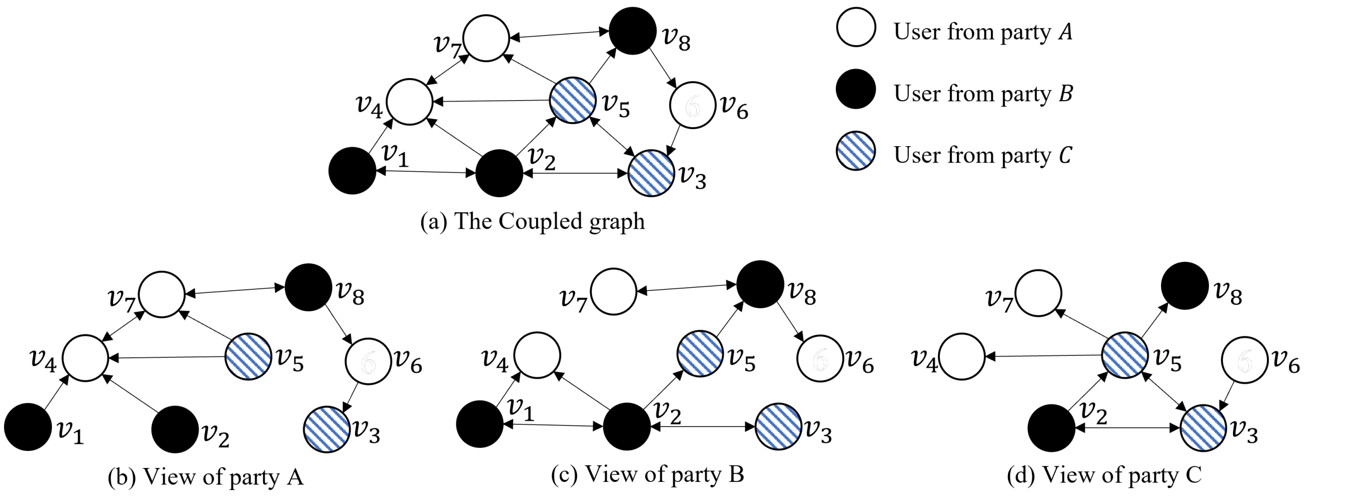

Figure 3: Overview of a simple directed coupled graph distributed over three parties. Nodes from different parties are labeled with different padding styles

where its three nodes attach to three different parties. In Fig. 3, each party records

$$(\wedge_{A,1},\ldots,\wedge_{A,6}) = (0,1,0,1,0,2),$$
$$(\wedge_{B,1},\ldots,\wedge_{B,6}) = (1,1,1,1,0,0),$$
$$(\wedge_{C,1},\ldots,\wedge_{C,6}) = (2,2,0,1,0,0).$$

Furthermore, for each directed wedge, the function `WedgeType` returns its type t. Hence we can find the directed triangle types that include an induced subgraph isomorphic to t. Clearly, in the view of party A, we have

$$\wedge_{A,2} \rightarrow (\Delta_1,\Delta_3,\Delta_4); \quad \wedge_{A,4} \rightarrow (\Delta_2,\Delta_3,\Delta_6); \quad \wedge_{A,6} \rightarrow (\Delta_1,\Delta_5).$$

In other words, party A detects three types of directed wedges: $\wedge_{A,2}$, $\wedge_{A,4}$, and $\wedge_{A,6}$, which indicates possible directed triangles in the coupled graph, i.e., $(\Delta_1,\Delta_3,\Delta_3)$, $(\Delta_2,\Delta_3,\Delta_6)$, and (Δ_1,Δ_5) respectively. In the view of party B, we have

$$\wedge_{B,1} \rightarrow (\Delta_1,\Delta_2); \quad \wedge_{B,2} \rightarrow (\Delta_1,\Delta_3,\Delta_4);$$
$$\wedge_{B,3} \rightarrow (\Delta_3,\Delta_5,\Delta_6); \quad \wedge_{B,4} \rightarrow (\Delta_2,\Delta_3,\Delta_6).$$

