# A Framework for Privacy Preserving Localized Graph Pattern Query Processing

LYU XU, Hong Kong Baptist University, China
BYRON CHOI*, Hong Kong Baptist University, China
YUN PENG*, Guangzhou University, China and Hong Kong Baptist University, China
JIANLIANG XU, Hong Kong Baptist University, China
SOURAV S BHOWMICK, Nanyang Technological University, Singapore

This paper studies privacy preserving graph pattern query services in a cloud computing paradigm. In such a paradigm, data owner stores the large data graph to a powerful cloud hosted by a service provider (SP) and users send their queries to SP for query processing. However, as SP may not always be trusted, the sensitive information of users' queries, importantly, the query structures, should be protected. In this paper, we study how to outsource the *localized graph pattern queries* (*LGPQ*s) on the SP side with privacy preservation. *LGPQ*s include a rich set of semantics, such as *subgraph homomorphism*, *subgraph isomorphism*, and *strong simulation*, for which each matched graph pattern is located in a subgraph called *ball* that have a restriction on its size. To provide privacy preserving query service for *LGPQ*s, this paper proposes the first framework, called Prilo, that enables users to privately obtain the query results. To further optimize Prilo, we propose Prilo* that comprises the first bloom filter for trees in the trust execution environment (*TEE*) on SP, a query-oblivious twiglet-based technique for pruning non-answers, and a secure retrieval scheme of balls that enables user to obtain query results early. We conduct detailed experiments on real world datasets to show that Prilo* is on average 4x faster than the baseline, and meanwhile, preserves query privacy.

CCS Concepts: • **Information systems → Query optimization**; • **Security and privacy → Management and querying of encrypted data**; *Privacy-preserving protocols*; Hardware-based security protocols.

Additional Key Words and Phrases: Localized graph pattern query, database outsourcing, trusted execution environment (*TEE*), query obliviousness

**ACM Reference Format:**
Lyu Xu, Byron Choi, Yun Peng, Jianliang Xu, and Sourav S Bhowmick. 2023. A Framework for Privacy Preserving Localized Graph Pattern Query Processing. *Proc. ACM Manag. Data* 1, 2, Article 129 (June 2023), 27 pages. https://doi.org/10.1145/3589274

## 1 INTRODUCTION

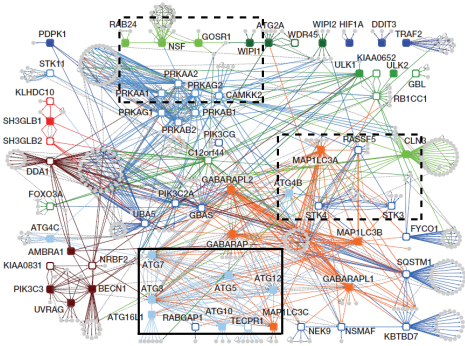
Graph pattern queries have been proposed in the literature (*e.g.*, [11, 38, 45]), and used in many recent applications, such as social network analysis, biology analysis, electronic circuit design, and chemical compound search [40, 43, 44, 53, 57]. On one hand, graph patterns often have high computational

*Corresponding Authors

Authors' addresses: Lyu Xu, Hong Kong Baptist University, Hong Kong, China, cslyuxu@comp.hkbu.edu.hk; Byron Choi, Hong Kong Baptist University, Hong Kong, China, bchoi@comp.hkbu.edu.hk; Yun Peng, Guangzhou University, Guang Zhou, China and Hong Kong Baptist University, Hong Kong, China, yunpeng@gzhu.edu.cn; Jianliang Xu, Hong Kong Baptist University, Hong Kong, China, xujl@comp.hkbu.edu.hk; Sourav S Bhowmick, Nanyang Technological University, Singapore, assourav@ntu.edu.sg.


2836-6573/2023/6-ART129 $15.00
https://doi.org/10.1145/3589274


(a) An autophagy pattern




(b) A *PPI* Network for Human Autophagy [5]

Fig. 1. Example of outsourcing *LGPQ* for biology analysis

complexities. On the other hand, different from matching the query to the whole data graph (*e.g.*, graph simulation [38]), graph pattern results that span through small subgraphs can be preferred, *e.g.*, in applications where humans would interpret the results. Hence, *localized graph pattern queries* (*LGPQ*), such as *subgraph homomorphism query* (*hom*) [26], *subgraph isomorphism query* (*sub-iso*) [11], and *strong simulation query* (*ssim*) [37], whose semantics require a size restriction on the matched patterns, have recently received much attention, *e.g.*, [17, 21, 42, 52].

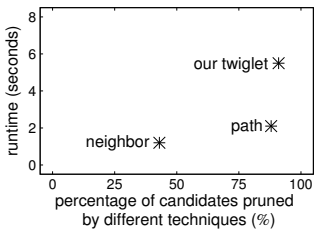
As *data owner*s and *query user*s may not always have the IT infrastructure to processing *LGPQ*s on the graph data, database outsourcing (such as to a *service provider* (SP) equipped with a cloud) has advantages to both of them, including elasticity, high availability, and cost savings. Database outsourcing inevitably has data privacy concerns. In particular, in the semi-honest model, the SP may infer sensitive information from both the queries and their processing. In Example 1, we illustrate the efficiency and query privacy challenges of this problem.

Example 1. *Consider a biotechnology company whose competitive advantages are its biological discoveries. The company has recently found a potentially valuable autophagy pattern of human cells, as shown in Fig. 1(a). Autophagy is* "the natural, conserved degradation of the cell that removes unnecessary or dysfunctional components through a lysosome-dependent regulated mechanism" *[27]. To explore the autophagy patterns with the same or similar structures as the found one, the company therefore uses an* LGPQ *to retrieve the subgraph data from an* SP*, who has a powerful IT infrastructure to host a publicly known large protein-protein interaction (*PPI*) network for autophagy interaction in human cells, as shown in Fig. 1(b). It may evaluate an* LGPQ *on small subgraphs, e.g., the solid box in Fig. 1(b). However, it neither evaluates the query on* all *the possible subgraphs* (*e.g., the dotted boxes in Fig. 1(b)*) *nor exposes the autophagy pattern (i.e., the query structure) to the* SP *side. Similar scenarios on other graphs, e.g., collaboration networks and social networks, can also be found [16, 36, 52].* □

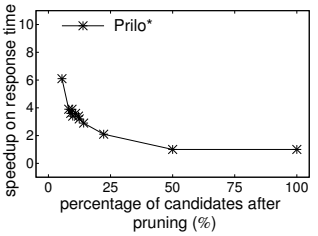
Existing works [16, 52] consider privacy preserving *LGPQ* under *individual* semantic, in particular, *sub-iso* queries [16] and *ssim* queries [52], and propose optimizations that focus on either minimizing the size of candidates to be matched or reducing the false positives of query results. Moreover, almost all existing works determine only the *existence* of matches in the data graphs. Except a trivial baseline [52], no previous work retrieves query matches as query results. In this paper, we take the first step towards the *first general* framework for finding the matches of *LGPQ*s with an SP. However, there are two main challenges.

*Challenge 1: To design general privacy preserving steps for* LGPQs*.*

To address this challenge, we propose a privacy preserving framework, called Prilo, that comprises three general steps, namely, *candidate enumeration*, *query verification*, and *query matching*. The




(a) Average performance of oblivious pruning by using different topologies: 3-hop neighbor's label [15], paths [52] and our twiglets of 4 labels




(b) Speedup on *Slashdot*: the ratio of Prilo*'s runtime (the time to obtain the first result from SP) to the runtime w/o Prilo*'s optimizations

Fig. 2. The performance of Prilo* from experiments

privacy preserving computation in these steps is done one ball at a time [37], where a ball is a subgraph of the graph defined by its center and radius, as the units (supersets) of *LGPQ* results. Since a ball can be much smaller than the whole graph, its privacy preserving processing can be efficient. The balls can also be precomputed, encrypted, and stored on the SP side.

To implement the privacy preserving processing on SP, we need to strike a balance between the efficiency and security of cryptography tools being used. In particular, it has been known that fully homomorphic encryptions (*FHE*) [18] and some partial homomorphic encryptions (*PHE*s), *e.g.*, *Paillier* [41], can be inefficient. We adopt an efficient symmetric encryption scheme called *cyclic group based encryption* (*CGBE*) [15] that is *CPA*-secure. In particular, to implement query verification, we use *CGBE* in a *query-oblivious* manner to detect the violations of the *LGPQ* semantics on the balls. Users then retrieve from SP the balls that have no violations. Users decrypt such balls, and compute query matches using existing algorithms on plaintext.

*Challenge 2: To propose privacy preserving optimizations for* Prilo.

It is evident that there can be *spurious balls* on SP, *i.e.*, balls that contain *no matches or duplicated matches*, and they should be *"pruned"*, *i.e.*, users *skip evaluating queries on them*. Hence, we propose Prilo* to optimize Prilo. In a nutshell, SP computes *pruning messages* to indicate whether balls are spurious or not, and users decrypt them to recognize non-spurious balls and skip spurious ones.

More specifically, the first technique in Prilo* is to exploit the *trusted execution environment* (*TEE*), *e.g.*, the *Intel software guard extensions* (SGX) [12]. Despite its popularity for ensuring its application's security, to our knowledge, it has not been exploited in privacy preserving *LGPQ*s. We propose to enumerate some small tree structures of the user queries for pruning inside the enclave of SGX. As it is known that SGX's enclave has a limited memory space, we propose to use bloom filters inside the enclave to compute pruning messages. The second technique is to propose a small structure called *twiglet* for query-oblivious pruning under the ciphertext domain, which does not need *TEE*. While previous work proposed simpler topologies for query-oblivious pruning, as shown in Fig. 2(a), twiglet can further enhance computing pruning messages at the expense of an additional runtime. To balance the pruning power and runtime, users can tune the size of twiglets.

The third technique optimizes the retrieval of non-spurious balls. We propose two kinds of SP servers (called Players and Dealer), and importantly, they enable that the non-spurious balls are securely returned to users early, while some dummy balls are also returned to make the ball retrieval patterns query-oblivious to SP. Then, users can compute all the matches early, as opposed to waiting the SP to finish its computation. From our preliminary experiments, we observe that on average, only 15% candidate balls contain matches. Specifically, on the SP side, a special server called Dealer uses pruning messages to generate sequences of balls mixed with dummy balls where non-spurious balls are placed in the front, in a secure way. The other servers called Players conduct *LGPQ* evaluation according to the sequences, without the knowledge of the balls' pruning messages.

These together result in the servers sending to users the balls that contain matches early, while query's privacy is preserved from SP. By using these optimization ideas, Prilo* can achieve a 4x speedup on *Slashdot* for the runtime for the users to obtain the first match pattern (Fig. 2(b)).

**Contributions.** The contributions of this paper are as follows.
• We propose the first *secure* general framework for *LGPQ*, called Prilo. Prilo has a unified encoding to encrypt the queries online. Prilo includes three general steps for processing *LGPQ*, namely candidate enumeration, query verification, and query matching.
• We propose an optimized framework called Prilo* that comprises i) a *bloom filter* checking that is the first to exploit *TEE* for pruning for *LGPQ*, ii) twiglets for query-oblivious pruning without the need of *TEE*, and iii) the first secure retrieval scheme that uses two kinds of servers in SP to return the evaluated results of non-spurious balls to users early for computing matched patterns, while existing works only check for their existence.
• We present the results of the privacy analyses and their corresponding proof sketches on Prilo*. For more detailed proofs, please refer to [51].
• Our experiments demonstrate that Prilo*'s pruning techniques outperform the SOTA on the pruning power with similar time cost and the query results are returned earlier than the baseline, in particular, 4x, 5x, and 8x faster than the baseline on *Slashdot*, *DBLP*, and *Twitter*, respectively. Our experiment with *LDBC* shows that Prilo is efficient for most of the queries and Prilo* furthers optimizes Prilo in 5 out of 10 queries, while Prilo and Prilo* exhibit similar performance in the other 5 queries.

**Organization.** The preliminaries and the problem statement are presented in Sec. 2. Sec. 3 presents the Prilo framework. Prilo*'s optimizations, the pruning techniques together with a secure scheme for ball retrieval, are presented in Sec. 4. Sec. 5 reports the privacy analysis. Sec. 6 reports the experimental results and Sec. 7 discusses the related work. This paper is concluded in Sec. 8.

## 2 PRELIMINARIES AND BACKGROUND

### 2.1 Notations of Graphs and Queries

This subsection presents some notations for describing *LGPQ*.
**Graph**. A *graph* is denoted by $G = (V_G, E_G, \Sigma_G, L_G)$, where $V_G$, $E_G$, $\Sigma_G$, and $L_G$ are the sets of vertices, directed edges, and labels, and the function for matching a vertex to its label, respectively. $(u, v)$ denotes the directed edge from $u$ to $v$, where $u, v \in V_G$, and $L_G(u)$ denotes the label of $u$. For graph $G$, the *distance* between $u$ and $v$ in $G$, denoted by $\mathsf{dis}(u, v)$, is the length of the shortest undirected paths from $u$ to $v$ in $G$ [37], and the *diameter* of $G$, denoted by $d_G$, is the largest distance between any pairs of vertices of $G$.

**Ball** [37]. A *ball*, denoted by $G[u, r]$, is a connected subgraph $B = (V_B, E_B, \Sigma_B, L_G, u, r)$ of graph $G$ which takes $u$ in $G$ as *center*, and $r$ as the *radius*, s.t. i) $V_B = \{v | v \in V_G, \mathsf{dis}(u, v) \leq r\}$, ii) $E_B$ has the edges that appear in $G$ over the same vertices in $V_B$, and iii) $\Sigma_B = \{L_G(v) | v \in V_B\}$.

**Adjacency matrix**. The *adjacency matrix* of graph $G$, denoted by $M_G$, is a $|V_G| \times |V_G|$ matrix. Given vertex $u$ (*resp.* vertex $v$) that locates in the $i^{\text{th}}$ row (*resp.* the $j^{\text{th}}$ column) of a matrix $M$, $M(i, j)$ is also denoted by $M(u, v)$ for simplicity. Then, $M_G(i, j) = 1$ if $(u, v) \in E_G$. Otherwise, $M_G(i, j) = 0$. The $i^{\text{th}}$ row vector of $M$ is denoted by $M(i)$. We may omit the subscript when it is clear from the context.

Some popular query semantics of localized graph pattern queries, namely, *subgraph homomorphism* (*hom*), *subgraph isomorphism* (*sub-iso*), and *strong simulation* (*ssim*), can be readily expressed by using *matrices* [15, 52]. We illustrate this with *hom* as follows.

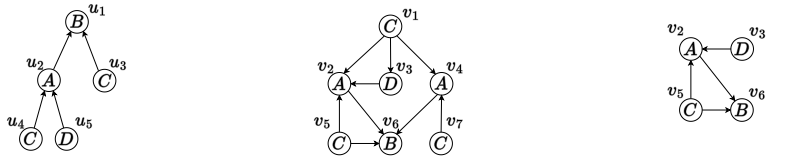



Fig. 3. An example for a *hom* subgraph in $G$ of $Q$

Definition 1. ***(Subgraph homomorphism (hom))*** *Given a connected query $Q$ and a graph $G$, a* subgraph homomorphism *of $Q$ in $G$ is a match function $\mathcal{H}: V_Q \rightarrow V_G$ that satisfies the conditions: (1) $\forall u \in V_Q, L_Q(u) = L_G(\mathcal{H}(u))$; and (2) $\forall u, v \in Q, M_Q(u, v) = 1 \Rightarrow M_G(\mathcal{H}(u), \mathcal{H}(v)) = 1$.* □

*Sub-iso* can be defined by modifying the match function $\mathcal{H}$ of Def. 1 with an injective function [15].[1] Due to space restrictions, we present the definition of *ssim* [52] in App. A.1 of the technical report [51]. Given a query semantic $\mathcal{F}$ (*e.g.*, *hom*, *sub-iso*, or *ssim*) and the vertex set $\{\mathcal{H}(v)|v \in V_Q\}$, an induced subgraph of the set in graph $G$ is called a *matching subgraph* for $Q$ under $\mathcal{F}$.

