A Framework for Privacy Preserving Localized Graph Pattern Query Processing

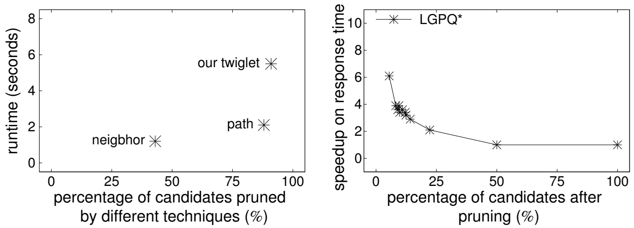
ABSTRACT

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*, which have a restriction on the size of the subgraph to be matched, called *balls*. To provide privacy preserving query service for *LGPQ*s, this paper proposes a general framework, called LGPQ, which enables users to privately obtain the query results. To further optimize LGPQ, we propose LGPQ* that comprises the first bloom filter for trees in the trust execution environment (*TEE*) on the 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 LGPQ* is on average 4x faster than the baseline, and meanwhile, preserves query privacy.

1 INTRODUCTION

Many graph pattern queries have been proposed in the literature (*e.g.*, [12, 41, 50]), and used in many recent applications, such as social network analysis, biology analysis, electronic circuit design, and chemical compound search [43, 48, 49, 59, 64]. On one hand, graph patterns often have high computational complexities. On the other hand, graph pattern results that span through small subgraphs can be preferred in applications where humans would interpret the results. Hence, localized graph pattern queries (*LGPQ*), such as *subgraph homomorphism query* (*hom*) [30], *subgraph isomorphism query* (*sub-iso*) [12], and *strong simulation query* (*ssim*) [40], whose semantics require a size restriction on the matched patterns, have recently received much attention, *e.g.*, [19, 25, 47, 57].

As data owners and query users 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 the *data owner* and *user*, including elasticity, high availability, and cost savings.



(a) Average performance of oblivious pruning by using different topologies: neighbor's label [17], label sequences (paths) [57], and our twiglets from experiments

(b) Speedup on *Slashdot*: the ratio of LGPQ*'s runtime (the response time for user to obtain first query results from SP) to the runtime without LGPQ*'s optimizations

Figure 1: Some highlights of performance of LGPQ*

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 this paper, we study the problem of *LGPQ*s on a large data graph while preserving the privacy of the structural information of the query in database outsourcing. In the example below, we illustrate the efficiency and query privacy challenges of this problem.

EXAMPLE 1. *(Privacy preserving* LGPQ *in data outsourcing) Consider a biotechnology company whose competitive advantages are its biological discoveries. The company has recently found a potentially valuable autophagy pattern. To explore the autophagy patterns with similar structure as the found one, the company therefore sends* LGPQ*s to a third party* SP *who has a powerful IT infrastructure to host a publicly known large protein-protein interaction (*PPI*) network for autophagy interaction in human cells. However, users do not want to expose 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 [18, 39, 57].* □

Existing works [18, 57] consider privacy preserving *LGPQ* under *individual* semantic, in particular, *sub-iso* queries [18] and *ssim* queries [57], 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 [57], 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 a SP. However, there are two main challenges.

(1) *What are general privacy preserving steps for* LGPQ*s?*
(2) *How to design optimizations for* LGPQ*s that preserve privacy?*

To address the first challenge, we propose a privacy preserving framework, called LGPQ, that comprises the general steps, namely, *candidate enumeration*, *query verification*, and *query matching*. Also, we adopt the idea of *ball* [40], a subgraph of data graph defined by its center and radius, as the unit (superset) of results of *LGPQ* for users to retrieve query matches. Privacy preserving query processing (*e.g.*, the enumeration and verification steps) is localized in balls,

SIGMOD '2023, June 18–23, 2023, Seattle, WA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06...$15.00
https://doi.org/XXXXXXX.XXXXXXX

instead of the whole graphs, and therefore, can be efficient. The balls can also be precomputed, encrypted, and stored on the SP side.

To implement the privacy preserving algorithms for *LGPQ*, homomorphic encryption schemes may be applied. However, it has been known that fully homomorphic encryption (*FHE*) [20] is inefficient. A partial homomorphic encryption (*PHE*), specifically, an asymmetric encryption scheme called *Paillier* [44], has been applied for *NN* or *kNN* queries [14, 15, 29]. This paper adopts a more efficient symmetric encryption scheme called *cyclic group based encryption* (*CGBE*) [17]. Specifically, we use *CGBE* in a *query-oblivious* manner to detect the violations of the *LGPQ* semantics on the balls that may contain matches (a.k.a *candidate balls*). The candidate balls that do not have violations contain query matches. We design LGPQ by using two different kinds of SP servers that enable users to retrieve the encrypted data of such balls while the SP cannot deduce the sensitive information even from the ball retrieval patterns. After receiving such balls, users decrypt them, and compute the matches simply using the existing algorithms on plaintext.

Regarding the second challenge, we note that there can be *spurious balls*, *i.e.*, candidate balls that *definitely* contain no matches, and they should be *"pruned"*, *i.e.*, *should not sent all candidate balls* to users. We propose to construct a *pruning message* that encrypts whether a candidate ball is spurious or not. It is hence important to detect the spurious balls as many as possible and compute pruning messages in a query oblivious way.

The first technique is to exploit the *trusted execution environment* (*TEE*), *e.g.*, the *Intel software guard extensions* (SGX) [13]. 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 query by users for pruning inside the enclave of SGX. It is known that SGX's enclave has a limited memory space. Hence, we propose to use bloom filters inside the enclave for space-efficient pruning. The second technique is to propose a small structure called *twiglet* for query-oblivious pruning under the ciphertext domain, without the need of *TEE*. While previous work proposed simpler topologies for query-oblivious pruning, as shown in Fig. 1(a), *twiglet* can further enhance pruning at the expense of an additional runtime. To balance the pruning and runtime, users can tune the size of *twiglet*s.

Next, we propose an algorithm for the SP to arrange the sequence to evaluate candidate balls by using LGPQ, which enables users to retrieve the encrypted balls that contain matches as early as possible, as opposed to at the end of the whole query processing. From our preliminary experiments, we observe that on average, only 15% candidate balls contain matches. On the SP side, a host server makes use of the pruning messages to generate sequences of candidate balls mixed with dummy balls, where candidate balls containing matches are placed in front in a non-trivial way. The other servers on SP 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 containing matches early, while query's privacy is preserved from the SP. By using these optimization ideas, LGPQ* can achieve a 4x speedup on *Slashdot* (Fig 1(b)).

Contributions. The contributions of this paper are as follows.

- We propose the first *secure* general framework called LGPQ that as long as the query is a *LGPQ*, LGPQ supports its privacy query processing. LGPQ consists of three general steps, namely candidate enumeration, query verification, and query matching.
- We propose an optimized framework called LGPQ* that comprises i) a *bloom filter* checking that exploits *TEE* for pruning, ii) a query-oblivious pruning by using twiglets under the ciphertext domain, which does not need *TEE*, and iii) the first secure scheme that fetches the candidate balls to the user early, which can significantly save the waiting time of the *user*.
- We present the results of the privacy analysis on LGPQ*. The detailed proofs are presented in Appendix B of [1].
- Our experiments verified that LGPQ*'s pruning techniques outperform the state-of-the-art 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.