In the view of party C, we have

$$\wedge_{C,1} \rightarrow (\Delta_1,\Delta_2); \quad \wedge_{C,2} \rightarrow (\Delta_1,\Delta_3,\Delta_4); \quad \wedge_{C,4} \rightarrow (\Delta_2,\Delta_3,\Delta_6).$$

Therefore, we can record the sorted wedge triplets (u^*,v^*,w^*) using the corresponding set $W_{i,j}$ as follows:

$$W_{i,j} = W_{i,j} \cup (u^*,v^*,w^*).$$

In subgraph $\mathcal{G}_A$, party A reports

$$W_{A,1} = \{(v_5,v_4,v_2),(v_5,v_4,v_1),(v_8,v_6,v_3)\};$$
$$W_{A,2} = \{(v_8,v_7,v_5)\};$$
$$W_{A,3} = \{(v_8,v_7,v_5),(v_8,v_6,v_3)\};$$
$$W_{A,4} = \{(v_8,v_6,v_3)\};$$
$$W_{A,5} = \{(v_5,v_4,v_2),(v_5,v_4,v_1)\};$$
$$W_{A,6} = \{(v_8,v_7,v_5)\};$$
$$W_{A,7} = \emptyset.$$

In subgraph $\mathcal{G}_B$, party B reports

$$W_{B,1} = \{(v_5,v_4,v_2),(v_8,v_6,v_5)\};$$
$$W_{B,2} = \{(v_5,v_4,v_2),(v_8,v_7,v_5)\};$$
$$W_{B,3} = \{(v_4,v_3,v_2),(v_8,v_7,v_5),(v_8,v_6,v_5)\};$$
$$W_{B,4} = \{(v_8,v_6,v_5)\};$$
$$W_{B,5} = \{(v_4,v_3,v_2)\};$$
$$W_{B,6} = \{(v_4,v_3,v2),(v_8,v_7,v_5)\};$$
$$W_{B,7} = \emptyset.$$

In subgraph $\mathcal{G}_C$, party C reports

$$W_{C,1} = \{(v_8,v_7,v_5),(v_8,v_5,v_4),(v_7,v_5,v_2),(v_5,v_4,v_2)\};$$
$$W_{C,2} = \{(v_8,v_7,v_5),(v_8,v_5,v_4),(v_6,v_3,v_2)\};$$
$$W_{C,3} = \{(v_7,v_5,v_2),(v_5,v_4,v_2),(v_6,v_3,v_2)\};$$
$$W_{C,4} = \{(v_7,v_5,v_2),(v_5,v_4,v_2)\};$$
$$W_{C,5} = \emptyset;$$
$$W_{C,6} = \{(v_6,v_3,v_2)\};$$
$$W_{C,7} = \emptyset.$$

Data aggregation. We let each party i share its $(\wedge_{i,1},\ldots,\wedge_{i,6})$ representing the number of 6 types of directed overlapping wedges, which will not breach any edge information of their users. Then we compute the total number of each particular type j across m parties:

$$y_j = \sum_{i=1}^{m} \wedge_{i,j}.$$

More concretely, in Fig. 3, we have:

$$(y_1,\ldots,y_6) = (3,4,1,3,0,2).$$

To securely count the predictive overlapping directed triangles, for each directed triangle type j, we use function `Cardi` to estimate the length of the union set across m parties:

$$y_{j+6} = \texttt{Cardi}(\cup_{i=1}^{m} W_{i,j}).$$

In the given example, we have:

$$(y_7, \ldots, y_{13}) = (7, 4, 7, 4, 3, 3, 0).$$

Next, we substitute the whole vector $Y = (y_1, \ldots, y_{13})$ into the below equation system:

$$\begin{cases} x_1 + x_7 + x_8 = y_1 & (1) \\ x_2 + x_7 + x_9 + 3x_{10} = y_2 & (2) \\ x_3 + x_9 + 2x_{11} + x_{12} = y_3 & (3) \\ x_4 + 2x_8 + x_9 + x_{12} = y_4 & (4) \\ x_5 + x_{12} + 3x_{13} = y_5 & (5) \\ x_6 + x_7 + x_{11} = y_6 & (6) \\ x_1 + x_2 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} = y_7 & (7) \\ x_1 + x_4 + x_7 + x_8 + x_9 + x_{12} = y_8 & (8) \\ x_2 + x_3 + x_4 + x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12} = y_9 & (9) \\ x_2 + x_7 + x_9 + x_{10} = y_{10} & (10) \\ x_3 + x_6 + x_7 + x_9 + x_{11} + x_{12} = y_{11} & (11) \\ x_3 + x_4 + x_5 + x_8 + x_9 + x_{11} + x_{12} + x_{13} = y_{12} & (12) \\ x_5 + x_{12} + x_{13} = y_{13} & (13) \end{cases}$$

And solve it to yield the unknown variables $X = (x_1, \ldots, x_{13})$, which signifies the number of directed overlapping wedges and triangles of all types in the coupled graph.

To further explain this equation system, we divide it into two parts. The first part includes functions (1)–(6). Take equation (5) for instance, we assume the number of overlapping graphlets in the coupled graph is as follows:

$$(\wedge_5, \Delta_6, \Delta_7) \longrightarrow (x_5, x_{12}, x_{13}).$$

It's not difficult to find that all the above graphlets involve a uniform induced directed wedge type $\wedge_5$. Each will cause one $\wedge_5$ in one of the distributed subgraphs $\mathcal{G}_i$. So the total number of overlapping $\wedge_5$ in all distributed subgraphs will be

$$x_5 + x_{12} + 3x_{13} = y_5,$$

which equals y_5 observed by participating parties (the coefficient of Δ_7 is 3 because it's made of three $\wedge_5$). The second part of the equation system includes functions (7)–(8). We also provide a perspective by the example of the formula (8). Let $(x_1, x_4, x_7, x_8, x_9, x_{12})$ denote the number of overlapping graphlets $(\wedge_1, \wedge_4, \Delta_1, \Delta_2, \Delta_3, \Delta_6)$ in the coupled graph. Each graphlet is comprised of several directed wedges or triangles. For instance, Δ_2 can be decomposed into two $\wedge_4$ and one $\wedge_1$ in distributed subgraphs. When parties discover those wedges triplets, they record them as a possible triangle in the corresponding set $W_{i,1}$, $W_{i,2}$, $W_{i,3}$, and $W_{i,6}$ (see Table 1 for isomorphic relation). We enumerate all the $W_{i,2}$ triplets submitted by m parties, each of whom expects a possible Δ_2, and leverage function `Cardi` to remove the repeating items. Accordingly, the value of predictive triangles can be written as:

$$y_8 = \texttt{Cardi}(\cup_{i=1}^{m} W_{i,2});$$

$$x_1 + x_4 + x_7 + x_8 + x_9 + x_{12} = y_8.$$

In Fig. 3, we eventually achieve the unknown variables X as:

$$(x_1, \ldots, x_{13}) = (1, 3, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0),$$

which is the actual number of overlapping graphlets in the entire coupled graph. We sum up the overlapping directed triangles using the formula:

$$t_1 = \sum_{j=7}^{13} x_j = 2.$$

Together with the directed triangles straightforwardly detected by each party:

$$t_2 = \sum_{j=1}^{7} \sum_{i=A}^{m} \Delta_{i,j} = 3, \quad m = A, B, C.$$

Finally, we reveal the value of $\Delta = t_1 + t_2 = 5$.

Table 1: Isomorphic relation between directed triangles and directed wedges. For example, one Δ_5 with three nodes belonging to three parties in the coupled graph will lead to two $\wedge_3$ wedge types and one $\wedge_6$ wedge type in the distributed subgraphs.