Example 2. *Consider the query $Q$ and the graph $G$ in Fig. 3. The induced subgraph of $\{v_2, v_3, v_5, v_6\}$ at the RHS of Fig. 3 is a matching subgraph of $G$ for $Q$ under* hom, *where $\mathcal{H}(u_1) = v_6$, $\mathcal{H}(u_2) = v_2$, $\mathcal{H}(u_3) = v_5$, $\mathcal{H}(u_4) = v_5$, and $\mathcal{H}(u_5) = v_3$.*

It is evident from Def. 1 and Example 2 that each matching subgraph for a *hom* query exists in a ball with a radius equal to $d_Q$. We are ready to give the definition of the query studied in the paper.

**Localized graph pattern query (*LGPQ*)**. Given a query semantic $\mathcal{F}$, a *localized graph pattern query* $Q = (V_Q, E_Q, \Sigma_Q, L_Q, \mathcal{F})$ on a graph $G$ is to find matching subgraphs for $Q$ under $\mathcal{F}$ for each ball $G[u, d_Q]$, where $u \in V_G$ and $\mathcal{F} \in \{hom, sub\text{-}iso, ssim\}$.

### 2.2 Background on Cryptosystem and Trusted Execution Environment

The security tools used in the technical presentation are as follows.

**Cyclic group based encryption (**CGBE**) [15]**. CGBE is a *CPA-secure* symmetric encryption scheme that supports the following homomorphic operations.

$$D(E(m_1) + E(m_2)) = m_1 \cdot r_1 + m_2 \cdot r_2 \; ; \; D(E(m_1) \cdot E(m_2)) = m_1 \cdot m_2 \cdot r_1 \cdot r_2,$$

where i) $m_1, m_2 \in \mathbb{Z}_p$, ii) $r_1$ and $r_2$ are two random numbers, and iii) $E(m)$ and $D(m)$ denote the encryption and decryption of message $m$, respectively. CGBE's homomorphic operations are used to design the privacy preserving solution by computation of encrypted messages. Note that CGBE requires $m_1 + m_2$ and $m_1 \cdot m_2$ are smaller than a large public prime $p$, or there are overflow errors [52].

**Trusted execution environment (*TEE*)**. Secure co-processors have recently been found efficient and effective in building secure applications. In particular, modern *Intel* CPUs have supported the software guard extensions (SGX) [12], a set of x86 instruction set architecture extensions, to construct a *TEE*. SGX provides users a secure and isolated hardware container called *enclave*. A secure channel is established between users and the enclave. A user encrypts the query and sends the encrypted query into the enclave for secure computation. The secure memory region in SGX is approximately 128 MB [2, 3, 50]. However, the cost of interaction with the enclave is huge that it is desirable to design space-efficient techniques when applying SGX for secure computation.

[1] In some existing works [26, 29], the match function $\mathcal{H}$ of *hom* (or *sub-iso*) also requires that the labels of the edges $(u, v)$ and $(\mathcal{H}(u), \mathcal{H}(v))$ are the same. For simplicity, we omit this requirement since it can be efficiently handled by transforming each edge $(u, v)$ into an intermediate vertex with $(u, v)$'s edge label.

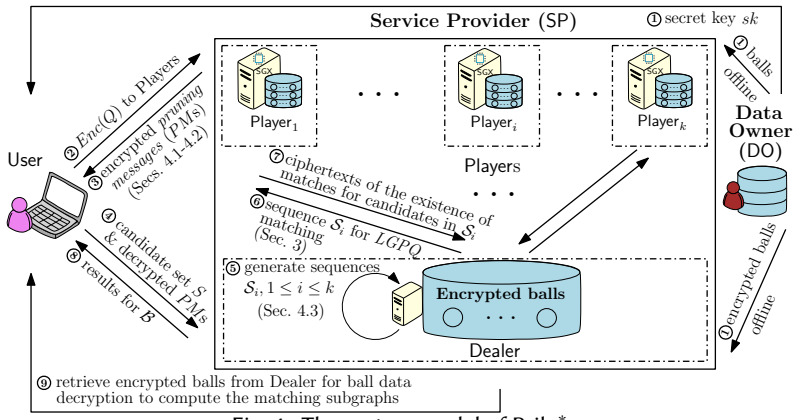



Fig. 4. The system model of Prilo*

## 2.3 Models and Problem Statement

This subsection presents the background of the system model and security model, and then presents our problem statement.

**System model**. We extend the commonly used system model on outsourced databases [13, 30, 49] that includes *data owner*, *user*, and *service provider*. Fig. 4 shows an overview of our system model.

• **Data owner (**DO**)**. A data owner first generates a secret key $sk$ and all balls of graph $G$ with various diameters offline. For each ball $B$ of $G$, data owner ① uses $sk$ to encrypt $B$ before sending $B$ or the encrypted $B$ to two different kinds of cloud servers on the service provider, respectively. Only authorized users can obtain $sk$.

• **User**. User ② sends to the service provider a query encrypted by using the private key $pk$ of CGBE. After ③ receiving the encrypted *pruning messages* (*PM*s) of candidate balls, User decrypts these *PM*s, and then ④ sends the decrypted *PM*s and ball identifier set $S$ of candidate balls to the service provider. After ⑧ receiving the ciphertext results about whether there exist matching subgraphs in candidate balls, User decrypts these results to find the target ball identifiers and then ⑨ retrieves the encrypted target balls from the service provider. Finally, User decrypts the encrypted balls with $sk$ and computes the matching subgraphs for the query.

• **Service provider (**SP**)**. We extend the SP of the widely used system model [13, 30, 49], to allow some optimizations enabled by SGX and to facilitate secure ball retrieval. Assume that SP consists of two kinds of servers, namely $k$ ($k \geq 2$) *player server*s (Players) equipped with SGX and a *dealer server* (Dealer), for ball retrieval. After ② receiving the encrypted query from User, the Players compute for each candidate ball $B$, under the ciphertext domain or inside SGX's enclave, the $PM$ that indicates whether $B$ may contain a match (a.k.a matching subgraph) of the query. Then, Players ③ send the *PM*s of candidate balls to User. After ④ receiving the set $S$ of candidate ball's identifiers and their decrypted *PM*s from User, Dealer ⑤ generates a sequence $\mathcal{S}_i$ of ball identifiers based on the set $S$ and *PM*s, and ⑥ sends $\mathcal{S}_i$ to Player$_i$, $1 \leq i \leq k$. For balls in $\mathcal{S}_i$, Player$_i$ run secure matching algorithms to generate a ciphertext result that indicates the existence of matching subgraphs. Player$_i$ ⑦ sends the ciphertext results of balls in $\mathcal{S}_i$ back to Dealer and then, Dealer ⑧ sends the ciphertext results of balls of $S$ to User.

**Security model**. This paper assumes the SP is honest but curious, *a.k.a.* the *semi-honest adversary model* [8, 9, 20, 33]. In a nutshell, SP performs the agreed computation protocol but may infer the private information. We made a mild assumption on SP. As shown in Fig. 4, there are multiple Players and a Dealer on SP. Unlike the existing multi-party computation (*MPC*) [6] where the servers communicate with others, Players only communicate with Dealer and Players do not

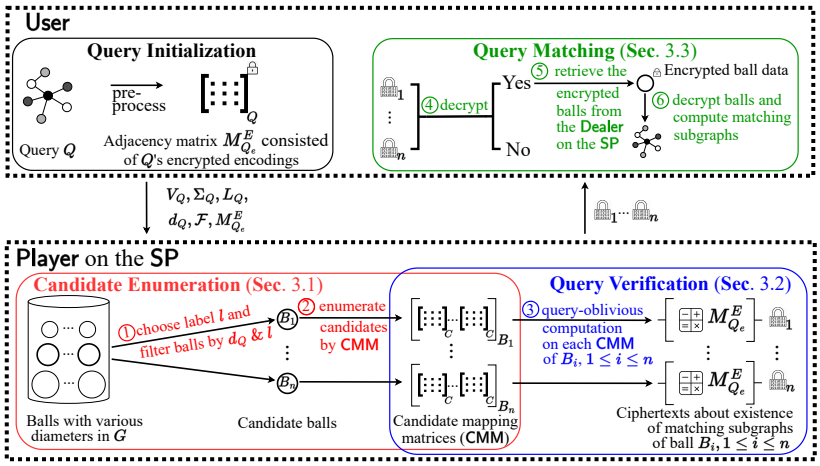



Fig. 5. Overview of the Prilo framework

collude with each other, which is similar to a recent work [32]. Regarding the communications between Players and Dealer, we adopt the commonly used collude-resistant model [13, 25, 34, 35] that Dealer and Players do not collude. Regarding the *attack model*, we assume the servers on SP adopt the *chosen plaintext attack* (*CPA*) [33], *i.e.*, the adversaries can choose arbitrary plaintexts to obtain their ciphertexts to gain sensitive information. The *privacy targets* of this paper are as follows.

- **Query privacy.** The structural information of User's query $Q$, *i.e.*, the values of elements in $M_Q$.
- **Access pattern privacy.** When querying on a graph, the access pattern privacy requires that the access pattern and the values of involved data during the computation process have no relations to the query privacy. That is, the computation process is *query-oblivious*.

**Problem statement**. *Assume the system and security models presented in Sec 2.3. Given an* LGPQ $Q = (V_Q, E_Q, \Sigma_Q, L_Q, \mathcal{F})$ *and a graph* $G$*, the goal is to compute all the subgraphs of* $G$ *that can be matched to* $Q$ *under* $\mathcal{F}$ *when preserving the privacy target.*

## 3 THE Prilo FRAMEWORK

In this section, we propose a general framework called Prilo for handling *LGPQ*s. Fig. 5 shows the overview of Prilo. An *LGPQ* can be answered by three generic steps, namely *candidate enumeration*, *query verification*, and *query matching*.

In the following subsections, we elaborate these steps with the *hom* query semantic, as an example.[2] For simplicity, we may use *ball* to refer to *candidate ball*, when it is clear from the context.

### 3.1 Candidate Enumeration

In this subsection, we present how the candidate enumeration step enumerates all candidate subgraphs of a ball in a query-oblivious manner. We first propose two propositions to filter redundant balls.

Proposition 1. *Given a query* $Q$ *with diameter* $d_Q$ *and label* $l$ *(*$l \in \Sigma_Q$*), for any subgraph* $G_s$ *of graph* $G$*, if* $G_s$ *is a matching subgraph for* $Q$ *under* hom*, there exists a vertex* $v$ *in* $G$ *such that i)* $L_G(v) = l$*, ii)* $G_s$ *is a subgraph of* $G[v, d_Q]$*, and iii)* $v \in G_s$*.*

[2]We remark that some *LGPQ* semantics may not require all three steps. *Sub-iso* can be extended with minor modifications on Sec. 3.1. *Ssim* is a special case that has a straightforward candidate enumeration step [37].

**Algorithm 1:** Candidate Enumeration Algorithm (*hom*)

**Input** : A query $Q$ with $V_Q$, $\Sigma_Q$, $L_Q$ and $d_Q$, and a ball $B = G[w, d_Q]$
**Output**: The set $R_1$ of CMMs of all $B$'s candidate subgraphs

**Procedure** CanEnum($V_Q$, $w$, $B$, $i$, $C$, $CV$):
1 **if** $i = 0$ **then**
2 $C \leftarrow 0$;
3 $(Q, B) \leftarrow$ opt$(Q, B)$; //opt():optimizations in [16]
4 **foreach** $u \in V_Q$ **do**
5 $CV(u) \leftarrow \emptyset$; //$CV(u)$:$B$'s vertices having label $L_Q(u)$
6 **foreach** $v \in V_B$ **do**
7 **foreach** $u \in V_Q$ **do**
8 **if** $L_Q(u) = L_B(v)$ **then**
9 $CV(u) \leftarrow CV(u) \cup \{v\}$;
10 **if** $i = |V_Q|$ **then**
11 **if** $\forall u \in V_Q, C(u, w) = 0$ **then**
12 **return** $\emptyset$; //$w$ cannot be matched to any vertices of $Q$
13 **return** $\{C\}$; //$C$ is a CMM
14 $R_1 \leftarrow \emptyset$;
15 $u \leftarrow$ the $(i+1)^{\text{th}}$ vertex in $V_Q$;
16 **foreach** $v \in CV(u)$ **do**
17 $C(u, v) \leftarrow 1$; //assign one 1 in the $(i+1)^{\text{th}}$ row
18 $R_1 \leftarrow R_1 \cup$ CanEnum$(Q, w, B, i+1, C, CV)$;
19 $C(u, v) \leftarrow 0$;
20 **return** $R_1$;

By Prop. 1, candidate enumeration can choose arbitrarily a label $l$ from $\Sigma_Q$ and consider as candidate balls *only* those balls having centers of label $l$ and diameters equal to $d_Q$ instead of all balls (① of Fig. 5). Then, we further derive Prop. 2.

Proposition 2. *Given a query $Q$, a label $l$ ($l \in \Sigma_Q$), and a ball $B = G[w, d_Q]$ of graph $G$, if there exists a subgraph $B_s$ of $B$ that $B_s$ is a matching subgraph for $Q$ under* hom *but $w \notin V_{B_s}$, there must exist a ball $B' = G[w', d_Q]$ of $G$ that i) $B_s$ is a subgraph of $B'$, ii) $w' \in V_{B_s}$, and iii) $L_G(w') = l$.*

Props. 1 and 2 is established by a simple proof by following the definition of the ball and the *LGPQ* semantic. The proofs are presented in App. A.2 of [51]. With Props. 1-2, we can enumerate for a ball $B$ only the *candidate subgraphs* that contains $B$'s center. Moreover, if $B$'s center cannot be matched to any vertices of the query, $B$ can be considered as a *spurious* ball. Even if there can be a matching subgraph of $B$ that does not contain $B$'s center, it can be found from other balls. We remark that Props. 1-2 can be also applied to *sub-iso* and *ssim*.

To illustrate the enumeration of all candidate subgraphs of $B$ for $Q$ (*e.g.*, a *hom* query), we introduce the *candidate mapping matrix* to represent the match function $\mathcal{H}$ that matches $V_Q$ to $V_{B_c}$, where $B_c$ is a candidate subgraph of $B$.