Organization. The preliminaries and the problem statement are presented in Sec. 2. Sec. 3 presents the LGPQ framework. LGPQ*'s optimizations, the pruning techniques together with a secure scheme for ball retrieval, are presented in Sec. 4. Sec. 5 reports the privacy analysis results. 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 provides a concise summary of the preliminaries.

Graph. A *graph* is denoted by $G = (V_G, E_G, \Sigma_G, L_G)$, where V_G, E_G, Σ_G, and L_G are the sets of vertices, directed edges, labels, and the function for matching the vertices to their labels, respectively. (u,v) denotes the directed edge from vertex u to v, and $L_G(u)$ denotes the label of u. For graph G, the *distance* between two vertices u and v in G, denoted by $\text{dis}(u, v)$, is the length of the shortest undirected paths from u to v in G [40], and the *diameter* of G, denoted by d_G, is the largest distance between any pairs of vertices of G.

Ball [40]. 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 vertex u in G as *center*, and r as the *radius*, s.t. i) $V_B = \{v | v \in V_G, \text{dis}(u, v) \leq r\}$, ii) E_B has the edges that appear in G over the same vertices in V_B, and iii) $\Sigma_B = \{L_G(v) | v \in V_B\}$.

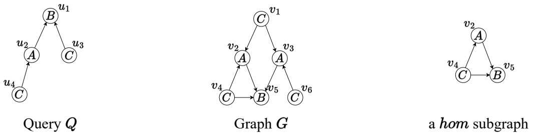
Adjacency matrix. The *adjacency matrix* of graph G, denoted by M_G, is a $|V_G| \times |V_G|$ matrix. Given a vertex u (*resp.* vertex v) that locates in the i^{th} row (*resp.* the j^{th} column) of a matrix M, $M(i, j)$ is also denoted by $M(u, v)$ for simplicity. Then, $M_G(i, j) = 1$ if $(u, v) \in E_G$. Otherwise, $M_G(i, j) = 0$. We may omit the subscript, such as G, 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* [17, 57]. We illustrate this with *hom* as follows.

Definition 1. ***(Subgraph homomorphism (*****hom*****))*** *Given a connected query graph Q and a data graph G, a* subgraph homomorphism *of Q in G is a match function $\mathcal{H}$: $V_Q \rightarrow V_G$ that satisfies the following conditions.*

(1) $\forall u \in V_Q$, $L_Q(u) = L_G(\mathcal{H}(u))$; and

(2) $\forall u, v \in Q$, $M_Q(u, v) = 1 \Rightarrow M_G(\mathcal{H}(u), \mathcal{H}(v)) = 1$. □



Query Q Graph G a *hom* subgraph

Figure 2: An example for a *hom* subgraph in G of Q

Sub-iso can be defined by modifying the match function $\mathcal{H}$ of Def. 1 with an injective function [17].[1] Due to space restrictions, we present the definition of *ssim* [57] in Appendix A.1 of the technical report [1]. 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 graph G in Fig. 2. The induced subgraph of $\{v_2,v_4,v_5\}$ at the RHS of Fig. 2 is a matching subgraph for Q under* hom*, where $\mathcal{H}(u_1) = v_5$, $\mathcal{H}(u_2) = v_2$, $\mathcal{H}(u_3) = v_4$, and $\mathcal{H}(u_4) = v_4$.*

From Def. 1 and Example 2, we can easily see that each matching subgraph for a *hom* query exists in a ball with a radius equal to d_Q. We also note that this holds for *sub-iso* and *ssim* queries. 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 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

Cyclic group based encryption (CGBE) **[17]**. CGBE is a *CPA-secure* symmetric encryption scheme. The private key pk is held by the user. The notations $E(m)$ and $D(m)$ are used to denoted the encryption and decryption of message m, respectively. CGBE supports the following homomorphic operations.

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

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

where m_1, $m_2 \in \mathbb{Z}_p$, and r_1 and r_2 are two random numbers. CGBE requires $m_1 + m_2$ and $m_1 \cdot m_2$ are smaller than a large public prime p, or there is an overflow error, *i.e.*, the above operations are no longer correct [57]. To make use of these operations, we encode 0 as rq where r is a random integer and q ($q \ll p$) is a predefined prime; and a non-zero as other values. By detecting whether there exists a factor q in CGBE's decrypted message, we can determine whether the plaintext of the ciphertext is a zero or not, but, we cannot recover the plaintext. According to [17], the probability of obtaining a false zero from CGBE's decryption is negligible.

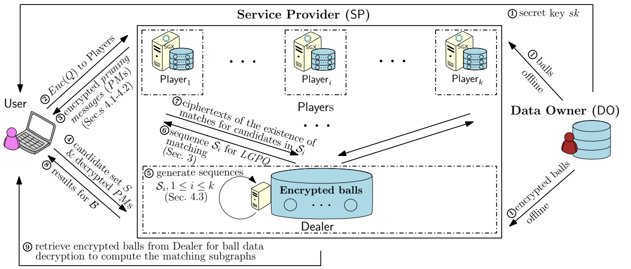

Service Provider (SP)
① secret key sk
User
② $Enc(Q)$ to Players
③ encrypted pruning messages (PMs) (Sec.s 4.1-4.2)
Player$_1$
Player$_i$
Player$_k$
SGX
Players
⑦ ciphertexts of the existence of matches for candidates in $\mathcal{S}_i$
⑥ sequence $\mathcal{S}_i$ for LGPQ matching (Sec. 3)
④ candidate set $\mathcal{S}$ & decrypted PMs
⑧ results for $\mathcal{B}$
⑤ generate sequences $\mathcal{S}_i, 1 \leq i \leq k$ (Sec. 4.3)
Encrypted balls
Dealer
① balls offline
Data Owner (DO)
① encrypted balls offline
⑨ retrieve encrypted balls from Dealer for ball data decryption to compute the matching subgraphs

Figure 3: The system model of LGPQ*

Intel software guard extensions (SGX) **[13].** Recently, secure co-processors have been found efficient and effective in building secure applications. In particular, modern *Intel* CPUs have supported SGX, a set of x86 instruction set architecture extensions, for a trusted execution environment (*TEE*). SGX provides users a secure and isolated hardware container called *enclave*. A secure channel is established between the user and the enclave. A user encrypts the query, *e.g.*, by using a symmetric-key algorithm, and sends the encrypted query into the enclave for secure computation. The secure memory region in SGX is approximately 128 MB [2, 3, 56]. However, the cost of interacting with the enclave is huge that it is desirable to design space-efficient techniques when applying SGX for secure computation.

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 adopt the commonly used system model on outsourced databases [14, 33, 55] that includes *data owner*, *user*, and *service provider*. Fig. 3 shows an overview of our system model.

- **Data owner** (DO). A data owner first generates a secret key sk and all balls of data 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. sk can be obtained by authorized users only.
- **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 encrypted 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 data of target balls from the service provider. Finally, User decrypts the encrypted data by sk and computes the matching subgraphs for the query.
- **Service provider** (SP). We have extended the SP of the widely used system model [14, 33, 55], 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 (Player) 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 results of balls in $\mathcal{S}_i$ back to Dealer and then, Dealer ⑧ sends the results of balls in S to User.

Security model. This paper assumes the SP is honest but curious, *a.k.a.* the *semi-honest adversary model* that is widely adopted in

[1] In some existing works [30, 32], the match function $\mathcal{H}$ of *hom* (or *sub-iso*) also requires that the labels of the edges (u, v) and $\big(\mathcal{H}(u), \mathcal{H}(v)\big)$ 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.