	$\wedge_1$	$\wedge_2$	$\wedge_3$	$\wedge_4$	$\wedge_5$	$\wedge_6$
Δ_1	1	1				1
Δ_2	1			2		
Δ_3		1	1	1		
Δ_4		3				
Δ_5			2			1
Δ_6			1	1	1	
Δ_7					3	

5 EVALUATION

In this section, we first introduce our datasets and experimental settings.

5.1 Experimental Setup

- **Datasets.** XXXXXXXX
- **Baselines.** We compare the performance of our method, XXXX, with two baselines, XXXXX.
- **Metrics.** XXXXX.

5.2 Results

We XXXXXXXXXXX

6 RELATED WORK

Private triangle counting. Triangle counting is one of the most fundamental tasks in graph mining. Since 2010, massive and comprehensive publications [1–10] have been proposed to study the triangle counting problem in various scenarios. Under the premise of accuracy as high as possible, approximation algorithms [11–14] mainly consider improving the efficiency when the graph is extremely large. To accelerate the inference speed, some works [15–18] decompose the whole graph into portions and distribute them to different parallel GPUs since they excel in matrix operation. Aside from static data, in real-time graph data, protocols [19–23] are required to work in a streaming fashion, where communication becomes a significant limitation. Compared to the global triangle number, sometimes the number of triangles incident to a specific node in the entire graph is a more representative parameter for different tasks, such as community search [24–27] and clustering coefficient computation [28–31].

Recent developments in big data science have heightened the importance of personal privacy. However, in the above methods, direct operations of the adjacent matrix will expose sensitive information to the adversaries, such as intimate nodes and connections. This makes private triangle counting an under-explored area. To address this problem, differential privacy (DP) [32, 33], contributed as a de facto standard in data privacy, has made advancements in protecting sensitive knowledge about graphs. A large volume of published studies [28, 29, 34–41] have shaped the general procedures of DP: each user holds a local subgraph containing connections to the neighbor users; a data curator then collects those subgraphs and compute the final number of triangles. Before revealing the outputs, users or data curators will generate a carefully designed random coefficient according to the privacy setting, which can protect their data from being inferred by adversaries. For example, Ding et al. [12] proposed a graph projection method for large graphs and reached a relatively small upper bound of sensitivity. When releasing the number of triangles, their mechanism could achieve better accuracy while maintaining the requirement of DP. Hou et al. [42] combined fuzzy set theory with DP to formulate a new model via fuzzy similarity measures. Hence their protocols possessed a more flexible trade-off between utility and privacy-preserving levels. To fully leverage users' local knowledge, Sun et al. [35] assumed that each node holds an enlarged 2-hop local view and lowered the perturbance to the results. In a follow-up study, Liu et al. [36] described an extended algorithm called Edge-RLDP. In Edge-LDP, data curators could allocate distinct privacy budgets for participating users based on their mutuality. Unfortunately, this does not apply to online APPs (e.g., Tik Tok and Facebook) because users tend to change the software settings and forbid their friends to peep at their personal relationship networks.

In LF-GDPR [39], Ye et al. studied a more challenging privacy setting that each node merely knows the directly connected edges to the 1-hop neighbors. Specifically, they exploited Warner's Randomized Response [43] to the adjacent matrix, and therefore actual edges remain hidden from the curators. However, their approaches are far too inaccurate due to the large sensitivity of the triangle counting task. Although in [38], they tried to reach a more reasonable result by an approximation method, the estimation was still prohibitively biased. A lot of effort has been paid to smooth this obstacle. AsgLDP [37] illustrated an attributed graph-generation technique satisfying local LDP, which could count triangles while avoiding excessive noise injection. Imola et al. [28] found that two-round communication can take full advantage of users' local information and reach a state-of-the-art accuracy. Furthermore, in [29], they applied edge sampling and double clipping mechanisms to their original protocol, eventually reducing the download time from 6 hours to 8 seconds or less. With the help of a newly proposed shuffle model [44, 45], Imola et al. [40] solved the accurate triangle counting problem in one round of interaction. The main component of their methods is a wedge shuffling block, which enables privacy amplification of graph data and obtains a small estimation error (relative error $\ll 1$).