Definition 2. ***(Candidate mapping matrix (***CMM***))*** *A* candidate mapping matrix *from a query $Q$ to a graph $G$, denoted by $C$, is a $|V_Q| \times |V_G|$ matrix that $\forall\, u \in V_Q$, $\exists\, v \in V_G$ satisfies i) $C(u, v) = 1$, ii) $L_Q(u) = L_G(v)$, and iii) $\forall\, w \in V_G - \{v\}$, $C(u, w) = 0$.*

Example 3. *Consider the query $Q$ and graph $G$ in Fig. 3 and the match function $\mathcal{H}$ in Example 2. We locate vertex $u_i$, $1 \le i \le 5$ (resp. $v_j$, $1 \le j \le 7$) of $Q$ (resp. $G$) on the $i^{th}$ row (resp. the $j^{th}$ column) of the* CMM*. Then, $\mathcal{H}$ can be represented by the* CMM *$C$ that $C(u_1) = (0, 0, 0, 0, 0, 1, 0)$, $C(u_2) = (0, 1, 0, 0, 0, 0, 0)$, $C(u_3) = (0, 0, 0, 0, 1, 0, 0)$, $C(u_4) = (0, 0, 0, 0, 1, 0, 0)$, and $C(u_5) = (0, 0, 1, 0, 0, 0, 0)$, where $C(u_i, u_j) = 1$ represents matching $u_i$ to $v_j$.*

Then, we present the algorithm to enumerate all CMMs of candidate subgraphs. Taking a query $Q$ with $V_Q$, $\Sigma_Q$, $L_Q$ and the diameter $d_Q$, and a ball $B$ with the center $w$ and radius $d_Q$ as inputs, Alg. 1 returns the set $R_1$ of CMMs of all $B$'s candidate subgraphs for *hom* queries as output. In Line

**Algorithm 2:** Query Verification Algorithm (*hom*)

**Input :** The encodings $M_{Q_e}$ of query $Q$'s adjacency matrix, the adjacency matrix $M_G$ for graph $G$ and a CMM $C$ for matching $V_Q$ to $V_G$

**Output :** An integer without factor $q$ if $C$ represents a valid match function under *hom* or a multiple of $q$, otherwise.

**Procedure** `Verify`($M_{Q_e}, M_G, C$):

```
1 r ← 1; //result initialization
2 M_p ← C · M_G · C^T; //M_p : G's adjacency matrix projected by C
3 foreach i ∈ [1, |V_Q|] do
4 |   foreach j ∈ [1, |V_Q|] do
5 |   |   if M_p(i, j) = 0 then
6 |   |   |   r ← r · M_Qe(i, j); //matching violation aggregation
7 return r;
```

2, CMM $C$ is initialized as a zero matrix. The optimizations [16] for minimizing the size of query $Q$ (*resp.* ball $B$) on User (*resp.* Player) are applied in Line 3. $\forall u \in V_Q$, Lines 6-9 generate a vertex set $CV(u)$ that contains the vertices of $B$ having the same label as $u$'s label by comparing $L(u)$ with $L(v)$, $v \in V_B$. Then, Line 14 initializes a CMM set $R_1$ as an empty set. Assume that vertex $u$ of $Q$ locates in the $i^{\text{th}}$ row, $1 \leq i \leq |V_Q|$. Lines 16-17 enumerate all the possible mappings from vertices of $B$ to $u$ by assigning value 1 in different columns in the $i^{\text{th}}$ row. Line 18 recursively calls Alg. 1 for the $(i+1)^{\text{th}}$ row's enumeration. If each row of $C$ has been assigned with a value 1 (Line 10), Line 13 returns $C$ as a CMM and then, Line 18 adds $C$ into $R_1$. In particular, Line 11 checks in matrix $C$ whether $B$'s center $w$ is mapped to any vertices of $Q$. If it is not, $C$ is not a CMM of a candidate subgraph so that Line 12 returns an empty set. Finally, Line 20 returns $R_1$ as output.

Example 4. *We illustrate Alg. 1 with the generation of the* CMM $C$ *in Example 3. Lines 1-5 are initialization. For each query vertex u, Lines 6-9 obtain the vertex set* $CV(u)$*, i.e.,* $CV(u_1) = \{v_6\}$, $CV(u_2) = \{v_2, v_4\}$, $CV(u_3) = CV(u_4) = \{v_1, v_5, v_7\}$, *and* $CV(u_5) = \{v_3\}$. *Lines 16-18 set* $C(u_1, v_6) = 1$, $C(u_2, v_2) = 1$, $C(u_3, v_5) = 1$, $C(u_4, v_5) = 1$, *and* $C(u_5, v_3) = 1$ *in turn to obtain C. Lines 10-13 return C as a* CMM *and Line 18 adds C into the set of all* CMM*s.*

**Analysis.** Alg. 1 is query-oblivious since its execution is only dependent of the vertex set $V_Q$ but independent of the edge set $E_Q$. The detailed proof is presented in App. A.2 of [51]. For the time complexity, Lines 6-9 take $O(|V_Q| \cdot |V_B|)$ time. Lines 16-19 enumerate $O(\sum_{v \in V_Q} |CV(v)|^{|V_Q|})$ CMMs. The optimizations (Line 3) take negligible time compared to the whole enumeration process.

### 3.2 Query Verification

In this subsection, we present how to design a query-oblivious algorithm that verifies whether a CMM represents a valid match function under an *LGPQ* semantic. The main ideas are as follows.

Given a query $Q$ with an *LGPQ* semantic $\mathcal{F}$, and a candidate subgraph $G_c$ of graph $G$, the verification for matching $V_Q$ to $V_{G_c}$ under $\mathcal{F}$ is to detect no *matching violation*, *i.e.*, the unsatisfaction conditions w.r.t. $\mathcal{F}$'s definition. For example, for *hom*, a matching violation can be detected if there exists at least one edge $e$ in $Q$ that no edges in $G_c$ can be matched to $e$ (unsatisfaction on condition (2) of Def. 1). To detect the existence of such edges in $Q$ on the Player side, we encode the existence of edges in $M_Q$ as follows. We remark that this encoding is also applicable for *sub-iso* and *ssim* queries.

**Encoding of $M_Q$ ($M_{Q_e}$).** $\forall\, i, j \in [1, |V_Q|]$,

$$M_{Q_e}(i,j) = \begin{cases} q, & \text{if } \overline{M_Q(i,j)} = 0; \text{ and} \\ 1, & \text{otherwise,} \end{cases}$$

where $q$ is a large prime number and $\overline{M_Q(i,j)} = 1 - M_Q(i,j)$. Then, we present the query-oblivious verification algorithm for *hom*. (*sub-iso* can be supported with a minor modification [16].)

Taking as inputs the encodings $M_{Q_e}$ of query $Q$'s adjacency matrix, the adjacency matrix $M_G$ of graph $G$, and a CMM $C$ for matching $V_Q$ to $V_G$, Alg. 2 returns an integer without factor $q$ if $C$ represents a valid match function under *hom*. Otherwise, an integer with factor $q$ is returned. In detail, Line 2 first computes the $|V_Q| \times |V_Q|$ adjacency matrix $M_p$ of $G$ that projects the vertices of $G$ onto the vertices of $Q$ according to $C$, where $C^{\mathrm{T}}$ denotes the transpose matrix of $C$. For each encoding $M_{Q_e}(i, j)$ (Lines 3-4), if $M_p(i, j) = 0$ (Line 5), it may lead to a matching violation on condition (2) of Def. 1 and Line 6 uses multiplication to aggregate $M_{Q_e}(i, j)$ into a result $r$. Line 7 returns $r$ as output.

Example 5. *Consider the query $Q$ and graph $G$ in Fig. 3 and the* CMM *$C$ in Example 3. The encoding $M_{Q_e}$ of $M_Q$ is as follows: $M_{Q_e}(u_1) = (1, 1, 1, 1, 1)$, $M_{Q_e}(u_2) = M_{Q_e}(u_3) = (q, 1, 1, 1, 1)$, and $M_{Q_e}(u_4) = M_{Q_e}(u_5) = (1, q, 1, 1, 1)$. In Alg. 2, Line 1 first set the value of $r$ as* 1*. By using $C$ and $M_G$, Line 2 computes the projected adjacency matrix $M_p$: $M_p(u_1) = (0, 0, 0, 0, 0)$, $M_p(u_2) = (1, 0, 0, 0, 0)$, $M_p(u_3) = M_p(u_4) = (1, 1, 0, 0, 0)$, and $M_p(u_5) = (0, 1, 0, 0, 0)$. For any $i, j$ such that $M_p(i, j) = 0$ (i.e., $M_p(u_i, u_j) = 0$), Lines 3-6 aggregate $M_{Q_e}(i, j)$ into $r$ by multiplication. Line 7 returns an $r$ with value* 1*, which means that $C$ represents a valid match function for $Q$ under* hom*. Otherwise, a multiple of $q$ is returned indicating that $C$ does not represent a valid match function.*

**Analysis.** For any vertex $u_i$ of the query $Q$ and any vertex $v_j$ of the graph $G$, we list all the possible cases of matching $u_i$ to $v_j$. The correctness of Alg. 2 can be proved by summarizing its outputs of the matching cases that does not satisfy the requirements of the definitions of *LGPQ* semantics. Moreover, Alg. 2 is query-oblivious since its execution (Lines 2-5) depends on $C$ and $M_G$, which are independent of the edge set $E_Q$. The detailed proofs are presented in App. A.2 of [51]. The encoding of $M_Q$ can *be encrypted* by CGBE to preserve the query privacy while the query-obliviousness of Alg. 2 and the homomorphic computation supported by CGBE preserves the access pattern privacy. For the time complexity, assume both the addition and multiplication take $O(1)$ time. Then, the matrix multiplication in Line 2 takes $O(|V_G|^3)$ time and Lines 3-6 take $O(|V_Q|^2)$ time. In practice, $|V_G|$ equals the size of each candidate ball, which is limited by the diameter $d_Q$ of the query.

### 3.3 Query Matching

In this subsection, we present the query matching step by the overall algorithm (Alg. 3) of the Prilo framework, as shown in Fig. 5.

• **On** Player**s.** Taking a query with $V_Q, \Sigma_Q, L_Q$, diameter $d_Q$ and the adjacency matrix $M^E_{Q_e}$ consisted of $Q$'s encrypted encodings, and all balls of graph $G$ as inputs, Alg. 3 outputs the matching subgraphs of $G$ for $Q$. Recall that the candidate enumeration can choose arbitrarily a label $l$ from $\Sigma_Q$ and consider as candidate balls only those balls having centers of label $l$. Hence, as a simple optimization, Player ① first chooses a label $l$ that maximizes the number of candidate balls in Line 2 after receiving the query $Q$ from User. Then, Player ① filters balls of graph $G$ by $d_Q$ and $l$ (Lines 3-4). For each candidate ball $B$, CMMs of all $B$'s candidate subgraphs are ② enumerated in Line 5 (Sec 3.1) and ③ verified in Lines 6-7 (Sec. 3.2). Player sends to User the sets $R$s of ciphertext results for all candidate balls of $G$ in Line 9. We remark that the evaluations of balls are independent of each other and hence, can be readily parallelized.

• **On** User**.** The query matching step is as follows. User ④ decrypts the ciphertexts in the received ciphertext result set $R$ (Lines 11-12). For each ciphertext $r_i$ in $R$ of ball $B_i$, if the decrypted $r_i$ contains a factor $q$, $B_i$ does not contain matching subgraphs for $Q$. Otherwise, User ⑤ retrieves the encrypted data of $B_i$ from Dealer (Line 13) and decrypts $B_i$'s data by using the secret key $sk$ sent from DO (Line 14). Finally, User ⑥ computes the matching subgraphs of the retrieve balls for $Q$ under the plaintext domain (Line 15) (*e.g.*, using any current state-of-the-art algorithms [23, 24, 37]) and outputs these subgraphs as query answers (Lines 16-17).

**Algorithm 3:** Overall Algorithm of Prilo (*hom*)

**Input :** A query $Q$ with $V_Q, \Sigma_Q, L_Q$, diameter $d_Q$ and matrix $M^E_{Q_e}$ consisted of $M_Q$'s encrypted encodings, and all balls of graph $G$

**Output :** The matching subgraphs of $G$ for $Q$

**On the** Player **side:**

1 $R \leftarrow \emptyset$;
2 $l \leftarrow \arg\max_{l' \in \Sigma_Q} \{|\{v | v \in V_G \text{ and } L_G(v) = l'\}|\}$; //opt: choose label $l$
3 **foreach** $v_i \in \{v | v \in V_G \text{ and } L_G(v) = l\}$ **do** // ①:filter balls by $l$
4 $B_i \leftarrow G[v_i, d_Q]$, $r_i \leftarrow 0$; //①:filter balls by $d_Q$
5 $CS_i \leftarrow \texttt{CanEnum}(Q, v_i, B_i, 0, 0, \emptyset)$; //②:candidate enumeration
6 **foreach** $C \in CS_i$ **do**
7 $r_i \leftarrow \texttt{Verify}(M^E_{Q_e}, M_{B_i}, C) + r_i$; //③:query verification
8 $R \leftarrow R \cup \{r_i\}$;
9 send $R$ to User;

**On the** User **side after** $R$ **is received from** Players**:**

10 $R_H \leftarrow \emptyset$;
11 **foreach** $r_i \in R$ **do**
12 **if** the decrypted $r_i$ does not have factor $q$ **then** // ④:decrypt ciphertexts
13 retrieve the encrypted ball $B_i$ from SP (Dealer); //⑤:retrieve ball
14 decrypt $B_i$'s data by using the $sk$ sent from DO; //⑥:decrypt ball
15 compute matching subgraphs $B'_i$s of $B_i$ for $Q$; //⑥:compute matches
16 $R_H \leftarrow R_H \cup \{B'_i\text{s}\}$;
17 **return** $R_H$;

Example 6. *Alg. 3 computes the match function $\mathcal{H}$ in Example 3 as follows. Line 2 chooses B as the label of l. Lines 3-8 evaluate those balls that i) have centers of label B, and ii) have diameters equal to $d_Q = 3$. Line 5 enumerates all the* CMMs *of ball $B = G[v_6, 3]$ by using Alg. 1. To compute $r_i$ that indicates whether B contains valid match functions under* hom*, Lines 6-7 aggregate into $r_i$ the outputs of Alg. 2 by addition for all the* CMMs *of B. In Example 5, Alg. 2's output for the* CMM *of $\mathcal{H}$ is* 1*. When taking as input the matrix $M^E_{Q_e}$ of query Q consisted of $M_Q$'s encrypted encodings but not the encodings $M_{Q_e}$, the output of $\mathcal{H}$ is $r^n$, where r (resp. n) denotes the random value of $M_{Q_e}(i, j)$ (resp. the number of multiplication) in Line 6 of Alg. 2. Therefore, there exists a decrypted $r_i$ in Line 12 that does not have factor q, which encodes* 0 *(Sec. 3.2). On the* User *side, Line 13 retrieves the encrypted data of B, Line 14 decrypts the data, and Line 15 computes the matching subgraphs.*

**Analysis.** For the query matching step, although Dealer knows the specific balls retrieved by User (Line 13), Dealer cannot infer the edge information of the query from the encrypted ball data and hence, the query matching step is query-oblivious.

## 4 THE OPTIMIZED Prilo FRAMEWORK

To optimize Prilo, we propose an optimized framework called Prilo* that i) enables Players to detect as many as possible the *spurious* balls when preserving the privacy target and record them in the *pruning message*s (*PM*s) of balls (Secs. 4.1-4.2), and ii) enables User to *early* obtain the balls contain matching subgraphs by using these *PM*s (Sec. 4.3).

### 4.1 Bloom Filter of Trees in *TEE* (BF)

Different from existing works [16, 52] that compute the *PM*s by using simple graph topologies (*e.g.*, *neighbors* and *paths*), our first pruning technique, called BF, uses the *tree* topology. As reported in an analytical study of graph queries [7], for vertices of most queries, the average degree is smaller than 4 and the maximal degree is not larger than 5. Hence, we propose the *h-label binary trees* for detecting spurious balls.