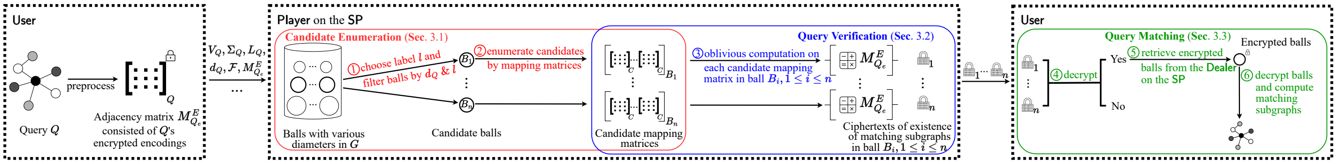

User
preprocess
Q
Query Q
Adjacency matrix $M^E_{Q_e}$ consisted of Q's encrypted encodings
$V_Q, \Sigma_Q, L_Q,$
$d_Q, \mathcal{F}, M^E_{Q_e}$
...
Player on the SP
Candidate Enumeration (Sec. 3.1)
①choose label l and filter balls by d_Q & l
B_1
B_n
②enumerate candidates by mapping matrices
Balls with various diameters in G
Candidate balls
C
B_1
B_n
Candidate mapping matrices
Query Verification (Sec. 3.2)
③oblivious computation on each candidate mapping matrix in ball $B_i, 1 \leq i \leq n$
$M^E_{Q_e}$
Ciphertexts of existence of matching subgraphs in ball $B_i, 1 \leq i \leq n$
1
n
User
Query Matching (Sec. 3.3)
④decrypt
Yes
No
⑤retrieve encrypted balls from the Dealer on the SP
Encrypted balls
⑥decrypt balls and compute matching subgraphs

Figure 4: Overview of the LGPQ framework

the literature [9, 10, 24, 36]. 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. 3, there are multiple Players and a Dealer on the SP side. Unlike the existing multi-party computation (*MPC*) [5] where the servers need to communicate with others, Players only communicate with Dealer and Players do not collude with each other, which is similar to a recent work [35]. Regarding the communications between Players and Dealer, we adopt the commonly used collude-resistant model [14, 29, 37, 38] that Dealer and Player do not collude. We also assume the cloud servers on the SP side adopt the *chosen plaintext attack* (*CPA*) [36], *i.e.*, the adversaries can choose arbitrary plaintexts to obtain their ciphertexts to gain sensitive information. The *privacy targets* of this paper are described as follows.

- **Query privacy.** The structural information of User's query graph Q, *i.e.*, the value of each element in Q's adjacency matrix.
- **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 data 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 LGPQ FRAMEWORK

In this section, we *propose* a general framework called LGPQ for handling *LGPQ*s. Fig. 4 shows the overview of LGPQ. An *LGPQ* can be answered by three generic steps, namely *candidate enumeration*, *query verification*, and *query matching*, as follows.

(1) **Candidate enumeration.** Given an *LGPQ* Q with V_Q, Σ_Q, L_Q, diameter d_Q, query semantic $\mathcal{F}$ and adjacency matrix $M^E_{Q_e}$ consisted of Q's encrypted encodings, Players ① arbitrarily choose a label l in Σ_Q and filter all balls of graph G by using d_Q and l to obtain the *candidate ball*s that may have matches. Then, Players ② enumerate candidate subgraphs from each candidate ball for query verification.

(2) **Query verification.** For each candidate subgraph G_c, Players ③ verify whether G_c can be matched to Q by conducting query-oblivious computation on the ciphertexts in $M^E_{Q_e}$ of Q. The computed ciphertext results are sent to User.

(3) **Query matching.** User ④ decrypts the received ciphertext results. Then User ⑤ securely retrieves the corresponding encrypted balls from Dealer that passed the verification, and ⑥ decrypts these balls to compute the matching subgraphs.

In the following subsections, we elaborate these steps with the query semantic of *hom*, 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 for filtering redundant balls. The proofs are provided in Appendix A.2 of [1].

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) $v \in G_s$.*

By Prop. 1, the candidate enumeration can choose arbitrarily a label l from Σ_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. 4). 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) $L_G(w') = l$.*

With Prop.s 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 since the matching subgraph of B that does not contain B's center can be found in other balls. We remark that Prop.s 1-2 can be also applied to *sub-iso* and *ssim*. For illustration, we focus on *hom*. Given a *hom* query Q and a ball B, to enumerate all candidate subgraphs of B for Q, 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) $\forall\ w \in V_G - \{v\}$, $C(u, w) = 0$.*

Then, we present the algorithm to enumerate all CMMs of candidate subgraphs. Taking a query Q with V_Q, Σ_Q, L_Q and diameter d_Q, and a ball B with 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 2, the current CMM C is initialized as a zero matrix. The optimizations [18] for minimizing the size of query Q (*resp.* ball B) on the User (*resp.* Player) side are applied in Line 3. $\forall\ u \in V_Q$, Lines 6-9 generate a vertex set $CV(u)$ that contains the vertices of

[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 process* [40].

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      $(Q, B) =$ opt($Q, B$); //opt():optimizations in [18]
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 $R_1$;
```

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  $r \leftarrow 1$; //result initialization
2  $M_p \leftarrow C \cdot M_G \cdot C^{\mathrm{T}}$; //$M_p$:$G'$s adjacency matrix projected by $C$
3  foreach $i \in [1, |V_Q|]$ do
4      foreach $j \in [1, |V_Q|]$ do
5          if $M_p(i, j) = 0$ then
6              $r \leftarrow r \cdot M_{Q_e}(i, j)$; //matching violation aggregation
7  return $r$;
```

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^{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^{th} row. Line 18 recursively calls Alg. 1 for the $(i+1)^{\text{th}}$ row’s enumeration. Note that the *sub-iso* semantic can be handled by adding one *if* clause between Line 16 and Line 17 to ensure that there exists at most one value 1 in each column of matrix C. 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 candidate subgraph that Line 12 returns an empty set. Finally, Line 20 returns R_1 as output.

Analysis. In Appendix A.2 of [1], we present the proof that Alg. 1 is query-oblivious. For the time complexity, Lines 6-9 take $O\big(|V_Q| \cdot |V_B|\big)$ time. Lines 16-19 enumerate $O\big(\sum_{v \in V_Q} |CV(v)|^{|V_Q|}\big)$ CMMs. The optimizations (Line 3) take negligible time when 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. the $\mathcal{F}$’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 [18].)

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^{T} denotes the transpose matrix of C. For each encoding $M_{Q_e}(i,j)$ (Lines 3-4), if $M_p(i,j) = 0$ that may lead to a matching violation on condition (2) of Def. 1 (Line 5), Line 6 uses multiplication to aggregate $M_{Q_e}(i,j)$ into a result r. Line 7 returns r as output.

Analysis. The correctness and query-obliviousness of Alg. 2 are presented in Appendix A.2 of [1]. The encodings can be encrypted by CGBE to preserve the query privacy when 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 $O\big(|V_G|^3\big)$ time when Lines 3-6 take $O\big(|V_G|^2\big)$ time. In practice, $|V_G|$ equals to 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 LGPQ framework, as shown in Fig. 4.