Multi-party computation. Our method falls into the Secure Multi-party computation (MPC) category, where a set of parties possesses a portion of the entire graph and invokes distributed computation to collaboratively compute the triangle number while ensuring correctness, privacy, and more. MPC is a typical cryptography method originated from [46]. Preliminary works, including Garbled Circuits [47], Secret Sharing [48], Private Information Retrieval[49], Partially Homomorphic Encryption[50], Fairplay [51], and Fully Homomorphic Encryption [52], mainly focus on the analysis of different security models but are generally inefficient and a long way from being practical. For the past few years, the issuance of data protection regulations in several countries and regions urged the need for data security under multi-party collaborative computation, To this end, more and more organizations (e.g., TF-Encrypted [53], CrypTen [54], MP-SPDZ [55], PJC [56], and Cerebro [57]) sought to address security issues of data exchange with MPC.

In recent years, based on the concept and theory of cryptology, MPC has emerged as a standard solution to achieving privacy protection for real-world applications [58]. For example, Burkart et al. [59] designed secure real-time protocols monitoring event correlation and aggregation of traffic statistics in a multi-domain network. Bogdanov et al. [60] introduced the Share Mind framework and Secrec pseudo-language model to analyze financial data from different banks safely. Doerner et al. [61] adapted the classic Gale-Shapley algorithm [62] to an MPC context on non-trivial inputs, which was previously considered computationally expensive and complex in memory access patterns. To compute statistics on the compensation of 166,705 employees across 114 companies, the Boston Women's Workforce Council deployed MPC [63] since companies refused to provide their raw data due to privacy concerns. Private set intersection (PSI) cardinality and private set union (PSU) cardinality are two special cases of MPC that have been extensively studied. In [64], Google proposed Private Set Intersection to compute accurate conversion rates from advertisements to actual purchases. Inspired by [65], Bay et al. [66] proposed a two-party protocol based on Bloom filter and threshold homomorphic public key encryption. The protocol significantly outperforms Kolesnikov's method [67], a temporal state-of-the-art MPC solution for cardinality, in both run time, computation and communication complexity. In the recent year, Ghosh and Simkin [68] circumvented the $O(d)$ lower bound of communication, where d refers to the size of the smallest input set. Before allowing parties communicating with each other, they estimate the scale of the intersection instead of counting the actual cardinality, and therefore the overall communication is sublinear to d. To further reduce the communication time, Badrinarayanan et al. [69] demonstrated three improved approaches using the TFHE [70]and TAHE [50] techniques, where the complexity only grows with the size of the intersection set rather the original input set. In [71], Kulshrestha et al. formalized a new variant of PSI, termed MPSIU-Sum, to securely analyze the incidental collection in Section 704 surveillance. MPC has also played a significant role in federated learning. The study in [72] proposed MPC protocols for secure linear regression models. This work extended [73] to high-dimensional data with over one million records and one hundred features. As follow, a longitudinal investigation [74] developed a multi-party data exchange system. Drawing on the linearly-homomorphic encryption concept, they could train a ridge linear regression model without compromising privacy. A recent study by Gu et al. [75] applied MPC to a popular language processing model known as BERT. They evaluated the pre-trained model in three downstream tasks: speaker identification, addressee recognition, and response selection.