Definition 3. *(**h-label binary tree**) Given a height h, a graph G, and a vertex u of G, the* h-label binary tree *$T_{u,h}$ of G is a binary tree projected by the labels of nodes of an* undirected *binary subtree*

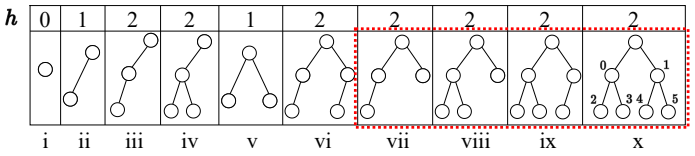



Fig. 6. $h$-label binary trees ($h \leq 2$) (labels omitted)

*of $G$ such that i) $u$ is its root, ii) $h$ is its height, and iii) For any two vertices $v$ and $v'$ of this subtree, $v \neq v' \Rightarrow L_G(v) \neq L_G(v')$.*

We apply the undirected tree topology to detect more spurious balls since there may exist few directed trees in the queries having small sizes. Fig. 6 shows 10 possible topologies of $h$-label binary trees for $h \leq 2$ and we denote the $h$-label binary trees of such topologies by using a superscript $i$, $i \in \{\text{i}, \ldots, \text{x}\}$. For vertex $u$ of the query in Fig. 3, Fig. 7 shows one $T^{\text{vii}}_{u,2}$ as an example. Then, we propose the following proposition for pruning.

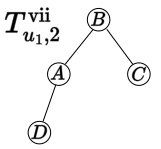
Proposition 3. *Consider a height $h$, a query $Q$, and a ball with the center $w$. If there exists at least one $T^i_{v,h}$, $i \in \{\text{i}, \ldots, \text{x}\}$ where $v$ is a vertex of $Q$ but there does not exist $T^i_{w,h}$ s.t. $T^i_{v,h}$ and $T^i_{w,h}$ are isomorphic, $w$ cannot be matched to $v$ under* sub-iso, hom *and* ssim.

Prop. 3 can be established by a simple proof by contradiction of the definition of the *LGPQ* semantics. The detailed proof is presented in App. A.2 of [51]. Recall that $B$ is spurious if $B$'s center cannot be matched to any vertices of $Q$. Hence, for each vertex $v$ of $Q$ having the same label as $B$'s center's label, we check Prop. 3. We consider $B$ as spurious if $T^i_{v,h}$ *does* exist for all such $v$s and record the existence in a ciphertext $c_{sgx}$ as one of the *PM* of $B$. In the following, we first present an algorithm to enumerate the 2-label binary trees and then present how BF securely computes the $c_{sgx}$.

*4.1.1 Enumeration of 2-label binary trees.* In Fig. 6, topologies i-ii and v (*resp.* vi) show labels of neighbors (*resp.* paths), whereas topology iv is a twiglet. For pruning purposes, we focus on four complex topologies (vii-x), shown within the red dotted rectangle. The enumeration is as follows.

Taking graph $G$ and a vertex $w$ of $G$ as inputs, Alg. 4 enumerates the cases of all subtrees with root $w$ and height 2 used to project $T^i_{w,2}$, $i \in \{\text{vii}, \ldots, \text{x}\}$. First, for each neighbor $u$ of $w$, Lines 1-2 compute a label set $\mathcal{L}(u) = \{L_G(v) \mid v \in \text{neighbors of } u, L_G(v) \neq L_G(u), L_G(v) \neq L_G(w)\}$ for $w$'s neighbors by using a *BFS*. Then, Lines 3-5 enumerate all subtrees with root $w$ by taking all combinations of $w$'s neighbors as $w$'s two child nodes. In detail, for the left child $u$ (*resp.* right child $v$) of $w$, Line 6 computes the number $n_u$ (*resp.* $n_v$) of distinct labels of $u$'s neighbors (*resp.* $v$'s neighbors). As topologies vii-x shown in Fig. 6, if $n_u = 1$ (Line 7), only subtrees of topology vii can be enumerated (Line 8). Otherwise (Line 9), the subtrees of topologies vii-x are enumerated according to the value of $n_v$ (Lines 10-15). Given $w$, $u$, $v$, and topology $i$, $i \in \{\text{vii}, \ldots, \text{x}\}$, Line 16 enumerates $G$'s subtrees of topology $i$ with height 2, which can be used to project $T^i_{w,2}$s.

Example 7. *Take the graph $G$ in Fig. 3 and $G$'s vertex $v_6$ as inputs. The $T^{\text{vii}}_{v_6,2}$ in Fig. 7 can be enumerated by Alg. 4 as follows. First, for $v_6$'s neighbors $v_2$, $v_4$ and $v_5$, Lines 1-2 compute $\mathcal{L}_Q(v_2) =$*




Legend

| Label | Encoding |
|---|---|
| $A$ | 1 |
| $B$ | 2 |
| $C$ | 3 |
| $D$ | 4 |

Canonical encoding: $1 \cdot 1 + 3 \cdot 4^1 + 4 \cdot 4^2 = 77$

Fig. 7. Example of 2-label binary tree of topology vii of $Q$ in Fig. 3

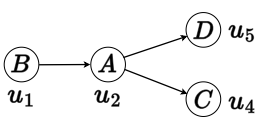



Fig. 8. Example of $h$-twiglet of $Q$ in Fig. 3 (where $h = 3$): $[B, A, [C, D]]$

**Algorithm 4:** Subtree Enumeration Algorithm

**Input :** The graph $G$ and a vertex $w$ of $G$
**Output :** The cases of enumerating subtrees of $G$ for projecting $T^{i}_{w,2}$, $i \in$ {vii, ..., x}

```
1  foreach neighbor u of w that L_G(u) ≠ L_G(w) do
2  |  start a BFS from u to obtain the set L(u);
3  foreach neighbor u of w that L_G(u) ≠ L_G(w) do // u: w's left child
4  |  foreach neighbor v of w that L_G(v) ≠ L_G(w) do // v: w's right child
5  |  |  CaseEnum(u, v, L);
   Procedure CaseEnum(u, v, L):
6  n_u ← |L(u) − {L_G(v)}|, n_v ← |L(v) − {L_G(u)}|;
7  if n_u = 1 then
8  |  TreeEnum(u, v, vii); //enumerate subtrees for T^vii_{w,2}
9  if n_u ≥ 2 then
10 |  foreach i ∈ {vii, viii} do
11 |  |  TreeEnum(u, v, i); //cases for T^vii_{w,2} and T^viii_{w,2}
12 |  if n_v = 1 then
13 |  |  TreeEnum(u, v, ix); //case for T^ix_{w,2}
14 |  if n_v ≥ 2 then
15 |  |  TreeEnum(u, v, x); //case for T^x_{w,2}
   Procedure TreeEnum(u, v, i):
16 enumerate subtrees of topology i by taking u as w's left child and v as w's right child ;
```

Table 1. Numbers of different 2-label binary trees for four distinct topologies of a ball $B$ ($d_{max}$: the maximum degree of vertices of $B$)

| Topology | Numbers of 2-Label binary trees ($\kappa = \min\{|\Sigma_Q|, d_{max}\}$) |
|---|---|
| vii | $A^3_{\kappa-1}$ |
| viii | $A^2_{\kappa-1} \cdot C^2_{\kappa-3}$ |
| ix | $A^3_{\kappa-1} \cdot C^2_{\kappa-4}$ |
| x | $C^2_{\kappa-1} \cdot C^2_{\kappa-3} \cdot C^2_{\kappa-5}$ |

$\{C, D\}$, $\mathcal{L}_Q(v_4) = \{C\}$ *and* $\mathcal{L}_Q(v_5) = \{A\}$. *By setting one neighbor (resp. another distinct neighbor) of* $v_6$ *as the left (resp. right) child* $u$ *(resp.* $v$*) (Lines 3-4), Line 5 enumerates the* $T_{v_6,2}$*s. Given* $v_2$ *(resp.* $v_5$*) as* $u$ *(resp.* $v$*), Line 8 enumerates the* $T^{\text{vii}}_{v_6,2}$ *shown in Fig. 7 since* $n_{v_2} = 1$ *and* $n_{v_5} = 0$.

**Analysis.** In Table 1, we present the maximum number of distinct 2-label binary trees of topologies vii-x in a ball, where C (*resp.* A) denotes the combination (*resp.* permutation) operator and $d_{max}$ is the maximum degree of vertices of this ball. Regarding the time complexity of Alg. 4, Lines 1-2 take $O(V_G + E_G)$ time for *BFS*. Line 5 calls `CaseEnum` for $O({d_{max}}^2)$ times. In `CaseEnum`, Lines 7-15 call `TreeEnum` for $O(1)$ times. For `TreeEnum`, Line 16 takes at most $O({d_{max}}^4)$ time for enumerating subtrees of topology x with height 2. Therefore, Alg. 4's time complexity is $O\big(\max\{{d_{max}}^6, V_G + E_G\}\big)$.

*4.1.2 Computation of* $c_{sgx}$. To compute the $c_{sgx}$ of a ball, we adopt the bloom filter to build a time- and space-efficient index used for securely checking the existence of query's 2-label binary trees by using the SGX. The 2-label binary trees are encoded as follows.

**Canonical encoding of 2-label binary tree.** Assume there is a *canonical encoding* of labels and 2-label binary trees such that *if* two trees are isomorphic, *then* their encodings are identical. Fig. 7 presents an example of converting one $T^{\text{vii}}_{u,2}$ into encoding. For the query $Q$ in Fig. 3, where $|\Sigma_Q| = 4$, assume the encoding of labels $A$, $B$, $C$ and $D$ are 1, 2, 3 and 4, respectively. We propose further *each position in a topology has a unique index*, as shown in Fig. 6's topology x. From the label encoding and the index, we can compute that the canonical encoding[3] of the $T^{\text{vii}}_{u,2}$ in Fig. 7 is $1{\cdot}4^0 + 3{\cdot}4^1 + 4{\cdot}4^2 = 77$. Given a graph $G$, we can enumerate subtrees of topologies vii-x of $G$ with height 2 by Alg. 4, and hence compute the encodings of their projected 2-label binary trees.

[3]For any two label nodes of a $h$-label binary tree $T$ that i) share the same parent, and ii) the unlabeled subtrees of $T$ starting from them are isomorphic, we always put the node with a larger label encoding on the left to ensure a unique encoding of $T$.

**Query-oblivious computation.** BF computes $c_{sgx}$ as follows.
• **On** User**.** Given a query $Q$, User computes $\eta$ encodings of distinct $T^i_{u,2}$s, $i \in \{\text{vii}, \ldots, \text{x}\}$ for each vertex $u$ of $Q$, where $\eta$ is a parameter used to i) ensure the query-oblivious checking, and ii) tune the false positive rates of bloom filters. If there are fewer than $\eta$ encodings for $u$, User takes 0s as the rest encodings. If there are more than $\eta$ encodings, User uses $\eta$ encodings only, which may allow some false positives, so that some spurious balls cannot be detected. Such false positives do not affect the correctness of the pruning. Then, User encrypts these encodings (*e.g.*, by using *AES*) and sends them into SGXs' enclaves on Players by establishing secure channels.
• **On** Player **outside the enclave.** Given a ball $B$ with the center $w$, Player i) computes the encodings of $T^i_{w,2}$, $i \in \{\text{vii}, \ldots, \text{x}\}$, ii) constructs a bloom filter for $B$ by using these encodings with an encoding 0, and iii) transmits the bloom filter into the enclave.
• **On** Player **inside the enclave.** Inside the enclave, $Q$'s encodings are decrypted when received. After $B$'s bloom filter has been transmitted into the enclave, for each vertex $u$ of $Q$ having the same label as that of $w$, BF uses $B$'s bloom filter to test whether $u$'s $\eta$ encodings exist in $B$. The tested results can be aggregated (*e.g.*, by addition) into an integer $r_{sgx}$. $r_{sgx}$ is encrypted as $c_{sgx}$ of $B$.

**Analysis.** The privacy analysis of BF is presented in Sec. 5. Regarding the bloom filter, the number of hash functions that minimizes its false positive rate (denoted by $p$) is $m/n \cdot \ln 2$, where $m$ and $n$ are the numbers of vector's bits and trees, respectively. Since the data transmission into SGX is known to be time-consuming, we focus on choosing the optimal value of parameter $m$. By some simple arithmetics on the number of trees listed in Table 1, we have the following equation,

$$m = -\frac{n \ln p}{(\ln 2)^2} < -4 \cdot \frac{\kappa^6}{2^3} \cdot \frac{\ln p}{(\ln 2)^2} < \frac{|V_Q|^6 \cdot \ln p}{-2 \cdot (\ln 2)^2} \tag{1}$$

With Eqa. 1, we can tune $p$ to balance the data transmission cost and the pruning power of BF.

### 4.2 Query-Oblivious Twiglet Pruning

*Without* using the *TEE*, previous works check under the ciphertext domain the existence of simple topologies, *e.g.*, neighbors [15] and paths [52] of the query. If such topologies exist in the query but do not exist in a ball, this ball is considered as a spurious ball (Sec. 3.1). In this subsection, we propose using *twiglet*s to compute a ciphertext $c_{phe}$ as another one of the *PM* of a ball.

First, we define *h-twiglet* as follows. Given a graph $G$, the *h-twiglet* of $G$ is a topology consisted of labels of $h$+1 vertices of $G$, denoted by $\big[L_G(v_1), \ldots, L_G(v_{h-1}), [L_G(v_h), L_G(v_{h+1})]\big]$, $v_i \in V_G$, which satisfies the following: i) $(v_i, v_{i+1}) \in E_G$ or $(v_{i+1}, v_i) \in E_G$, $1 \le i \le h-2$, ii) $(v_{h-1}, v_h) \in E_G$ and $(v_{h-1}, v_{h+1}) \in E_G$, and iii) $\forall i, j \in [1, h+1]$, $i \neq j \Rightarrow L_G(v_i) \neq L_G(v_j)$. We denote such a topology as a $h$-twiglet $t$ starting from $v_1$. Fig. 8 shows an example of 3-twiglet $\big[A, B, [C, D]\big]$ starting from $u_1$. Then, we have the following proposition.