- **On the** Players**.** Taking a query with V_Q, Σ_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 Σ_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[3] 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 Rs of ciphertext results for all candidate balls of G in Line 9. Note that the evaluation of a ball is independent to each other and hence, can be readily parallelized.
- **On the** User**.** The query matching step is as follows. User ④ decrypts the ciphertexts in the received ciphertext result set R (Lines

[3] Label l that maximizes the number of candidate balls, in practice, can often avoid enumerating a large number of mappings from query’s vertices of label l.

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

```
Input  : A query $Q$ with $V_Q, \Sigma_Q, L_Q$, diameter $d_Q$ and adjacency matrix $M^E_{Q_e}$ consisted
         of $Q$'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  $l \leftarrow \arg\max_{l' \in \Sigma_Q} \{|\{v | v \in V_G$ and $L_G(v) = l'\}|\}$; //opt: choose label $l$
3  foreach $v_i \in \{v | v \in V_G$ 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      $CS_i \leftarrow$ CanEnum($Q, v_i, B_i, 0, 0, \emptyset$); //②:candidate enumeration
6      foreach $C \in CS_i$ do
7          $r_i \leftarrow$ 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 by 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 $R_H$;
```

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 by 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 [27, 28, 40]) and outputs these subgraphs as query answers (Lines 16-17).

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 LGPQ FRAMEWORK

To optimize LGPQ, we propose an optimized framework called LGPQ* that i) enables Players to detect as many as possible the *spurious* balls whose centers cannot be matched to any vertices of the query when preserving the privacy target and record them in the *pruning messages* (*PMs*) of balls (Sec.s 4.1-4.2), and ii) enables User to *early* obtain the balls contain matching subgraphs in a query-oblivious manner by using these *PMs* (Sec. 4.3).

4.1 Bloom Filter of Trees in *TEE*

In this subsection, we present our first pruning technique. Different from existing works [18, 57] that compute the *PMs* by using simple graph topologies (*e.g.*, *neighbors* and *paths*), we propose using the *tree* topology. To overcome the large time cost and ciphertext overflow problem [57] of the homomorphic computation for a tree-based pruning under the ciphertext domain, we make use of the trusted execution environment (*TEE*). In particular, we equip Players with SGXs, which enable secure computation under the plaintext domain inside SGX's enclave to alleviate the time cost. 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 make use of small binary tree structures to propose the *h-label binary trees* for detecting spurious balls.

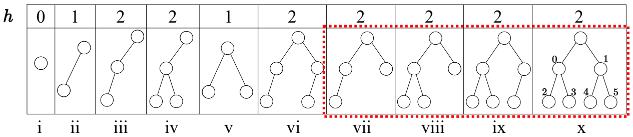
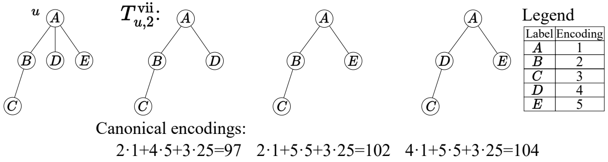


Figure 5: h-label binary trees ($h \leq 2$) (labels omitted)



(a) Graph G (b) $\mathbf{2}$-label binary trees of topology vii in G

Figure 6: Example of 2-label binary tree

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 a binary subtree 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,* $v \neq v' \Rightarrow L_G(v) \neq L_G(v')$.

Fig. 5 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, $i \in \{\text{i}, ..., \text{x}\}$. For example, Fig. 6(b) shows three $T^{\text{vii}}_{u,2}$s for vertex u in Fig. 6(a). Then, we have the following proposition for pruning.

Proposition 3. *Consider a height h, a query Q, and a ball B with the center w. If there exists at least one* $T^i_{v,h}$, $i \in \{\text{i}, ..., \text{x}\}$ *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, then w cannot be matched to v under* sub-iso*,* hom *and* ssim.

Recall that B is spurious if B's center cannot be matched to any vertices of Q. Therefore, 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 vs and record the existence in a ciphertext c_{sgx} as one of the *PM* of B. In the following, we i) first present the algorithm to enumerate the 2-label binary trees of an input graph, and then ii) propose the BF pruning technique that makes use of *both bloom filter* and SGX to efficiently check the existence of 2-label binary trees for a ball and hence obtain its c_{sgx}.

4.1.1 Enumeration of 2-label binary trees. In Fig. 5, 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 algorithm 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 $T^i_{w,2}$, $i \in \{\text{vii}, ..., \text{x}\}$. Firstly, for each neighbor u of w, Lines 1-2 use *BFS* to obtain 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 any two neighbors u and v of w whose labels are different from w's label (Lines 3-4), Line 5 enumerates all cases for generating subtrees with root w by considering u (*resp.* v) as the left (*resp.* right) child of w. In detail, Line 6 computes the number n_u (*resp.* n_v) of distinct labels of u's neighbors (*resp.* v's neighbors). n_u and n_v are used to determine the topologies of subtrees to be enumerated. As topologies vii-x shown in Fig. 5, there always exists at least one child of the root's left child. Hence, we