MPC itself is also quickly evolving in the mechanism. Choudhuri et al. [76] mainly considered the scenario that existing MPC approaches require all parties to engage for the whole duration of the protocol. Their protocols allow participants to go offline or rejoin flexibly whenever they have available computational resources. In Conclave [77], researchers found that data scale excessively inhibits the application of MPC in real industry. Conclave can compile and accelerate queries in a parallel mode, thus improving the scalability of big data. In response to the trust issues caused by the breach of the Trusted Execution Environment (TEE), Wu et al.[78] built a generic framework for users with hybrid trusts to the TEE. Also, they introduced a new cryptographic model that captures different levels of trust perceived by various parties.

7 CONCLUSION

In this work, XXXXX compared with existing baselines. In the future, we will work to XXXXXXXXX.

REFERENCES

[1] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(3):1–28, 2010.

[2] Madhav Jha, Comandur Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 589–597, 2013.

[3] Kanat Tangwongsan, Aduri Pavan, and Srikanta Tirthapura. Parallel triangle counting in massive streaming graphs. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 781–786, 2013.

[4] John Kallaugher and Eric Price. A hybrid sampling scheme for triangle counting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1778–1797. SIAM, 2017.

[5] András Strausz, Flavio Vella, Salvatore Di Girolamo, Maciej Besta, and Torsten Hoefler. Asynchronous distributed-memory triangle counting and lcc with rma caching. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 291–301. IEEE, 2022.

[6] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th international conference on world wide web*, pages 1431–1440, 2017.

[7] Charalampos Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th international conference on world wide web*, pages 1122–1132, 2015.

[8] Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. Triest: Counting local and global triangles in fully dynamic streams with fixed memory size. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 11(4):1–50, 2017.

[9] Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. Sparse matrix multiplication and triangle listing in the congested clique model. *Theoretical Computer Science*, 809:45–60, 2020.

[10] Pedro Ribeiro, Pedro Paredes, Miguel EP Silva, David Aparicio, and Fernando Silva. A survey on subgraph counting: concepts, algorithms, and applications to network motifs and graphlets. *ACM Computing Surveys (CSUR)*, 54(2):1–36, 2021.

[11] Suman K Bera and C Seshadhri. How to count triangles, without seeing the whole graph. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 306–316, 2020.

[12] Xiaofeng Ding, Shujun Sheng, Huajian Zhou, Xiaodong Zhang, Zhifeng Bao, Pan Zhou, and Hai Jin. Differentially private triangle counting in large graphs. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[13] Xiangyang Gou and Lei Zou. Sliding window-based approximate triangle counting over streaming graphs with duplicate edges. In *Proceedings of the 2021 International Conference on Management of Data*, pages 645–657, 2021.

[14] Ata Turk and Duru Turkoglu. Revisiting wedge sampling for triangle counting. In *The World Wide Web Conference*, pages 1875–1885, 2019.

[15] Santosh Pandey, Zhibin Wang, Sheng Zhong, Chen Tian, Bolong Zheng, Xiaoye Li, Lingda Li, Adolfy Hoisie, Caiwen Ding, Dong Li, et al. Trust: Triangle counting reloaded on gpus. *IEEE Transactions on Parallel and Distributed Systems*, 32(11):2646–2660, 2021.

[16] Yang Hu, Hang Liu, and H Howie Huang. Tricore: Parallel triangle counting on gpus. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 171–182. IEEE, 2018.

[17] Mauro Bisson and Massimiliano Fatica. High performance exact triangle counting on gpus. *IEEE Transactions on Parallel and Distributed Systems*, 28(12):3501–3510, 2017.

[18] Lin Hu, Lei Zou, and Yu Liu. Accelerating triangle counting on gpu. In *Proceedings of the 2021 International Conference on Management of Data*, pages 736–748, 2021.

[19] Kijung Shin, Sejoon Oh, Jisu Kim, Bryan Hooi, and Christos Faloutsos. Fast, accurate and provable triangle counting in fully dynamic graph streams. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(2):1–39, 2020.

[20] Kijung Shin, Euiwoong Lee, Jinoh Oh, Mohammad Hammoud, and Christos Faloutsos. Cocos: Fast and accurate distributed triangle counting in graph streams. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(3):1–30, 2021.