Proposition 4. *Consider a query $Q$ and a ball $B$ with the center $w$. For any vertex $u$ of $Q$ that having the same label as $w$'s label, if there exists one $h$-twiglet in $Q$ starting from $u$ but there does not exist such $h$-twiglet in $B$ starting from $w$, $w$ cannot be matched to $u$ under* hom, sub-iso, *and* ssim *semantics.*

Prop. 4 can be proved by contradiction of the definition of the *LGPQ* semantics. The detailed proof is presented in App. A.2 of [51]. For each vertex $u$ of $Q$ having the same label as the ball center $w$'s label, we check the matching from $w$ to $u$ by using Prop. 4. If $w$ cannot be matched to any vertices of $Q$, this ball is spurious. We illustrate the query-oblivious step for computing the $c_{phe}$ by the twiglet pruning algorithm (Alg. 5).
• **On** User**.** Given a length $h$, for each vertex $u$ of $Q$, User enumerates all the possible $h$-twiglets starting from $u$ consisted of $|\Sigma_Q|$ labels and record them in a $h$-twiglet table of $u$. Take $h = 3$ and

Table 2. The 3-twiglet table $\mathcal{T}(u_1)$ of $u_1$ of query $Q$ in Fig. 3, where $\Sigma = \{A, B, C, D\}$ and $L(u_1) = B$

| 3-twiglet $t$s in $\mathcal{T}(u_1)$ | ciphertext $c_t$s | plaintext | meaning |
|---|---|---|---|
| $[B, A, C]$ | $g^x rq$ | 0 | exists |
| $[B, A, D]$ | $g^x rq$ | 0 | exists |
| $\boxed{B, A, [C, D]}$ | $g^x rq$ | 0 | exists |
| $[B, C, A]$ | $g^x r$ | 1 | not exists |
| $B, C, D]$ | $g^x r$ | 1 | not exists |
| $\boxed{B, C, [A, D]}$ | $g^x r$ | 1 | not exists |
| $[B, D, A]$ | $g^x r$ | 1 | not exists |
| $[B, D, C]$ | $g^x r$ | 1 | not exists |
| $\boxed{B, D, [A, C]}$ | $g^x r$ | 1 | not exists |

**Algorithm 5:** Twiglet Pruning Algorithm `TwigletPrune`

**Input :** The length $h$, the encrypted $h$-twiglet table $\mathcal{T}$, and a ball $B$ with the center $w$
**Output:** The ciphertext $c_{phe}$ of $B$

```
1  Procedure TwigletPrune( T, M_B):
2  r ← 0;
3  start a DFS from w to find all h-twiglets in B and record them in a set R;
4  foreach u in Q that L_Q(u) = L_B(w) do
5      r' ← 1;
6      foreach h-twiglet t in T(u) do
7          if t ∈ R then
8              r' ← r' · c_1;                          // aggregate ciphertext of 1
9          else
10             r' ← r' · c_t;                          // if violation, u has t but w doesn't
11     r ← r + r';
12 return r;
```

vertex $u_1$ in Fig. 8 as an example. Table 2 is a 3-twiglet table $\mathcal{T}(u_1)$ of $u_1$ where the first column of $\mathcal{T}(u_1)$ records *all* possible 3-twiglets starting from $u_1$. If a $h$-twiglet $t$ in $\mathcal{T}(u)$ exists in $Q$, $t$ is encoded and encrypted with 0 and $g^x rq$, respectively, where $g$, $r$ and $q$ are the generator of cyclic group, a random value and the predefined prime used in CGBE, respectively. The absence of $t$ in $Q$ is encoded and encrypted in $\mathcal{T}(u)$ as 1 and $g^x r$, respectively. User sends to Players the first two columns of all $Q$'s vertices' 3-twiglet tables together with $Enc(Q)$ in ② of Fig. 4.

• **On** Player**s.** After receiving the $h$-twiglet tables $\mathcal{T}$s, Player runs `TwigletPrune` (Alg. 5) to compute the $c_{phe}$ for each ball. Taking $h$, $\mathcal{T}$s, and a ball $B$ with center $w$ as inputs, Alg. 5 outputs a ciphertext $r$ as the $c_{phe}$ of $B$. Line 3 first starts a *DFS* from $w$ to find all the $h$-twiglets starting from $w$ and record them in a set $R$. For each vertex $u$ of $Q$ having the same label as $w$'s label (Line 4), Lines 5-11 aggregate into $r$ the ciphertext $r'$ for matching $u$ to $w$. In detail, for each $h$-twiglet $t$ in $\mathcal{T}(u)$ (Line 6), if there also exists $t$ starting from center $w$ in $B$ (Line 7), Line 8 multiplies $r'$ with a chosen ciphertext of 1 (denoted as $c_1$), which ensures the consistency of the power of the private key of CGBE for each $r'$ in Line 11. This is to ensure the correctness of CGBE's decryption on User. If $t$ does not exist in $B$ (Line 9), whether $u$ matches $w$ depends on the existence of $t$ in $Q$, and hence, $r'$ is multiplied by the ciphertext $c_t$ of $t$ in $\mathcal{T}(u)$ (Line 10). Line 11 aggregates all the $r'$s into a ciphertext $r$, which indicates the existence of vertices of $Q$ that may match $w$. If $r$ is a multiple of $q$ after decryption, no vertices of $Q$ can match $w$ that $B$ is spurious. Line 12 returns $r$ as the $c_{phe}$ of $B$.

Example 8. *Consider the query $Q$ and graph $G$ in Fig. 3. Taking length* 3*, the encrypted* 3*-twiglet table $\mathcal{T}$s of $Q$ and the ball $B' = G[v_6, 3]$ of $G$ as inputs, Alg. 5 computes $c_{phe}$ of $B'$ as follows. Since $L_Q(u_1) = L_G(v_6) = B$ (Line 4), Lines 5-11 check whether $u_1$ matches $v_6$. Specifically, consider the first twiglet $[B, A, C]$ in $\mathcal{T}(u_1)$ as shown in Table 2. Since $[B, A, C]$ exists in $B'$, Line 8 aggregates $c_1$ into $r'$. For the last twiglet $\boxed{B, D, [A, C]}$ in $\mathcal{T}(u_1)$, $\boxed{B, D, [A, C]}$ does not exist in $G[v_6, 3]$ and hence Line 10 aggregates $g^x r$ in $r'$. Finally, a ciphertext $r'$, whose decrypted value has no factor $q$, is aggregated into $r$ by addition. Since the decrypted $c_{phe}$ of $B'$ (i.e., the decrypted $r$) has no factor $q$, $B'$ is not spurious.*

**Analysis.** In Sec. 5, we present Alg. 5 meets the privacy targets. Regarding the value of $h$, $h$=3 is

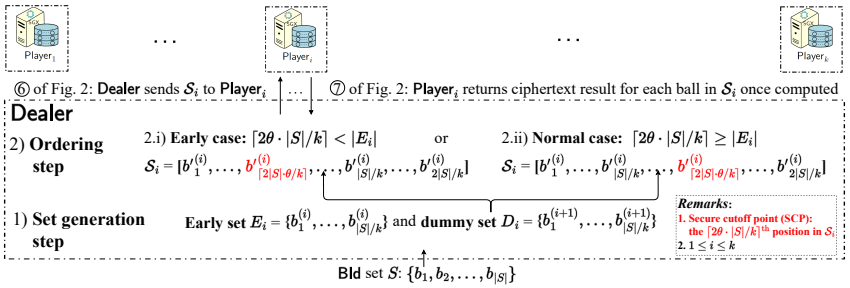



Fig. 9. Steps of SSG: 1) The set generation step (bottom) uses the BId set $S$ to generate $S_i$ ($1 \leq i \leq k$), which consists of the early set $E_i$ and the dummy set $D_i$. 2) The ordering step generates a sequence $\mathcal{S}_i$ for Player$_i$ by using $E_i$ and $D_i$, and orders the BIds in $\mathcal{S}_i$. There are two cases of the ratio $\theta$ (middle) of the positives to all balls in $S$: 2.i) early case ($\theta < 1/2$), and 2.ii) normal case ($\theta \geq 1/2$).

used to covered label information of paths contained by topologies i-vi in Fig. 6. We assume $3 \leq h \leq 5$ for efficiency. For the complexity, Line 3 takes $O(|V_B| + |E_B|)$ time for *DFS* on ball $B$ and hence, $O((|V_B| + |E_B|) \cdot {d_B}^2)$ time to enumerate all $h$-twiglets, where $d_B$ is the maximum vertex degree of $B$. The ciphertext aggregation (Lines 4-11) takes $O(|V_Q| \cdot \mathrm{A}_{h-2}^{|\Sigma_Q|-1} \cdot \mathrm{C}_2^{|\Sigma_Q|-h+1})$ time in the worst case, where C (*resp.* A) denotes the combination (*resp.* permutation) operator.

### 4.3 Secure Retrieval of Balls

Prilo* introduces a secure retrieval scheme into Prilo. The major steps of the scheme can be summarized as follows: 1) Players first compute the *PM*s of balls by using the pruning techniques in Secs. 4.1-4.2; 2) Dealer then generates sequences for Players to evaluate balls in their sequences, using techniques in Secs. 3.1-3.2, to obtain ciphertext results; and 3) From Dealer, User receives ciphertext results and retrieve the encrypted data of balls that contain matching subgraphs. We elaborate on the scheme in relation to ③-⑨ of Fig. 4 below.

**On** User**.** After ③ receiving the encrypted *PM*s (Secs. 4.1-4.2) from Players, User decrypts them. Given a $PM = (c_{sgx}, c_{phe})$ of a ball $B$, if the plaintext of either $c_{sgx}$ or $c_{phe}$ indicates that $B$ is spurious, $B$ is denoted as *negative*. Otherwise, $B$ is denoted as *positive*. The information of whether $B$ is negative ④ is sent from User to Dealer as $B$'s decrypted $PM$. User waits for Dealer to ⑧ send the ciphertext results, and then decrypts them for ⑨ retrieving the encrypted data of balls that contains matching subgraphs.

**On** Dealer**.** After ④ receiving the set $S$ of the ball identifiers (BIds) with their decrypted *PM*s, Dealer ⑤ generates for Player$_i$ ($1 \leq i \leq k$, where $k$ is the number of Players) a BId sequence $\mathcal{S}_i$ by using the *PM*s. $\mathcal{S}_i$ consisted of *a part of* BIds in $S$. The BIds of positives are *put in the front part of* $\mathcal{S}_i$ *in a query-oblivious manner*. After $\mathcal{S}_i$ ⑥ is sent to Player$_i$, Player$_i$ conducts candidate enumeration and query verification (Secs. 3.1-3.2) for balls in $\mathcal{S}_i$. For each ball $B$ evaluated on Player$_i$, Player$_i$ ⑦ sends $B$'s ciphertext result ($r_i$ in Line 7 of Alg. 3) to Dealer *as soon as the evaluation on B finishes.* Dealer ⑧ sends the ciphertext results to User for decryption.

The scheme above enables User to i) find the balls containing matches among positives early, ii) ⑨ retrieve the encrypted data of such balls from Dealer early, and iii) start computing the matching subgraphs early, while each Player$_i$ may still be evaluating the rest of the balls in $\mathcal{S}_i$.

To ensure the privacy preservation of the retrieval scheme, *the key is to generate secure sequences of* BIds, so that each Player is *unaware* of the time when *all* its positives have been evaluated.

**Secure sequence generation (**SSG**).** SSG has the set generation step and the ordering step, as illustrated with Fig. 9.

**1) Set generation step.** Given a BId set $S$ and $k$ Players, SSG i) generates a BId set $S_i$ for $\text{Player}_i$ ($1 \le i \le k$) that consists of two subsets, namely *early set* $E_i$ and *dummy set* $D_i$, and then ii) orders the BIds in $E_i$ and $D_i$ to obtain sequences $\mathcal{E}_i$ and $\mathcal{D}_i$, and hence the BId sequence $\mathcal{S}_i = \mathcal{E}_i || \mathcal{D}_i$, where || is concatenation. The detailed generation of these two subsets can be described as follows.

- **Early set ($E$).** Assume there are $|S|\cdot\theta$ ($0 \le \theta \le 1$) BIds of positives in $S$. Note that $\theta$ can be derived by the decrypted *PM*s and is *unknown* to Players. SSG partitions $S$ into $k$ early sets ($E_i$, $1 \le i \le k$) of the same size by assigning random $|S|\cdot\theta/k$ BIds of positives to $E_i$.
- **Dummy set ($D$).** Given the early set $E_i$ ($1 \le i \le k$), for each $\text{Player}_i$, SSG generates the dummy set $D_i$ by assigning random $|S|/k$ BIds in $S - E_i$ to $D_i$, s.t. i) $\forall i \in [1,k]$, $E_i \cap D_i = \emptyset$, ii) $\forall i,j \in [1,k]$ ($i \ne j$), $D_i \cap D_j = \emptyset$, and iii) $D_1 \cup \ldots \cup D_k = S$. Note that SSG can simply generate $D_i = E_{(i+1) \bmod k}$, $1 \le i \le k$ when $k \ge 2$.

Next, we present how SSG orders the BIds in $E_i$ and $D_i$ to obtain $\mathcal{S}_i$.

**2) Ordering step.** The length of sequence $\mathcal{S}_i$ ($1 \le i \le k$) to be generated by SSG for $\text{Player}_i$ is $|\mathcal{S}_i| = |E_i| + |D_i| = 2\cdot|S|/k$. We denote the $\lceil 2\theta\cdot|S|/k \rceil^{\text{th}}$ position in $\mathcal{S}_i$ as the *secure cutoff point* (SCP) and use SCP to help ordering BIds, such that all positives of $S_i$ would have been evaluated by $\text{Player}_i$ when $\text{Player}_i$ finishes the evaluation of the ball located on SCP. The ordering has two cases.

- **i) Early case ($\theta < 1/2$).** As Fig. 9 shows, SCP resides in the front half part of $\mathcal{S}_i$ since $\lceil 2\theta\cdot|S|/k \rceil < |S|/k \rceil = |E_i|$. W.l.o.g, assume that $y = \lceil 2\theta\cdot|S|/k \rceil$ and $y$ is even. For $\text{Player}_i$ ($1 \le i \le k$), SSG i) randomly chooses $y/2$ BIds of negatives in $E_i$, ii) inserts into a set $E_i'$ the chosen $y/2$ BIds together with the BIds of all the $y/2$ positives in $E_i$, and iii) orders the BIds in $E_i'$ $\left(\textit{resp.}\ (E_i - E_i') \cup D_i\right)$ with a random sequence to obtain the sequence $\mathcal{E}_i$ (*resp.* $\mathcal{D}_i$). Then, $\mathcal{S}_i = \mathcal{E}_i || \mathcal{D}_i$ .
- **ii) Normal case ($\theta \ge 1/2$).** SCP resides in the rear half part of $\mathcal{S}_i$ since $\lceil 2\theta\cdot|S|/k \rceil \ge |S|/k \rceil = |E_i|$. In this case, SCP cannot lead to an early return of positives' ciphertext results to User. Hence, SSG simply applies a random sequence generation (denoted by RSG), which i) randomly partitions $S$ into $k$ subsets of the same size (for simplicity, assume $|S|$ is divisible by k), and ii) randomly orders the BIds in the subsets to obtain sequences for Players. Note that the value of $\theta$ depends on the pruning power of our proposed pruning techniques in Secs. 4.1-4.2.

Example 9. *Consider* $k = 3$ Player*s and a* BId *set* $S = \{b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9\}$ *for an example of* SSG. *Assume* $b_5$, $b_6$ *and* $b_7$ *are* BId*s of positives. Hence,* $\theta = 3/9$ *and* SCP *is the* $2^{nd}$ *position.* SSG *generates subsets* $E_1 = \{b_8, b_2, b_5\} = D_2$, $E_2 = \{b_6, b_1, b_9\} = D_3$ *and* $E_3 = \{b_7, b_3, b_4\} = D_1$. *Then,* SSG *generates* BId *sequences* $\mathcal{S}_1 = [b_5, b_8] \,||\, [b_9, b_2, b_1, b_6]$, $\mathcal{S}_2 = [b_6, b_1] \,||\, [b_4, b_7, b_9, b_3]$ *and* $\mathcal{S}_3 = [b_3, b_7] \,||\, [b_2, b_5, b_8, b_4]$. *Dealer can receive the ciphertext results of all positives when* $b_8$ *in* $\mathcal{S}_1$, $b_1$ *in* $\mathcal{S}_2$, *and* $b_7$ *in* $\mathcal{S}_3$ *have been evaluated by* $\text{Player}_1$, $\text{Player}_2$, *and* $\text{Player}_3$, *respectively.*

We remark that $\theta$ and SCP are not known to Players and Players have no way to identify the case and therefore the positives. The privacy analysis of SSG is presented in Sec. 5. With the sequences generated by SSG, User can receive the ciphertext results of all positives before the end of the whole query processing of Prilo on Players. After decrypting the ciphertexts, User identifies the balls containing matching subgraphs and sends to Dealer the BIds of these balls for retrieving their encrypted ball data to decrypt and to compute the matching subgraphs.