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 $L_G(u) \neq L_G(w)$ do
2      start a BFS from $u$ to obtain the set $\mathcal{L}(u)$;
3  foreach neighbor $u$ of $w$ that $L_G(u) \neq L_G(w)$ do // $u$: $w'$s left child
4      foreach neighbor $v$ of $w$ that $L_G(v) \neq L_G(w)$ do // $v$: $w'$s right child
5          CaseEnum($u$, $v$, $\mathcal{L}$);
   Procedure CaseEnum($u$, $v$, $\mathcal{L}$):
6  $n_u \leftarrow |\mathcal{L}(u) - \{L_G(v)\}|$, $n_v \leftarrow |\mathcal{L}(v) - \{L_G(u)\}|$;
7  if $n_u = 1$ then
8      TreeEnum($u$, $v$, vii); //enumerate subtrees for $T^{\text{vii}}_{w,2}$
9  if $n_u \geq 2$ then
10     foreach $i \in$ {vii, viii} do
11         TreeEnum($u$, $v$, $i$); //cases for $T^{\text{vii}}_{w,2}$ and $T^{\text{viii}}_{w,2}$
12     if $n_v = 1$ then
13         TreeEnum($u$, $v$, ix); //case for $T^{\text{ix}}_{w,2}$
14     if $n_v \geq 2$ then
15         TreeEnum($u$, $v$, x); //case for $T^{\text{x}}_{w,2}$
   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 ;
```

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

Table 1: No.s of 2-label binary trees for distinct topologies of ball B (d_{max}: max. degree of vertices of B)

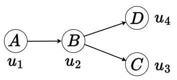


Figure 7: Example of h-twiglet (where h = 3): $\big[A, B, [C, D]\big]$

can stop the enumeration for u and v if $n_u = 0$. 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's left child u, right child v, and topology i, $i \in$ {vii, ..., x}, Line 16 enumerates G's subtrees of topology i with height 2, which can be used to project $T^{i}_{w,2}$s.

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 Bloom filter (BF) checking inside SGX's enclave.

We adopt BF to build a time- and space-efficient index used for securely checking 2-label binary trees' existence inside SGX's enclave, due to the limited memory space of the enclave.

Canonical encoding of h-label binary tree. Assume there is a *canonical encoding* of labels and h-label binary trees such that if two trees are isomorphic, their encodings are identical. Fig. 6 presents an example of converting three $T^{\text{vii}}_{u,2}$s into encodings. For graph G in Fig. 6(a), where $|\Sigma_G| = 5$, assume the encoding of labels A, B, C, D and E are 1, 2, 3, 4 and 5, respectively. We propose further *each position in a topology has a unique index*, as shown in Fig. 5's topology x. From the label encoding and the index, we can compute that the canonical encoding[4] of the leftmost $T^{\text{vii}}_{u,2}$ in Fig. 6(b) is $2{\cdot}5^0 + 4{\cdot}5^1 + 3{\cdot}5^2 = 97$. 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 for the BF pruning.

Query-oblivious BF pruning. The BF pruning is as follows.

- **On the User.** Given a query Q, User computes η encodings of distinct $T^{i}_{u,2}$s, $i \in$ {vii, ..., x} for each vertex u of Q, where η is a parameter used to i) ensure the query-oblivious checking, and ii) tune BF's false positive rate. If there are no more than η encodings for u, User takes 0s as the rest encodings. Otherwise, some encodings are not involved in the BF pruning that may allow some false positives that some spurious balls cannot be detected. (It is obvious that the kind of false positives do not affect the correctness of the pruning since all non-spurious need to be evaluated by LGPQ.) Then, User encrypts these encodings and sends them into SGXs' enclaves on Players by establishing secure channels.
- **On the Player outside the enclave.** Given a ball B with the center w, Player i) computes the encodings of $T^{i}_{w,2}$, $i \in$ {vii, ..., x}, ii) constructs a BF for B by using these encodings with an encoding 0, and iii) transmits the BF into the enclave.
- **On the Player inside the enclave.** Inside the enclave, the encodings of Q are decrypted when received. After B's BF has been transmitted into the enclave, for each vertex u of Q having the same label as the label of w, B's BF is used to test whether u's η encodings exist in the BF. The tested results can be aggregated (e.g. added) into an integer in a query-oblivious manner. The aggregated integer is encrypted as the c_{sgx} for B that indicates whether B is spurious.

Analysis. We present the privacy result of the BF pruning in Sec. 5. Next, the number of hash functions in BF that minimizes its false positive rate 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,

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

where p is the false positive rate of BF. With Eqa. 1, we can tune p to balance the data transmission cost and the BF's pruning power.

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 [17] and paths [57] 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. 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 $\big[L_G(v_1), ..., L_G(v_{h-1}), [L_G(v_h), L_G(v_{h+1})]\big]$, $v_i \in V_G$, which satisfies the following: i) $(v_i, v_{i+1}) \in E_G$, $1 \leq i \leq h{-}2$, 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

[4] For any two label nodes of a h-label binary tree T that i) they 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.

Table 2: The 3-twiglet table $\mathcal{T}(u_1)$ of vertex u_1 of the graph shown in Fig. 7, where $\Sigma = \{A, B, C, D\}$ and $L(u_1) = A$

3-twiglet ts in $\mathcal{T}(u_1)$	ciphertext c_ts	plaintext	meaning
$[A, B, C]$	$g^x rq$	0	exists
$[A, B, D]$	$g^x rq$	0	exists
$[A, B, [C, D]]$	$g^x rq$	0	exists
$[A, C, B]$	$g^x r$	$\mathbb{Z}^+$	not exists
$[A, C, D]$	$g^x r$	$\mathbb{Z}^+$	not exists
$[A, C, [B, D]]$	$g^x r$	$\mathbb{Z}^+$	not exists
$[A, D, B]$	$g^x r$	$\mathbb{Z}^+$	not exists
$[A, D, C]$	$g^x r$	$\mathbb{Z}^+$	not exists
$[A, D, [B, C]]$	$g^x r$	$\mathbb{Z}^+$	not exists

Algorithm 5: Twiglet Pruning Algorithm `TwigletPrune`