[21] Andrew McGregor and Sofya Vorotnikova. Triangle and four cycle counting in the data stream model. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 445–456, 2020.

[22] Xu Yang, Chao Song, Mengdi Yu, Jiqing Gu, and Ming Liu. Distributed triangle approximately counting algorithms in simple graph stream. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 16(4):1–43, 2022.

[23] Jianqiang Huang, Haojie Wang, Xiang Fei, Xiaoying Wang, and Wenguang Chen. $tc-stream$ t c-s t r e a m: Large-scale graph triangle counting on a single machine using gpus. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):3067–3078, 2021.

[24] Konstantinos Sotiropoulos and Charalampos E Tsourakakis. Triangle-aware spectral sparsifiers and community detection. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1501–1509, 2021.

[25] Arnau Prat-Pérez, David Dominguez-Sal, and Josep-Lluis Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In *Proceedings of the 23rd international conference on World wide web*, pages 225–236, 2014.

[26] Jure Leskovec, Kevin J Lang, and Michael Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th international conference on World wide web*, pages 631–640, 2010.

[27] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, pages 1–8, 2012.

[28] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. Locally differentially private analysis of graph statistics. In *Proc. USENIX Security'21*, pages 983–1000, 2021.

[29] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. {Communication-Efficient} triangle counting under local differential privacy. In *Proc. USENIX Security'22*, pages 537–554, 2022.

[30] Shaikh Arifuzzaman, Maleq Khan, and Madhav Marathe. Fast parallel algorithms for counting and listing triangles in big graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(1):1–34, 2019.

[31] Hao Yin, Austin R Benson, and Jure Leskovec. The local closure coefficient: A new perspective on network clustering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 303–311, 2019.

[32] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28-June 1, 2006. Proceedings 25*, pages 486–503. Springer, 2006.

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

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

[35] Haipei Sun, Xiaokui Xiao, Issa Khalil, Yin Yang, Zhan Qin, Hui Wang, and Ting Yu. Analyzing subgraph statistics from extended local views with decentralized differential privacy. In *Proc. CCS'19*, pages 703–717, 2019.

[36] Yuhan Liu, Suyun Zhao, Yixuan Liu, Dan Zhao, Hong Chen, and Cuiping Li. Collecting triangle counts with edge relationship local differential privacy. In *Proc. ICDE'22*, pages 2008–2020. IEEE, 2022.

[37] Chengkun Wei, Shouling Ji, Changchang Liu, Wenzhi Chen, and Ting Wang. Asgldp: collecting and generating decentralized attributed graphs with local differential privacy. *IEEE Transactions on Information Forensics and Security*, 15:3239–3254, 2020.

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

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

[40] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. Differentially private subgraph counting in the shuffle model. *arXiv preprint arXiv:2205.01429*, 2022.

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

[42] Yongchao Hou, Xiaofang Xia, Hui Li, Jiangtao Cui, and Abbas Mardani. Fuzzy differential privacy theory and its applications in subgraph counting. *IEEE Transactions on Fuzzy Systems*, 2022.

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

[44] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2468–2479. SIAM, 2019.

[45] Vitaly Feldman, Audra McMillan, and Kunal Talwar. Hiding among the clones: A simple and nearly optimal analysis of privacy amplification by shuffling. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 954–964. IEEE, 2022.

[46] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[47] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*, pages 162–167. IEEE, 1986.

[48] Silvio Micali, Oded Goldreich, and Avi Wigderson. How to play any mental game. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*, pages 218–229. ACM New York, NY, USA, 1987.

[49] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[50] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*, pages 223–238. Springer, 1999.

[51] Dahlia Malkhi, Noam Nisan, Benny Pinkas, Yaron Sella, et al. Fairplay-secure two-party computation system. In *USENIX security symposium*, volume 4, page 9. San Diego, CA, USA, 2004.