## 5 PRIVACY ANALYSIS

Due to space restrictions, we present the main ideas of the privacy analysis in this section, and provide the detailed proofs in App. B of [51]. We recall that the privacy target (Sec. 2.3) consists of i) the query privacy, and ii) the access pattern privacy. We start with Prop. 5.

Proposition 5. *Given a query $Q$, i) the encrypted encodings $M^E_{Q_e}$ of $Q$'s adjacency matrix, ii) the twiglet tables $\mathcal{T}$s, and iii) the encrypted encodings of 2-label binary trees of $Q$'s vertices are preserved from* SP *against the attack model.*

As $M^E_{Q_e}$ and $\mathcal{T}$ (*resp.* the encrypted encodings of $Q$'s 2-label binary trees) are encrypted by CGBE (*resp. AES*), they are protected from Players. These encrypted messages are protected from Dealer since Dealer i) does not have the private keys of CGBE and *AES*, and ii) cannot obtain these messages due to the assumption that Dealer and Players do not collude (Sec. 2.3). Prop. 5 holds.

For the Prilo framework, we have i) Algs. 1-3 are *query-oblivious* (as shown in the analyses in Sec. 3) that preserves the access pattern privacy, and ii) the encryption used in Algs. 1-3 preserves the query privacy [15]. Hence, we can derive Prop. 6 as follows.

Proposition 6. Prilo *preserves the privacy target from* SP *against the attack model.*

Regarding the BF pruning, the bloom filter always tests $O(\eta)$ encodings ($\eta$ is a parameter set by User) inside SGX's enclave (Sec. 4.1.2). BF is query-oblivious since $\eta$ is independent of the edge set of the query. The enclave preserves the privacy of the plaintexts of these encodings. The size of each bloom filter is smaller than $4KB$ under our experimental settings (Sec. 6.1). Hence, the granularity of the memory access pattern attacks on the enclave becomes finer that "*the attackers become harder to get valid information*" [22]. Putting these ideas together, we have Prop. 7.

Proposition 7. *The* BF *pruning preserves the privacy target from* SP *against the attack model.*

Regarding the twiglet pruning (in Sec. 4.2), let $\mathcal{G}(\mathcal{T}, i)$ $\big($*resp.* $\mathcal{G}(r)\big)$ be a function that returns 1 if SP can compute the corresponding plaintext of ciphertext $c_{t_i}$ of $h$-twiglet $t_i$ in $\mathcal{T}$ (*resp.* the output ciphertext $r$ of Alg. 5) and returns 0, otherwise. Then, we quantify the probability that SP can attack $r$ $\big($*i.e.*, $\mathcal{G}(r) = 1\big)$ after applying twiglet pruning, as presented in Prop. 8.

Proposition 8. *After running* `TwigletPrune`, $Pr[\mathcal{G}(r) = 1] \leq 1/2^n + \epsilon$, *where $n$ is the number of ciphertexts $c_t$s aggregated into $r$ in Line 10 of Alg. 5 and $\epsilon$ is a negligible value.*

$Pr[\mathcal{G}(r) = 1]$ equals the product of $Pr[\mathcal{G}(\mathcal{T}, i) = 1]$ for the $n$ $t_i$s used in `TwigletPrune`. As $t_i$s are encrypted by CGBE (secure against *CPA* [15]), we have $Pr[\mathcal{G}(\mathcal{T}, i) = 1] \leq 1/2 + \epsilon'$ ($\epsilon'$ is a negligible value) which has a negligible difference from random guessing. Prop. 8 holds. Moreover, Lines 6-11 of Alg. 5 are independent of the edge set of the query that `TwigletPrune` is query-oblivious, we have established Prop. 9.

Proposition 9. `TwigletPrune` *preserves the privacy target from* SP *against the attack model.*

For SSG (Sec. 4.3), Dealer knows i) the decrypted pruning messages (*PM*s) and ball identifiers (BIds) of the balls sent from User, and ii) the ciphertext results sent from Players. However, Dealer has only the encrypted ball data without i) the secret key for ball data's decryption, and ii) the private key of CGBE for decrypting ciphertext results. Dealer cannot infer the query structure by using such information. For Players, each $\text{Player}_i$ knows only the BId sequence $\mathcal{S}_i$ without the plaintexts of the *PM*s. The probability that $\text{Player}_i$ can determine whether a ball in $\mathcal{S}_i$ is spurious is smaller than $1/2 + \epsilon$, where $\epsilon$ is a negligible value (see App. B.4 of [51] for details). Hence, the access pattern privacy is preserved by SSG from Players. The query privacy is also preserved by SSG from Players since each $\text{Player}_i$ knows only the ball data and $\mathcal{S}_i$ without the plaintexts of the *PM*s when evaluating the balls in $\mathcal{S}_i$. With the assumption that Dealer and Player do not collude, we have Prop. 10.

Proposition 10. *The privacy target is preserved by* SSG *from* SP *against the attack model.*

By putting Props. 5, 6, 7, 9 and 10 together, we have Theorem. 1.

Theorem 1. Prilo$^*$ *preserves the privacy target from* SP *against the attack model.*

Table 3. Statistics of three real-world datasets

| Graph $G$ | $\|V_G\|$ | $\|E_G\|$ | $\|\Sigma_G^H\|$ | $\|\Sigma_G^S\|$ |
|---|---|---|---|---|
| *Slashdot* | 82,168 | 948,464 | 100 | 64 |
| *DBLP* | 317,080 | 1,049,866 | 150 | 64 |
| *Twitter* | 81,306 | 1,768,149 | 100 | 64 |

Table 4. Statistics of candidate balls $B$s for 10 random queries under the default setting

| Graph | Avg. no. of balls per query | avg. $\|V_B\|$ | stddev. of $\|V_B\|$ | avg. $\|E_B\|$ | stddev. of $\|E_B\|$ | Max. degree |
|---|---|---|---|---|---|---|
| $Slashdot_{100}$ | 204 | 243 | 218 | 1085 | 1062 | 333 |
| $Slashdot_{64}$ | 3383 | 580 | 538 | 3324 | 3325 | 689 |
| $DBLP_{150}$ | 18 | 25 | 11 | 34 | 25 | 20 |
| $DBLP_{64}$ | 3001 | 45 | 38 | 66 | 64 | 38 |
| $Twitter_{100}$ | 378 | 245 | 245 | 822 | 854 | 214 |
| $Twitter_{64}$ | 5734 | 467 | 495 | 2113 | 2344 | 398 |

## 6 EXPERIMENTAL EVALUATION

In this section, we evaluate the efficiency of Prilo* and the effectiveness of Prilo*'s optimizations.

### 6.1 Experimental Settings

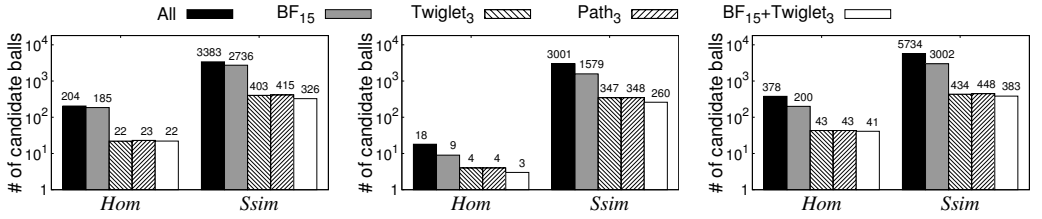
**Platform**. We implemented the prototype of Prilo* in C++ using a machine with an Intel Core i7-7567U 3.5GHz CPU and 32GB RAM running Ubuntu 20.04.4 LTS with Intel(R) SGX SDK[4] to test the performance for both User and SP (including multiple Players and a Dealer). CGBE was implemented by using the *GMP* libraries.

**Datasets and query sets**. We used three real-world datasets, namely *Slashdot*, *DBLP*, and *Twitter* [31], which are also used in [16, 48, 52, 54]. The vertices of these datasets do not have labels. Similar to existing works [16, 37, 52], we generated a random label for each vertex to evaluate Prilo*'s performance. We focused on *hom* and *ssim* queries, but omitted *sub-iso* queries, as the performance is similar to that of *hom* queries. Table 3 shows the statistics of these datasets, where $|\Sigma_G^H|$ (*resp.* $|\Sigma_G^S|$) is the size of label set for *hom* (*resp. ssim*) queries, whose value was set according to [16, 52]. Regarding the query sets, we used the same query generator *QGen* [52]. We generated 10 random queries for each experiment. Taking a query size $|V_Q|$, a diameter $d_Q$ and a graph $G$ as inputs, *QGen* returned random subgraphs of $G$ as output queries. The default values of $|V_Q|$ and $d_Q$ were 8 and 3, respectively. Table 4 shows the statistics of balls evaluated on Players under the default setting. For each ball $B$, we used the size of $B$'s vertex set, $|V_B|$, as the *ball size* of $B$.

**Default parameters**. These parameters are described as follows:

- CGBE. The encoding $q$ and random number $r$ for CGBE were both of 32 bits and the public value was of 4096 bits [52].
- *Query*. The default values of $|V_Q|$ and query diameter $d_Q$ were 8 and 3, respectively. We set $d_Q = 4$ to investigate the pruning power of $h$-twiglet by varying $h$ from 3 to 5.
- BF *pruning* (BF). For the parameter $\eta$ used to ensure BF's obliviousness, we set $\eta = 256$. In practice, the number $n$ of distinct 2-label binary trees starting from a ball's center is much smaller than $|V_Q|^6/2$, in particular, $n \leq 10K$ for almost all our experiments. Hence, we set $n = 10K$ and the desired false positive rate $p = 0.3$, and $m = 25K$ bits are required for the bloom filter by Eqa. 1. Compared with passing $25K$ bits of data between the enclave and the application of SGX, BF's online construction of bloom filters for ball centers may take more time, especially for the enumeration of subtrees of topology x (Lines 14-15 of Alg. 4). Hence, we used a threshold $t$ for BF to balance the efficiency and pruning performance. Specifically, for the ball center, if there exist more than $t$ neighbors whose $\mathcal{L}$ (Line 2 of Alg. 4) has size larger than 3, BF simply marked the ball as positive. We varied $t = 5$, 15, or 25 for BF and denoted the algorithm as $\mathsf{BF}_t$. The default value of $t$ was 15.

[4]https://github.com/intel/linux-sgx#license




(a) *Slashdot* (b) *DBLP* (c) *Twitter*

Fig. 10. Average number of candidate balls for algorithms

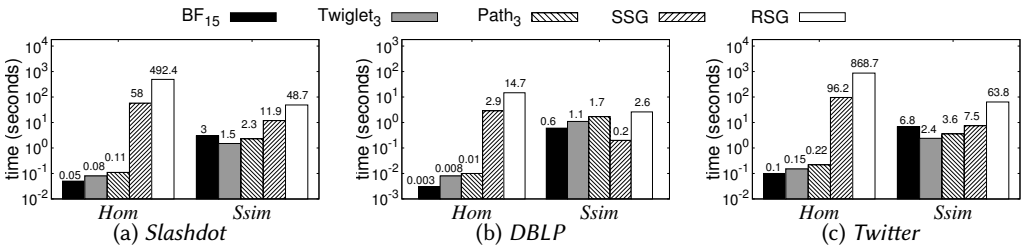
• *Twiglet pruning* (Twiglet). We pruned balls using $i$-twiglets, $3 \leq i \leq h$, where the hop length $h$ ranged from 3 to 5. We use a $h$ value to denote Twiglet as $\texttt{Twiglet}_h$. The default value of $h$ was 3.
• *Path-based pruning* (Path) [52]. We used the existing path-based pruning technique as the baseline for comparison. We denote Path using a $h$ value as $\texttt{Path}_h$.
• *Number of* Players. The number of Player servers is denoted by $k$, where $k = 4$, 8, or 16. The default value of $k$ was 4.

## 6.2 Overall Performance

**EXP-1. Performance on the** User **side**. User i) generates the encrypted messages for queries, ii) decrypts the pruning messages, and iii) decrypts the ciphertext results sent from Dealer.
• *Preprocessing*. Given a query $Q$, User generated the encrypted encoding $M_{Q_e}^E$ of $Q$'s adjacency matrix, a twiglet table $\mathcal{T}$ and the encrypted encodings of 2-label binary trees for each vertex of $Q$. The total preprocessing time was always less than 0.25$s$, including i) *AES256*'s encryption for the encodings of 2-label binary trees to be sent to the enclave, and ii) CGBE's encryption for the $M_{Q_e}^E$, the $\mathcal{T}$s, and the value 1 to obtain a chosen ciphertext $c_1$ used for Twiglet (Line 8 of Alg. 5).
• *Decryption*. User decrypted the ciphertexts returned by BF (Sec. 4.1.2), Twiglet (Alg. 5) and Prilo (Alg. 3). The total decryption time under our experiments was always less than 0.5$s$ only.
• *Message sizes*. Given query $Q$ and $h$ used for Twiglet, User sent to Players i) $\eta \cdot |V_Q|$ encodings of 2-label binary trees encrypted by *AES256*, and ii) the $M_{Q_e}^E$ and encrypted $\mathcal{T}$s which contained $|V_Q|^2$ and $|V_Q| \cdot P_{h-1}^{|\Sigma_Q|-1} \cdot C_2^{|\Sigma_Q|-h}$ ciphertexts encrypted by CGBE, respectively. For $\text{Player}_i$, where $1 \leq i \leq k$ on the SP side, Algs. 4 and 5 returned $O(\mathcal{N}_i)$ ciphertexts to be sent to User, where $\mathcal{N}_i$ is the number of balls evaluated on $\text{Player}_i$. Take the queries used for *Twitter* in **EXP-2** as an example, the size of $M_{Q_e}^E$ and encrypted $\mathcal{T}$s under our experimental settings were at most $k \times 8MB$, where $k$ is the number of Players. The size of encodings encrypted by *AES256* is smaller than 10$MB$. The total size of messages sent from Players to User was no larger than 20$MB$ only.

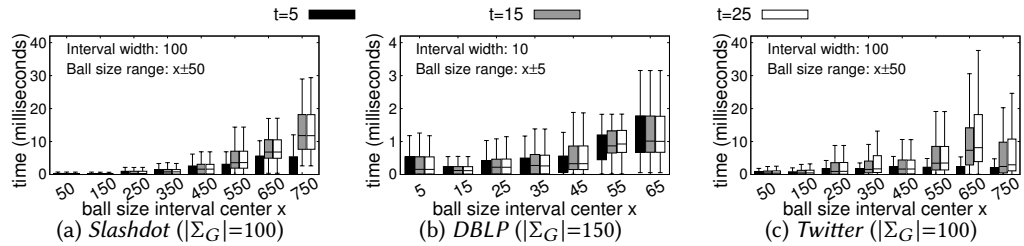
**EXP-2. Overall runtimes of** $\texttt{BF}_{15}$ **and** $\texttt{Twiglet}_3$. We investigate the efficiency of BF and Twiglet under the default setting. Due to space restrictions, we report only the results when the figures and detailed analysis are presented in App. C of [51]. $\texttt{BF}_{15}$ took hundreds of milliseconds for *hom* queries and few seconds for *ssim* queries, respectively. The runtimes of $\texttt{BF}_{15}$ on all datasets increase as the ball sizes increase. The runtimes of $\texttt{Twiglet}_3$ on *Slashdot* and *Twitter* increase slightly as the ball sizes increase when $\texttt{Twiglet}_3$'s runtime on *DBLP* is not sensitive to the ball size.