```
Input  :The length $h$, the encrypted $h$-twiglet table $\mathcal{T}$, and a ball $B$ with center $w$
Output :The ciphertext $c_{phe}$ of $B$
1  Procedure TwigletPrune( $\mathcal{T}$, $M_B$ ):
2  $r \leftarrow 0$;
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      $r' \leftarrow 1$;
6      foreach h-twiglet $t$ in $\mathcal{T}(u)$ do
7          if $t \in R$ then
8              $r' \leftarrow r' \cdot c_1$;              // aggregate ciphertext of 1
9          else
10             $r' \leftarrow r' \cdot c_t$;  // if violation, $u$ has $t$ but $w$ doesn't
11     $r \leftarrow r + r'$;
12 return $r$;
```

t starting from v_1. Fig. 7 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 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.*

The proof of Prop. 4 is presented in Appendix A.2 of [1]. For each vertex of Q having the same label as the ball center's label, we check whether Prop. 4. If the ball center 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 the** 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 vertex u_1 in Fig. 7 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 a number in $\mathbb{Z}^+$ 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. 3.
- **On the** Players**.** 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 start 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 in 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 is just used to ensure the consistency of the power of the private key for CGBE's decryption on the User. If t does not exist in B (Line 9), whether w can be matched to u 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 decrypted, w cannot be matched to any vertices of Q that B is spurious. Line 12 returns r as the c_{phe} of B.

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

4.3 Secure Retrieval of Balls

In this subsection, we i) present a secure retrieval scheme that enables User to early receive from Dealer the ciphertext results of non-spurious balls, and then ii) present our proposed scheme to generate the ball identifier sequences used to achieve the secure retrieval.

Overview of the secure retrieval scheme. Given a $PM = (c_{sgx}, c_{phe})$ of ball B obtained by the BF pruning and twiglet pruning, if the plaintext of either c_{sgx} or c_{phe} indicates that B is spurious, B is called *negative*. Otherwise, B is called *positive*. Then, we elaborate Fig. 3 further. After ③ receiving the encrypted PMs (Sec.s 4.1-4.2) from Players, User decrypts them and ④ sends them with the set S of the ball identifiers (BIds) of all candidate balls to Dealer. Dealer ⑤ generates for Player$_i$ ($1 \le i \le k$, where k is the number of Players) a BId sequence $\mathcal{S}_i$ consisted of a part of BIds in S by *putting the* BId*s of positives 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 (Sec.s 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. Once Dealer has received the ciphertext results of all positives, Dealer ⑧ sends them to User for decryption. This enables User to 1) find the balls containing matching subgraphs among positives early, ii) ⑨ retreive the encrypted data of such balls from Dealer early, and iii) ⑨ 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 query-oblivious BId sequences that each Player is *unaware* of the time when *all* its positives, have been evaluated.

Random sequence generation (RSG)**.** We first present a baseline BId sequence generation scheme called *random sequence generation*. Foremost, we present its input, namely a BId set. A BId set, denoted by S, is a set of the BIds of candidate balls sent from User to Dealer. Given k Players, RSG i) partitions S into k subsets (S_i, $1 \le i \le k$) of the same size by assigning random $|S|/k$ BIds in S to each subset S_i,[5] where $S = \bigcup_{1 \le i \le k} S_i$, and then ii) orders the BIds in S_i with a random sequence to obtain the sequence $\mathcal{S}_i$ for Player$_i$, $1 \le i \le k$.

[5] For presentation simplicity, we assume $|S|$ is divisible by k.

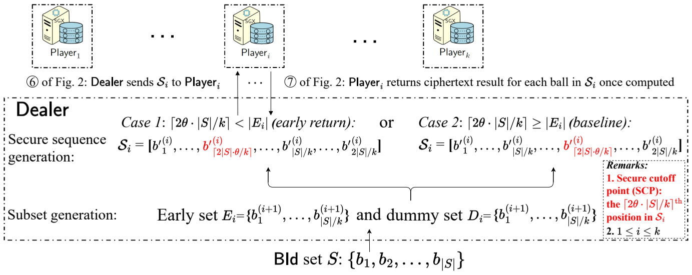

SGX
Player$_1$
Player$_i$
Player$_k$
⑥ of Fig. 2: Dealer sends $\mathcal{S}_i$ to Player$_i$
⑦ of Fig. 2: Player$_i$ returns ciphertext result for each ball in $\mathcal{S}_i$ once computed
Dealer
Case 1: $\lceil 2\theta \cdot |S|/k \rceil < |E_i|$ (early return):
or
Case 2: $\lceil 2\theta \cdot |S|/k \rceil \geq |E_i|$ (baseline):
Secure sequence generation:
$\mathcal{S}_i = [b'^{(i)}_1, \ldots, b'^{(i)}_{\lceil 2|S|\cdot\theta/k \rceil}, \ldots, b'^{(i)}_{|S|/k}, \ldots, b'^{(i)}_{2|S|/k}]$
$\mathcal{S}_i = [b'^{(i)}_1, \ldots, b'^{(i)}_{|S|/k}, \ldots, b'^{(i)}_{\lceil 2|S|\cdot\theta/k \rceil}, \ldots, b'^{(i)}_{2|S|/k}]$
Remarks:
1. Secure cutoff point (SCP): the $\lceil 2\theta \cdot |S|/k \rceil^{\text{th}}$ position in $\mathcal{S}_i$
2. $1 \leq i \leq k$
Subset generation:
Early set $E_i = \{b^{(i+1)}_1, \ldots, b^{(i+1)}_{|S|/k}\}$ and dummy set $D_i = \{b^{(i+1)}_1, \ldots, b^{(i+1)}_{|S|/k}\}$
BId set S: $\{b_1, b_2, \ldots, b_{|S|}\}$

Figure 8: Illustration of SSG

RSG allows Players to simultaneously process balls, by the algorithms in Sec.s 3.1-3.2. To achieve an *early* and *query-oblivious* return of ciphertexts of all positives, we further propose a sequence generation scheme called *secure sequence generation* (SSG). SSG is illustrated with Fig. 8.

Secure sequence generation (SSG)**.** Given a BId set S and k Players, SSG i) generates a BId set S_i for Player$_i$ ($1 \leq i \leq k$) 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 $|S| \cdot \theta$ ($0 \leq \theta \leq 1$) BIds of positives in S. Note that θ can be derived by the decrypted *PM*s and is *unknown* to Players. SSG partitions S into k early sets (E_i, $1 \leq i \leq k$) 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 ($1 \leq i \leq k$), for each 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) $\forall i, j \in [1, k]$ ($i \neq j$), $D_i \cap D_j = \emptyset$, and iii) $D_1 \cup \ldots \cup D_k = S$. Note that SSG can simply generate $D_i = E_{(i+1) \bmod k}$, $1 \leq i \leq k$ when $k \geq 2$.

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

The ordering step. The length of sequence $\mathcal{S}_i$ ($1 \leq i \leq k$) to be generated by SSG for 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 Player$_i$ when Player$_i$ finishes the evaluation of the ball located on SCP. The ordering has two cases.

- **Case 1** ($\theta < 1/2$)**.** As Fig. 8 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 Player$_i$ ($1 \leq i \leq k$), 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, $\mathcal{S}_i = \mathcal{E}_i || \mathcal{D}_i$.
- **Case 2** ($\theta \geq 1/2$)**.** SCP resides in the rear half part of $\mathcal{S}_i$ since $\lceil 2\theta \cdot |S|/k \rceil \geq |S|/k \rceil = |E_i|$. In this case, SCP cannot lead to an early return of positives' ciphertext results to User, and hence, SSG simply orders the BIds in E_i (*resp.* D_i) to obtain the sequence $\mathcal{E}_i$ (*resp.* $\mathcal{D}_i$) by using RSG. The value of θ depends on the pruning power of our proposed pruning techniques in Sec.s 4.1-4.2. As the experimental results show in Sec. 6, this case is not popular.

We remark that θ and SCP are not known to Players and Players have no way to identify the case and therefore the positives.

Example 3. *Given $k = 3$ Players and a BId set $S = \{b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9\}$ where b_5, b_6 and b_7 are BIds of positives,* SSG *generates subsets $E_1 = \{b_8, b_2, b_5\} = D_2$, $E_2 = \{b_6, b_1, b_9\} = D_3$ and $E_3 = \{b_7, b_3, b_4\} = D_1$. Since $\theta = 3/9$,* SCP *is the 2^{th} position and* 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 $\mathcal{S}_3 = [b_3, b_7] \,||\, [b_2, b_5, b_8, b_4]$.*