[52] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[53] Tensorflow. Tf-encrypted. https://github.com/tf-encrypted/tf-encrypted, 2019.

[54] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973, 2021.

[55] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 1575–1590, 2020.

[56] Inc. Google and its affiliates. Private join and compute. https://github.com/google/private-join-and-compute, 2022.

[57] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. Cerebro: A platform for multi-party cryptographic collaborative learning. In *USENIX Security Symposium*, 2021.

[58] Sennur Ulukus, Salman Avestimehr, Michael Gastpar, Syed A Jafar, Ravi Tandon, and Chao Tian. Private retrieval, computing, and learning: Recent progress and future challenges. *IEEE Journal on Selected Areas in Communications*, 40(3):729–748, 2022.

[59] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. Sepia: privacy-preserving aggregation of multi-domain network events and statistics. In *Proceedings of the 19th USENIX conference on Security*, pages 15–15, 2010.

[60] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis: (short paper). In *Financial Cryptography and Data Security: 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers 16*, pages 57–64. Springer, 2012.

[61] Jack Doerner, David Evans, and Abhi Shelat. Secure stable matching at scale. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1602–1613, 2016.

[62] Lester E Dubins and David A Freedman. Machiavelli and the gale-shapley algorithm. *The American Mathematical Monthly*, 88(7):485–494, 1981.

[63] Andrei Lapets, Frederick Jansen, Kinan Dak Albab, Rawane Issa, Lucy Qin, Mayank Varia, and Azer Bestavros. Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, pages 1–5, 2018.

[64] Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. Private intersection-sum protocol with applications to attributing aggregate ad conversions. *Cryptology ePrint Archive*, 2017.

[65] Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. In *Information Security and Privacy: 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3–5, 2017, Proceedings, Part II 22*, pages 261–278. Springer, 2017.

[66] Aslı Bay, Zekeriya Erkin, Jaap-Henk Hoepman, Simona Samardjiska, and Jelle Vos. Practical multi-party private set intersection protocols. *IEEE Transactions on Information Forensics and Security*, 17:1–15, 2022.

[67] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829, 2016.

[68] Satrajit Ghosh and Mark Simkin. The communication complexity of threshold private set intersection. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part II*, pages 3–29. Springer, 2019.

[69] Saikrishna Badrinarayanan, Peihan Miao, Srinivasan Raghuraman, and Peter Rindal. Multi-party threshold private set intersection with sublinear communication. In *Public-Key Cryptography–PKC 2021: 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10–13, 2021, Proceedings, Part II*, pages 349–379. Springer, 2021.

[70] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38*, pages 565–596. Springer, 2018.

[71] Anunay Kulshrestha and Jonathan Mayer. Estimating incidental collection in foreign intelligence surveillance:{Large-Scale} multiparty private set intersection with union and sum. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1705–1722, 2022.

[72] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies*, 2017(4):345–364.

[73] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE symposium on security and privacy*, pages 334–348. IEEE, 2013.

[74] Irene Giacomelli, Somesh Jha, Marc Joye, C David Page, and Kyonghwan Yoon. Privacy-preserving ridge regression with only linearly-homomorphic encryption. In *Applied Cryptography and Network Security: 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings 16*, pages 243–261. Springer, 2018.

[75] Jia-Chen Gu, Chongyang Tao, Zhenhua Ling, Can Xu, Xiubo Geng, and Daxin Jiang. Mpc-bert: A pre-trained language model for multi-party conversation understanding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3682–3692, 2021.

[76] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid mpc: secure multiparty computation with dynamic participants. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II 41*, pages 94–123. Springer, 2021.

[77] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. Conclave: secure multi-party computation on big data. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–18, 2019.

[78] Pengfei Wu, Jianting Ning, Jiamin Shen, Hongbing Wang, and Ee-Chien Chang. Hybrid trust multi-party computation with trusted execution environment. In *The Network and Distributed System Security (NDSS) Symposium*, 2022.