**EXP-3. Overall performance of** Prilo*. The following average results were from 10 random queries under the default setting. In Fig. 10, *All* denotes the average number of candidate balls, which can be either positive or negative. Fig. 10 reports the average number of candidate balls after pruning negatives by each method. Although $\texttt{BF}_{15}$ pruned fewer negatives than $\texttt{Twiglet}_3$ or $\texttt{Path}_3$, $\texttt{BF}_{15}$ helped $\texttt{Twiglet}_3$ to further prune negatives, especially for *ssim* queries.




(a) *Slashdot* (b) *DBLP* (c) *Twitter*

Fig. 11. Average runtimes of various algorithms in Prilo*




(a) *Slashdot* ($|\Sigma_G|$=100) (b) *DBLP* ($|\Sigma_G|$=150) (c) *Twitter* ($|\Sigma_G|$=100)

Fig. 12. Per-ball runtimes by varying $t$ for $\mathsf{BF}_t$

In Fig. 11, we denote Steps ⑤-⑦ of Fig. 4 of Prilo* by using SSG (*resp.* a baseline that applies RSG for both cases of SSG) as SSG (*resp.* RSG), and denote the times for Dealer to obtain the ciphertext results of positives as their runtimes. Fig. 11 shows the average total runtimes of (i) $\mathsf{BF}_{15}$, $\mathtt{Twiglet}_3$, and $\mathtt{Path}_3$, and (ii) SSG and RSG for both the *hom* and *ssim* queries.[5] First, it can be observed that the runtimes of $\mathsf{BF}_{15}$, $\mathtt{Twiglet}_3$, and $\mathtt{Path}_3$ are very small. We can see that the runtimes of the pruning methods ($\mathsf{BF}_{15}$, $\mathtt{Twiglet}_3$, and $\mathtt{Path}_3$) for *hom* queries are much smaller than the ones for *ssim* queries due to fewer candidate balls. Next, by comparing the runtimes of SSG and RSG, we can better present the performance of the optimization in Sec. 4.3. The runtime of SSG is often one order of magnitude smaller than the runtime of RSG. Regarding the runtime for User to start receiving from Dealer the ciphertext results of non-spurious balls,[6] the runtime of Prilo* equals the sum of the runtimes of BF, `Twiglet` and SSG while the runtime of Prilo equals that of RSG. For example, the average runtimes of Prilo* and Prilo on *Twitter* are 6.8 + 2.4 + 7.5 = 16.7 and 63.8, respectively. The runtimes of Prilo* was either much smaller than or similar to that of Prilo, which verified the effectiveness of Prilo*.

### 6.3 Effects of Varying Settings

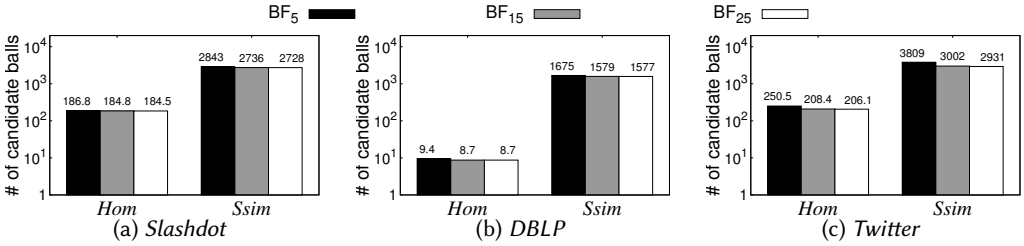
To investigate the performance of the algorithms, we ran them by varying one parameter at a time while keeping the other parameters to their default values. It is known that the numbers of balls to be evaluated vary with queries. Hence, for ease of presentation, we used boxplots to present the runtimes.[7] The following figures do not display the outliers (fewer than 1%).

**EXP-1. Varying $t$ for $\mathsf{BF}_t$.** Fig. 12 shows that $\mathsf{BF}_t$'s runtimes increase as $t$ increases until $t$'s value reaches 15. The reason is that most ball centers of these datasets have fewer than 15 neighbors $v$s of the ball center that $|\mathcal{L}(v)| \geq 3$ and hence decreasing $t$'s value can hardly reduce the runtime of

[5]To save the runtime, the balls that obviously involve numerous candidate enumeration simply bypass the pruning.

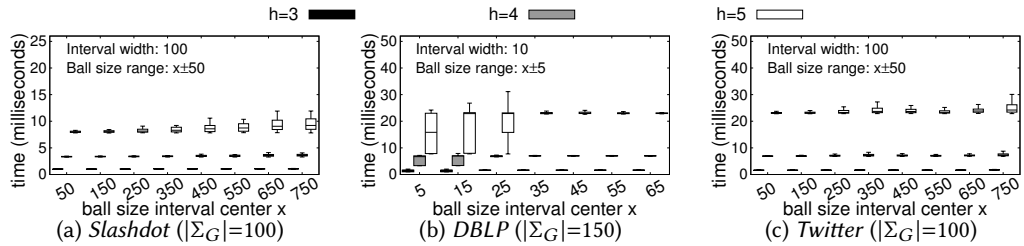
[6]We omitted the evaluation on User to compute the detailed subgraphs by using the state-of-the-art *LGPQ* matching algorithms [23, 24, 37] under the plaintext domain.

[7]In $x$-axis, we grouped the balls according to their sizes. Only 1% of balls were beyond the $x$ range. The box of each interval was drawn around the region between the first and third quartiles, and a horizontal line at the median value. The whiskers extended from the ends of the box to the most distant point with a runtime within 1.5 times the interquartile range. Points that lie outside the whiskers were outliers.



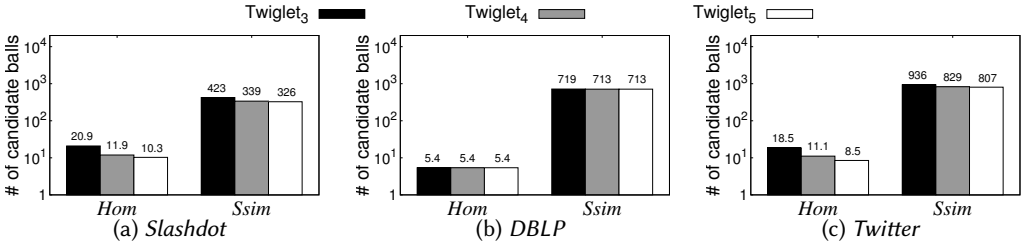

(a) *Slashdot* (b) *DBLP* (c) *Twitter*

Fig. 13. Average number of candidate balls by varying $t$ for $\mathsf{BF}_t$



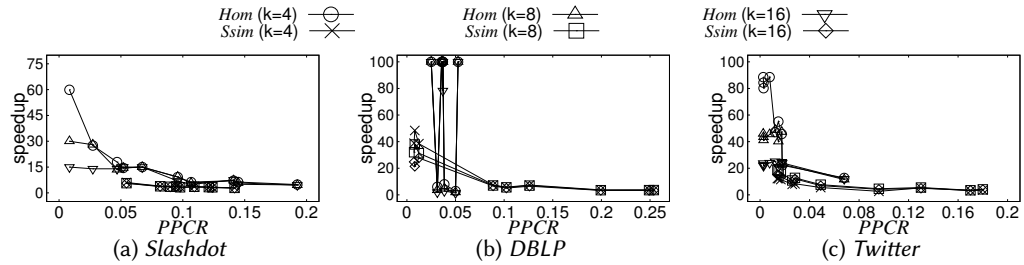

(a) *Slashdot* ($|\Sigma_G|$=100) (b) *DBLP* ($|\Sigma_G|$=150) (c) *Twitter* ($|\Sigma_G|$=100)

Fig. 14. Per-ball runtimes by varying $h$ for $\mathtt{Twiglet}_h$




(a) *Slashdot* (b) *DBLP* (c) *Twitter*

Fig. 15. Average number of candidate balls by varying $h$ for $\mathtt{Twiglet}_h$




(a) *Slashdot* (b) *DBLP* (c) *Twitter*

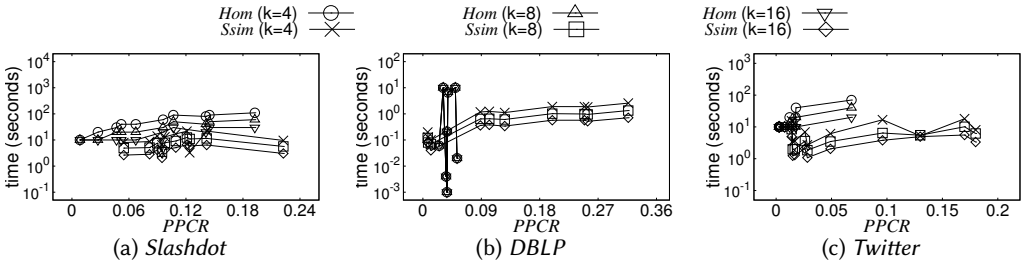
Fig. 16. Prilo*'s speedup by varying *PPCR*

$\mathsf{BF}_t$ when $t \geq 15$. From Fig. 13, we see that $\mathsf{BF}_t$ with a larger $t$ prunes more negatives and $t = 15$ reaches a balance between the efficiency and pruning power of $\mathsf{BF}_t$.

**EXP-2. Varying $h$ for $\mathtt{Twiglet}_h$.** As Fig. 14 shows, the runtimes of $\mathtt{Twiglet}_h$ increase as $h$ increases since a larger $h$ requires more time for *DFS* in $\mathtt{Twiglet}_h$ to obtain the $i$-twiglets, $3 \leq i \leq h$. In Fig. 14(b), $\mathtt{Twiglet}_h$'s runtimes on *DBLP* vary a lot for balls of small sizes. This is because the number of twiglets enumerated for small balls varies a lot.

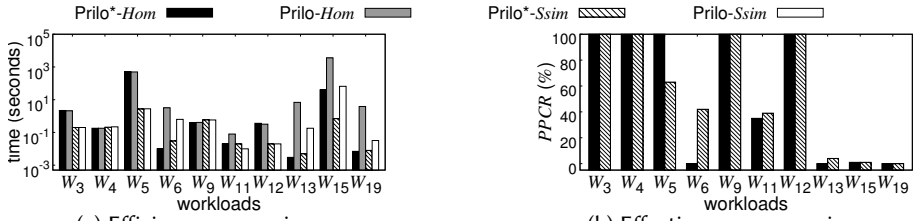
Fig. 15 shows that $\mathtt{Twiglet}_h$ with a larger $h$ can prune more negatives by using more twiglets. However, the improvement in pruning is not obvious since there are few $i$-twiglets where $i > 3$. In practice, $h = 3$ demonstrates a good balance between the efficiency and the pruning power.

**EXP-3. Varying $k$ for** Players. In Fig. 11, we use the runtimes of SSG and RSG to present the efficiency of Prilo*. As shown in Sec. 4.3, the runtime of SSG is related to the value of $\theta$, which




(a) *Slashdot* (b) *DBLP* (c) *Twitter*

Fig. 17. Runtimes of SSG by varying *PPCR*




(a) Efficiency comparison (b) Effectiveness comparison

Fig. 18. Performance of 10 workloads derived from *LDBC*

depends on the pruning power of BF and Twiglet. Note that $\theta$ is also defined as the *predicted positive condition rate* (*PPCR*), *i.e.*, $(TP + FP) / (TP + TN + FP + FN)$, where $TP$ (*resp.* $TN$) denotes *true positive* (*resp. true negative*), and $FP$ (*resp.* $FN$) denotes *false positive* (*resp. false negative*), respectively. Thus, we use *PPCR* to present the pruning powers of our proposed methods.

In Fig. 16, we use the ratio of RSG's runtime to SSG's runtime (*i.e.*, Prilo*'s speedup) as the $y$-axis and *PPCR* as the $x$-axis to present Prilo*'s improvement in efficiency when varying the number $k$ of Players. For presentation purpose, we cap the speedup at 100. Given large *PPCR*s, we can observe that the speedup is not sensitive to $k$. However, for small *PPCR*s, the speedup decreases when $k$ increases. This is because the number of positives is relatively small that increasing the number of Players cannot return the positives' results to User earlier. In Fig. 16(b), the speedup varies a lot for *hom* queries on *DBLP* due to the variation in Prilo*'s runtime on few candidate balls, where there is only 1 positive for most queries.

Fig. 17 shows the runtimes of SSG for both *hom* and *ssim* queries when $k$ varies. We can see that SSG's runtime decreases when $k$ increases. When *PPCR* is small, more Players cannot decrease SSG's runtime as there are few positives. Similar to Fig. 16(b), in Fig. 17(b), SSG's runtimes for *hom* queries on *DBLP* vary a lot for small *PPCR*s.

### 6.4 Experiments on *LDBC* Workloads

***LDBC* social network dataset.**[8] We used the value of *tag-class* as the label of each vertex. The transformed graph with scale factor 1 has 3,156,275 vertices, 10,375,137 edges and 213 labels.

**Queries.** To derive *LGPQ* queries from practical *LDBC* workloads [1], we made a few simplifications: We omitted the value predicates (*e.g.*, *date* and *name*) of vertex label, reachability queries, negations, trivial structures (*e.g.*, single edge), and well-known relationships (*e.g.*, between "*city*" and "*country*"). We obtained the structures of 10 out of 20 business intelligence workloads. Some characteristics of the workloads are reported in App. C of [51]. For each workload pattern, we generated a query by randomly assigning a label to each query vertex by using the *tag-class* of *LDBC* [1].

For presentation clarity, we report the efficiency and effectiveness under *hom* and *ssim* in Figs. 18(a) and 18(b). We can see from Fig. 18(a) that, for $W_3$, $W_4$, $W_5$, $W_9$ and $W_{12}$, the runtimes of

[8]https://ldbcouncil.org/benchmarks/snb/

Prilo* and Prilo are similar. This is because these queries have simple patterns (*e.g.*, 2-hop path) that Prilo* detects few spurious balls. Their *PPCR*s are larger than 0.5 as shown in Fig. 18(b) and hence, Prilo* applies RSG, as Prilo does. For $W_6$ under *ssim*, the *PPCR* is slightly smaller than 0.5 but Prilo*'s speedup is larger since the sizes of the non-spurious balls are much smaller than that of the spurious balls. For the rest queries, Prilo*'s speedup is obvious due to small *PPCR*s.

## 7 RELATED WORK

**Privacy preserving queries.** There have been works on privacy preserving query processing [4, 13, 30, 48, 49, 55] in recent years. For graph pattern queries, three kinds of privacy models are studied. These models differ in the privacy target about the following structure(s).