With the sequences generated by SSG, User can receive the ciphertext results of all positives before the end of the whole query processing of LGPQ on Players and can retrieve the encrypted balls from Dealer to decrypt and to compute the matching subgraphs.

5 PRIVACY ANALYSIS

Due to space restrictions, we present the main ideas of the privacy analysis, but present the detailed proofs in Appendix B of [1]. The privacy target in Sec. 2.3 contains i) the query privacy, and ii) the access pattern privacy. We first present 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.*

Prop. 5 is due to the security of the encryption schemes and the assumption that Players and Dealer do not collude. Alg. 1-3 in the LGPQ framework are query-oblivious that preserves the access pattern privacy, while the ciphertext results of Alg. 1-3 preserve the query privacy by CGBE [17]. Hence, we derive Prop. 6 as follows.

Proposition 6. *The privacy target is preserved by the* LGPQ *framework from* SP *against the attack model.*

Regarding the BF pruning, $O(\eta)$ encodings are assessed inside the SGX's enclave in a query-oblivious way. Hence, we have Prop. 7.

Proposition 7. *The privacy target is preserved by the* BF *pruning from* SP *against the attack model.*

Next, we analyze the twiglet pruning. Let $\mathcal{G}(r)$ be a function that returns 1 if SP can compute the plaintext of the output ciphertext r of Alg. 5 and returns 0, otherwise. Then, the probability that $\mathcal{G}(r) = 1$ is negligible as stated in Prop. 8.

Proposition 8. *After running* `TwigletPrune`*, $Pr[\mathcal{G}(r) = 1] \leq 1/2 + \epsilon$, where ϵ is negligible.*

With Prop. 8, `TwigletPrune` preserves the query privacy. Since `TwigletPrune` is also oblivious, we have Prop. 9.

Proposition 9. *The privacy target is preserved by the twiglet pruning from* SP *against the attack model.*

For SSG, Dealer knows only the *PM*s without the ball data, while Players knows only the ball data and ball identifier sequences without the *PM* of each ball. Hence, we have Prop. 10.

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

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

Theorem 1. LGPQ* *preserves the privacy target from* SP *against the attack model.*

Table 3: Statistics of three real-world datasets

Graph G	$\lvert V_G \rvert$	$\lvert E_G \rvert$	$\lvert \Sigma_G^H \rvert$	$\lvert \Sigma_G^S \rvert$
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 Bs for 10 random queries under the default setting

Graph	Avg. no. of balls per query	avg. $\lvert V_B \rvert$	stddev. of $\lvert V_B \rvert$	avg. $\lvert E_B \rvert$	stddev. of $\lvert E_B \rvert$	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 LGPQ* and the effectiveness of the proposed pruning techniques. We first present the experimental settings in Sec. 6.1. Then, we report the overall performance of the proposed BF pruning, twiglet pruning, and the LGPQ* framework in Sec. 6.2. In Sec. 6.3, we present a detailed evaluation by varying crucial parameters.

6.1 Experimental Settings

Platform. We implemented the prototype of LGPQ* in C++ using a machine with an Intel Core i7-7567U 3.5GHz CPU and 32GB RAM running Ubuntu 20.04.4 LTS to test the performance on both User and SP (including Players equipped with SGX 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* [34], which are also used in [18, 53, 57, 61]. The vertices of these datasets do not have labels. Similar to existing works [18, 40, 57], we generated a random label for each vertex to evaluate LGPQ*'s performance. We focused on *hom* and *ssim* queries, but omitted *sub-iso* queries, as the performance is similar to *hom*'s. 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 [18, 57]. Regarding the query sets, we used the same query generator *QGen* [57]. We generated 10 random queries for each experiment. Taking a query size $|V_Q|$, a diameter d_Q and a data 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. We followed the related work [57]. The encoding q and random number r for CGBE were both of 32 bits. The public value was of 4096 bits.
- *Query*. We varied the query sizes $|V_Q| = 6$, 8, or 10 and denoted the queries as $Q_{|V_Q|}$. 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* (TreeBF). For the paramter η used to ensure TreeBF'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 BF by Eqa. 1. Compared with passing $25K$ bits of data between the enclave and the application of SGX, TreeBF's online contruction of BFs for ball centers may take more time, especially for the enumeration of subtrees of topology x (Lines 14-15 of Alg. 4). Therefore, we used a threshold t for TreeBF to balance the efficiency and pruning performance. Specifically, for the ball center, if there exist more than t neighbors whose $\mathcal{L}$ size is greater than 3, TreeBF simply marked the ball as positive. We varied $t = 5$, 15, or 25 for TreeBF and denoted the corresponding algorithm as TreeBF$_t$. The default value of t was 15.
- *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 Twiglet$_h$. The default value of h was 3.
- *Path-based pruning* (Path) [57]. We used the existing path-based pruning technique as the baseline for comparison. We denote Path using a h value as Path$_h$.
- *Number of* Player*s*. The number of Player servers is denoted by k, where $k = 4$, 8, or 16. The default value of k was 4.

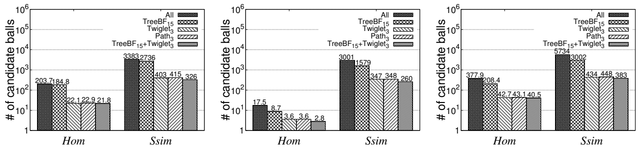
6.2 Overall Performance

We first investigate the overall performance of some computations on the User side, TreeBF & Twiglet, and LGPQ*.

EXP-1. Performance on the User **side**. As Fig. 3 shows, User i) generates the encrypted messages for queries, ii) decrypts the pruning messages for candidate balls, and iii) decrypts the ciphertext results sent by Dealer.

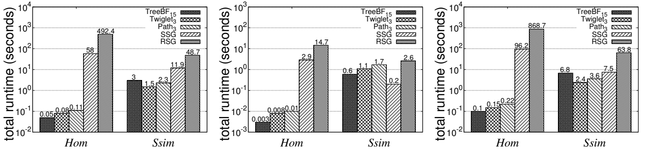
- *Preprocessing*. Given a query Q, User generated the encrypted encoding $M_{Q_e}^E$ of Q's adjacency matrix, a twiglet table $\mathcal{T}$ and the encrypted encodings of 2-label binary trees for each vertex of Q. The total preprocessing time was always less than $0.25s$, including *AES256*'s encryption for the encodings of 2-label binary trees to be sent to the enclave and CGBE's encryption for i) the encoding M_{Q_e}, ii) the $\mathcal{T}$s, and iii) 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 TreeBF (Sec. 4.1.2), Twiglet (Alg. 5) and LGPQ (Alg. 3). The total decryption time under our experiments was always less than $0.5s$ 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 Player$_i$, where $1 \leq i \leq k$ on the SP side, Alg.s 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 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 $10MB$. The total size of messages sent from Players to User was no larger than $20MB$ only.

EXP-2. Overall runtimes of TreeBF$_{15}$ **and** Twiglet$_3$. We investigate the efficiency of TreeBF and Twiglet under the default setting. Due to space restrictions, we report only the results when the figures and detailed analysis are presented in Appendix C of [1]. TreeBF$_{15}$ took hundreds of milliseconds for *hom* queries and few seconds for *ssim* queries, respectively. Not surprisingly, the runtimes of TreeBF$_{15}$ on all datasets increase as the ball sizes increase. The



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

Figure 9: Average number of candidate balls for algorithms



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

Figure 10: Average total runtimes for algorithms

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.

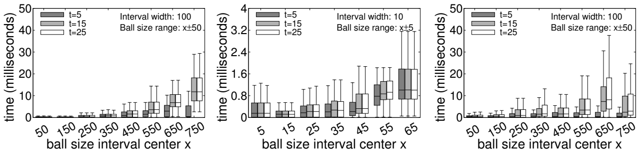