- *Query and data graph*: Cao et al. [8] studied tree pattern queries on encrypted *XML* documents by predetermining the traversal order for each query. Cao et al. [9] proposed a *filtering and verification* method to solve the *sub-iso* query over encrypted graph-structured data in cloud computing. Fan et al. [15] also studied the *sub-iso* query and proposed the *cyclic group based encryption scheme* (CGBE) to determine only the existence of matches. They did not consider ball retrieval.
- *Data graph only*: By utilizing the *k-automorphic graph*, Chang et al. [10] and Huang et al. [21] studied the *sub-iso* query semantic, while Gao et al. [17] studied the *strong simulation* query [37]. In this model, users' queries are known by SP.
- *Query only*: By using CGBE, Fan et al. [16] only answered *Yes/No* to the existence of matching subgraphs for the *sub-iso* query, while Xu et al. [52] studied the problem of the *ssim* query and only sketched a trivial baseline for retrieving the matching subgraphs.

**Secure query framework.** Fully homomorphic encryption (*FHE*) supports both addition and multiplication computations on encrypted messages. However, we cannot adopt *FHE* due to the known poor performance [18]. Goldreich et al. [19] introduced a compiler called oblivious *RAM* simulator (*ORAMs*). *ORAMs* can transform algorithms in a way that the input-output behavior of the original algorithm is preserved by the transformed algorithm. *ORAMs* cannot be adopted since the query cannot be known to SP. *GraphSC* [39], *GraphSE*$^2$ [28] and *PeGraph* [47] are frameworks for privacy preserving query on encrypted graph data, which also need the query structure for the matching process. The trusted execution environment is a recent solution [46, 50, 56]. The security is guaranteed by the hardware.

## 8 CONCLUSION

This paper proposes Prilo* for the problem of privacy preserving *localized graph pattern query* (*LGPQ*) processing in a cloud computing paradigm. Prilo* is a general framework that can handle *subgraph isomorphism*, *subgraph homomorphism*, and *strong simulation* semantics, where their query results are localized in subgraphs called balls. Moreover, Prilo* presents a BF pruning technique that exploits a *trusted execution environment* and a twiglet-based pruning technique under the ciphertext domain to securely compute the pruning messages of balls of the data graph. Based on these pruning messages, Prilo* proposes a ball retrieval scheme to enable User to privately retrieve from the service provider the balls that contain results early and hence, compute the query results early. Privacy analysis results and proof sketches are presented. The experimental results have shown the efficiency of Prilo*. As for future work, we plan to apply Prilo* to other *LGPQ*s, such as *p-homomorphism* query [42] and *conditional graph pattern* (*CGP*) query [14].

## ACKNOWLEDGMENTS

This work is supported by HKRGC GRF HKBU 12201119, 12201518, 12232716, and 12202221; HKRGC C2004-21GF; NSFC 62072132, and 92067104; NSF of China for Joint Fund Project U1936218; GD-NSF 2019B1515130001, and 2023A1515030273.

## REFERENCES


[1] Renzo Angles, János Benjamin Antal, Alex Averbuch, Altan Birler, Peter Boncz, Márton Búr, Orri Erling, Andrey Gubichev, Vlad Haprian, Moritz Kaufmann, et al. 2020. The LDBC social network benchmark. *arXiv* (2020).

[2] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'keeffe, Mark Stillwell, et al. 2016. SCONE: Secure linux containers with intel SGX. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Vol. 16. 689–703.

[3] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. 2019. SPEICHER: Securing LSM-based key-value stores using shielded execution. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST)*. 173–190.

[4] Johes Bater, Yongjoo Park, Xi He, Xiao Wang, and Jennie Rogers. 2020. Saqe: practical privacy-preserving approximate query processing for data federations. *Proceedings of the VLDB Endowment (PVLDB)* 13, 12 (2020), 2691–2705.

[5] Christian Behrends, Mathew E Sowa, Steven P Gygi, and J Wade Harper. 2010. Network organization of the human autophagy system. *Nature* 466, 7302 (2010), 68–76.

[6] Dan Bogdanov, Sven Laur, and Jan Willemson. 2008. Sharemind: A framework for fast privacy-preserving computations. In *13th European Symposium on Research in Computer Security (ESORICS)*. Springer, 192–206.

[7] Angela Bonifati, Wim Martens, and Thomas Timm. 2020. An analytical study of large SPARQL query logs. *The VLDB Journal (VLDBJ)* 29, 2-3 (2020), 655–679.

[8] Jianneng Cao, Fang-Yu Rao, Mehmet Kuzu, Elisa Bertino, and Murat Kantarcioglu. 2013. Efficient tree pattern queries on encrypted xml documents. In *Proceedings of the Joint EDBT/ICDT 2013 workshops (EDBT/ICDT)*. 111–120.

[9] Ning Cao, Zhenyu Yang, Cong Wang, Kui Ren, and Wenjing Lou. 2011. Privacy-preserving query over encrypted graph-structured data in cloud computing. In *IEEE 31st International Conference on Distributed Computing Systems (ICDCS)*. 393–402.

[10] Zhao Chang, Lei Zou, and Feifei Li. 2016. Privacy preserving subgraph matching on large graphs in cloud. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD)*. 199–213.

[11] Stephen A Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM Symposium on Theory of Computing (STOC)*. 151–158.

[12] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).

[13] Ningning Cui, Xiaochun Yang, Bin Wang, Jianxin Li, and Guoren Wang. 2020. SVkNN: Efficient secure and verifiable k-nearest neighbor query on the cloud platform. In *IEEE 36th International Conference on Data Engineering (ICDE)*. 253–264.

[14] Grace Fan, Wenfei Fan, Yuanhao Li, Ping Lu, Chao Tian, and Jingren Zhou. 2020. Extending graph patterns with conditions. In *Proceedings of the 2020 International Conference on Management of Data (SIGMOD)*. 715–729.

[15] Zhe Fan, Byron Choi, Qian Chen, Jianliang Xu, Haibo Hu, and Sourav S. Bhowmick. 2015. Structure-preserving subgraph query services. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 27, 8 (2015), 2275–2290.

[16] Zhe Fan, Byron Choi, Jianliang Xu, and Sourav S. Bhowmick. 2015. Asymmetric structure-preserving subgraph queries for large graphs. In *IEEE 31st International Conference on Data Engineering (ICDE)*. 339–350.

[17] Jiuru Gao, Jiajie Xu, Guanfeng Liu, Wei Chen, Hongzhi Yin, and Lei Zhao. 2018. A privacy-preserving framework for subgraph pattern matching in cloud. In *Database Systems for Advanced Applications: 23rd International Conference (DASFAA)*. 307–322.

[18] Craig Gentry and Dan Boneh. 2009. *A fully homomorphic encryption scheme*.

[19] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.

[20] Haibo Hu, Jianliang Xu, Qian Chen, and Ziwei Yang. 2012. Authenticating location-based services without compromising location privacy. In *Proceedings of the 2012 International Conference on Management of Data (SIGMOD)*. 301–312.

[21] Kai Huang, Haibo Hu, Shuigeng Zhou, Jihong Guan, Qingqing Ye, and Xiaofang Zhou. 2021. Privacy and efficiency guaranteed social subgraph matching. *The VLDB Journal (VLDBJ)* (2021), 1–22.

[22] Qin Jiang, Yong Qi, Saiyu Qi, Wenjia Zhao, and Youshui Lu. 2020. Pbsx: A practical private boolean search using Intel SGX. *Information Sciences* 521 (2020), 174–194.

[23] Hyunjoon Kim, Yunyoung Choi, Kunsoo Park, Xuemin Lin, Seok-Hee Hong, and Wook-Shin Han. 2021. Versatile equivalences: Speeding up subgraph query processing and subgraph matching. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*. 925–937.

[24] Hyunjoon Kim, Yunyoung Choi, Kunsoo Park, Xuemin Lin, Seok-Hee Hong, and Wook-Shin Han. 2022. Fast subgraph query processing and subgraph matching via static and dynamic equivalences. *The VLDB Journal (VLDBJ)* (2022), 1–26.

[25] Hyeong-Il Kim, Hyeong-Jin Kim, and Jae-Woo Chang. 2019. A secure kNN query processing algorithm using homomorphic encryption on outsourced database. *Data & Knowledge Engineering (DKE)* 123 (2019), 101602.

[26] Jinha Kim, Hyungyu Shin, Wook-Shin Han, Sungpack Hong, and Hassan Chafi. 2015. Taming subgraph isomorphism for RDF query processing. *Proceedings of the VLDB Endowment (PVLDB)* 8, 11 (2015), 1238–1249.

[27] Daniel J Klionsky. 2008. Autophagy revisited: a conversation with Christian de Duve. *Autophagy* 4, 6 (2008), 740–743.

[28] Shangqi Lai, Xingliang Yuan, Shi-Feng Sun, Joseph K Liu, Yuhong Liu, and Dongxi Liu. 2019. GraphSE$^2$: An encrypted graph database for privacy-preserving social search. In *Proceedings of the 2019 Asia Conference on Computer and Communications Security (AsiaCCS)*. 41–54.

[29] Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. 2012. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *Proceedings of the VLDB Endowment (PVLDB)* 6, 2 (2012), 133–144.

[30] Xinyu Lei, Alex X Liu, Rui Li, and Guan-Hua Tu. 2019. SecEQP: A secure and efficient scheme for SkNN query problem over encrypted geodata on cloud. In *IEEE 35th International Conference on Data Engineering (ICDE)*. 662–673.

[31] Jure Leskovec and Andrej Krevl. 2014. SNAP datasets: Stanford large network dataset collection. http://snap.stanford.edu/data.

[32] Yin Li, Dhrubajyoti Ghosh, Peeyush Gupta, Sharad Mehrotra, Nisha Panwar, and Shantanu Sharma. 2021. Prism: private verifiable set computation over multi-owner outsourced databases. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*. 1116–1128.

[33] Yehuda Lindell and Jonathan Katz. 2014. *Introduction to modern cryptography*.

[34] An Liu, Kai Zhengy, Lu Liz, Guanfeng Liu, Lei Zhao, and Xiaofang Zhou. 2015. Efficient secure similarity computation on encrypted trajectory data. In *IEEE 31st International Conference on Data Engineering (ICDE)*. 66–77.

[35] Jinfei Liu, Juncheng Yang, Li Xiong, and Jian Pei. 2017. Secure skyline queries on cloud platform. In *IEEE 33rd International Conference on Data Engineering (ICDE)*. 633–644.

[36] Rongxing Lu, Xiaodong Lin, Zhiguo Shi, and Jun Shao. 2014. PLAM: A privacy-preserving framework for local-area mobile social networks. In *2014 IEEE International Conference on Computer Communications (INFOCOM)*. 763–771.

[37] Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai, and Tianyu Wo. 2014. Strong simulation: Capturing topology in graph pattern matching. *ACM Transactions on Database Systems (TODS)* 39, 1 (2014), 1–46.

[38] Robin Milner. 1989. *Communication and concurrency*.

[39] Kartik Nayak, Xiao Shaun Wang, Stratis Ioannidis, Udi Weinsberg, Nina Taft, and Elaine Shi. 2015. GraphSC: Parallel secure computation made easy. In *2015 IEEE Symposium on Security and Privacy (S&P)*. 377–394.

[40] Miles Ohlrich, Carl Ebeling, Eka Ginting, and Lisa Sather. 1993. Subgemini: Identifying subcircuits using a fast subgraph isomorphism algorithm. In *Proceedings of the 30th International Design Automation Conference (DAC)*. 31–37.

[41] Pascal Paillier. 1999. 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 (EUROCRYPT)*. 223–238.

[42] Yinglong Song, Huey Eng Chua, Sourav S Bhowmick, Byron Choi, and Shuigeng Zhou. 2018. BOOMER: Blending visual formulation and processing of p-homomorphic queries on large networks. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*. 927–942.

[43] Einat Sprinzak, Shmuel Sattath, and Hanah Margalit. 2003. How reliable are experimental protein–protein interaction data? *Journal of Molecular Biology (JMB)* 327, 5 (2003), 919–923.

[44] Yuanyuan Tian and Jignesh M. Patel. 2008. Tale: A tool for approximate large graph matching. In *IEEE 24th International Conference on Data Engineering (ICDE)*. 963–972.

[45] Julian R Ullmann. 1976. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)* 23, 1 (1976), 31–42.

[46] Sheng Wang, Yiran Li, Huorong Li, Feifei Li, Chengjin Tian, Le Su, Yanshan Zhang, Yubing Ma, Lie Yan, Yuanyuan Sun, et al. 2022. Operon: an encrypted database for ownership-preserving data management. *Proceedings of the VLDB Endowment (PVLDB)* 15, 12 (2022), 3332–3345.

[47] Songlei Wang, Yifeng Zheng, Xiaohua Jia, and Xun Yi. 2022. PeGraph: A system for privacy-preserving and efficient search over encrypted social graphs. *IEEE Transactions on Information Forensics and Security (TIFS)* 17 (2022), 3179–3194.

[48] Songlei Wang, Yifeng Zheng, Xiaohua Jia, and Xun Yi. 2022. Privacy-preserving analytics on decentralized social graphs: The case of eigendecomposition. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 1 (2022), 1–15.

[49] Songrui Wu, Qi Li, Guoliang Li, Dong Yuan, Xingliang Yuan, and Cong Wang. 2019. Servedb: Secure, verifiable, and efficient range queries on outsourced database. In *IEEE 35th International Conference on Data Engineering (ICDE)*. 626–637.

[50] Cheng Xu, Ce Zhang, Jianliang Xu, and Jian Pei. 2021. SlimChain: scaling blockchain transactions through off-chain storage and parallel processing. *Proceedings of the VLDB Endowment (PVLDB)* 14, 11 (2021), 2314–2326.

[51] Lyu Xu, Byron Choi, Yun Peng, Jianliang Xu, and Sourav S Bhowmick. 2023. Technical report: A framework for privacy preserving localized graph pattern query processing. https://www.comp.hkbu.edu.hk/%7Ecslyuxu/prilo2023tr.pdf.

[52] Lyu Xu, Jiaxin Jiang, Byron Choi, Jianliang Xu, and Sourav S Bhowmick. 2021. Privacy preserving strong simulation queries on large graphs. In *IEEE 37th International Conference on Data Engineering (ICDE)*. 1500–1511.

[53] Xifeng Yan, Philip S Yu, and Jiawei Han. 2004. Graph indexing: a frequent structure-based approach. In *Proceedings of the 2004 International Conference on Management of Data (SIGMOD)*. 335–346.
[54] Can Zhang, Liehuang Zhu, Chang Xu, Kashif Sharif, Chuan Zhang, and Ximeng Liu. 2020. PGAS: Privacy-preserving graph encryption for accurate constrained shortest distance queries. *Information Sciences* 506 (2020), 325–345.
[55] Yandong Zheng, Rongxing Lu, Yunguo Guan, Songnian Zhang, Jun Shao, and Hui Zhu. 2022. Efficient and privacy-preserving similarity query with access control in eHealthcare. *IEEE Transactions on Information Forensics and Security (TIFS)* 17 (2022), 880–893.
[56] Wenchao Zhou, Yifan Cai, Yanqing Peng, Sheng Wang, Ke Ma, and Feifei Li. 2021. Veridb: An sgx-based verifiable database. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*. 2182–2194.
[57] Lei Zou, Lei Chen, and M Tamer Özsu. 2009. Distance-join: Pattern match query in a large graph database. *Proceedings of the VLDB Endowment (PVLDB)* 2, 1 (2009), 886–897.

Received October 2022; revised January 2023; accepted February 2023