EXP-3. Overall performance of $\textsf{LGPQ}^*$**.** The following average results were from 10 random queries under the default setting. In Fig. 9, *All* denotes the average number of candidate balls, which can be either positive or negative. Fig. 9 reports the average number of candidate balls after pruning negatives by each method. It can be observed that, although $\texttt{TreeBF}_{15}$ pruned fewer negatives than $\texttt{Twiglet}_3$ or $\texttt{Path}_3$, $\texttt{TreeBF}_{15}$ helped $\texttt{Twiglet}_3$ to further prune negatives, especially for *ssim* queries.

In Fig. 10, we denote Steps ⑤-⑦ of Fig. 3 of $\textsf{LGPQ}^*$ by using SSG (*resp.* RSG) as SSG (*resp.* RSG), and denote the times for Dealer to obtain the ciphertext results of positives as their runtimes. Fig. 10 shows the average total runtimes of (i) $\texttt{TreeBF}_{15}$, $\texttt{Twiglet}_3$, and $\texttt{Path}_3$, and (ii) SSG and RSG for both the *hom* and *ssim* queries.[6] First, it can be observed that the runtimes of $\texttt{TreeBF}_{15}$, $\texttt{Twiglet}_3$, and $\texttt{Path}_3$ are very small. We can see that the runtimes of the pruning methods ($\texttt{TreeBF}_{15}$, $\texttt{Twiglet}_3$, and $\texttt{Path}_3$) 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 illustrate the effect of Sec. 4.3 for User to early start.[7] The runtime of SSG is often one order of magnitude smaller than the runtime of RSG.

6.3 Effects of Varying Settings

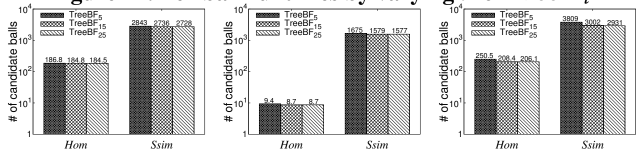
To investigate the performance of the algorithms, we ran them by varying a parameter at a time while other parameters are kept to their default values. It is known that the numbers of balls to be evaluated vary with queries. Hence, for ease of presentation, we used boxplots to present the runtimes.[8] The following figures do not display the outliers (fewer than 1%).

EXP-1. Varying t **for** $\texttt{TreeBF}_t$. Fig. 11 shows that $\texttt{TreeBF}_t$'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 us of the ball center that $|\mathcal{L}(u)| \geq 3$ and hence decreasing t's value can hardly reduce the runtime of $\texttt{TreeBF}_t$ when $t \geq 15$.



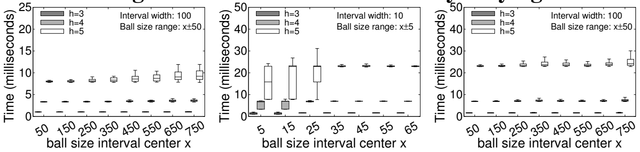
(a) ***Slashdot*** **($|\Sigma_G|$=100)** **(b)** ***DBLP*** **($|\Sigma_G|$=150)** **(c)** ***Twitter*** **($|\Sigma_G|$=100)**

Figure 11: Per-ball runtimes by varying t **for** $\texttt{TreeBF}_t$



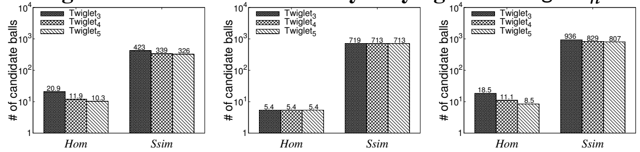
(a) ***Slashdot*** **(b)** ***DBLP*** **(c)** ***Twitter***

Figure 12: Average number of candidate balls by varying t **for** $\texttt{TreeBF}_t$



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

Figure 13: Per-ball runtimes by varying h **for** $\texttt{Twiglet}_h$



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

Figure 14: Average number of candidate balls by varying h **for** $\texttt{Twiglet}_h$

From Fig. 12, we can see that $\texttt{TreeBF}_t$ with a larger t can prune more negatives and $t = 15$ reaches a balance between the efficiency and pruning power of $\texttt{TreeBF}_t$.

EXP-2. Varying h **for** $\texttt{Twiglet}_h$. As Fig. 13 shows, the runtimes of $\texttt{Twiglet}_h$ increase as h increases since a larger h requires more time for *DFS* in $\texttt{Twiglet}_h$ to obtain the i-twiglets, $3 \leq i \leq h$. In Fig. 13(b), $\texttt{Twiglet}_h$'s runtimes on *DBLP* vary a lot for balls of small sizes. This is because the number of twiglets enumerated for small balls simply varies a lot.

Fig. 14 shows that $\texttt{Twiglet}_h$ 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, the default value 3 of h has a good balance between the efficiency and the pruning power.

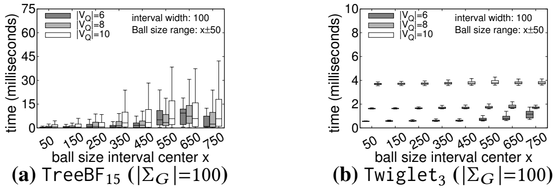
EXP-3. Varying $|V_Q|$. Fig. 15 shows the runtimes of $\texttt{TreeBF}_{15}$, $\texttt{Twiglet}_3$ on *Twitter* for queries Q_6, Q_8 and Q_{10}. It can be observed from Fig. 15(a) that $\texttt{TreeBF}_{15}$'s runtime increases slightly as $|V_Q|$ increases. This is because a larger $|V_Q|$ leads to a larger $|\Sigma_Q|$, which may increase the number of distinct labels of neighbors of each vertex of balls. From Fig. 15(b), we can see that the runtimes of $\texttt{Twiglet}_3$ increase as $|V_Q|$ increases since both a larger $|V_Q|$ and a larger $|\Sigma_Q|$ cost Alg. 5 more time.

EXP-4. Varying k **for** Players. We use SSG and RSG in Fig. 10 to present $\textsf{LGPQ}^*$'s performance. As shown in Sec. 4.3, the runtime of SSG is related to the value of θ, which depends on the pruning power of $\texttt{TreeBF}$ and $\texttt{Twiglet}$. Note that θ is also defined as the *predicted positive condition rate* (*PPCR*), *i.e.*, $(TP + FP)$ / $(TP +$

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

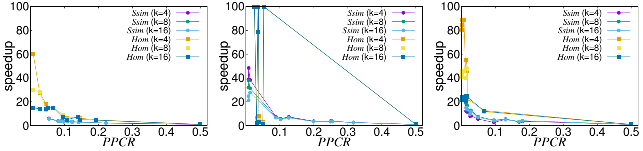
[7] We omitted the evaluation on User to compute the detailed subgraphs by using the state-of-the-art *LGPQ* matching algorithms [27, 28, 40] under the plaintext domain.

[8] 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.



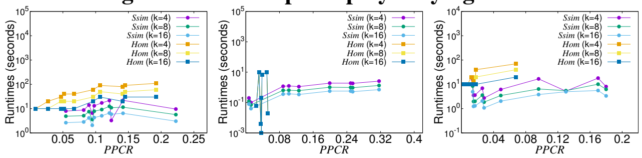
(a) TreeBF$_{15}$ ($|\Sigma_G|$=100) **(b)** Twiglet$_3$ ($|\Sigma_G|$=100)

Figure 15: Per-ball runtimes by varying $|V_Q|$ on *Twitter*



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

Figure 16: SSG**'s speedup by varying *PPCR***



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

Figure 17: Runtimes of SSG **by varying *PPCR***

$TN + FP + FN$), 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.*, SSG's speedup) as the y-axis and *PPCR* as the x-axis to show LGPQ*'s performance 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 are relatively small that increasing the number of Players cannot return the ciphertext results of positives to the user earlier. In Fig. 16(b), the speedup varies a lot for *hom* queries on *DBLP* due to the variety of SSG's runtime on few candidate balls, where the numbers of positives for most queries equal to 1.

Fig. 17 shows the runtimes of SSG for both *hom* and *ssim* queries when k varies. Similarly, we can see that SSG's runtime decreases when k increases. However, when *PPCR* is small, more Players cannot decrease SSG's runtime since 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.

7 RELATED WORK

Privacy preserving queries. There have been works on privacy preserving query processing [4, 14, 33, 53, 55, 62] in recent years. For graph pattern queries, three kinds of privacy models are studied. The difference between these models resides in the privacy target about the following structure(s).

- Both query and data graphs: Cao et al. [9] studied tree pattern queries on encrypted *XML* documents by predetermining the traversal order for each query. Cao et al. [10] proposed a *filtering and verification* method to solve the *sub-iso* query over encrypted graph-structured data in cloud computing. Fan et al. [17] also studied the *sub-iso* query semantic and proposed the *cyclic group based encryption scheme* (*CGBE*) to answer only the existence of matching subgraphs but does not consider ball/matching retrieval.
- Data graph only: By utilizing the *k-automorphic graph*, Chang et al. [11] and Huang et al. [25] studied the *sub-iso* query semantic, while Gao et al. [19] studied the *strong simulation* query [40]. In this model, users' queries are known by the service provider (SP).
- Query graph only: By using the *CGBE* encryption scheme, Fan et al. [18] answered *Yes/No* to the existence of matching subgraphs for the *sub-iso* query, while Xu. et al. [57] studies the problem of the *strong simulation* query only, which also sketches a trivial baseline for retrieving the matching subgraphs.

Cryptosystem. To compute the pruning messages securely, SP may adopt the inner product of encrypted messages from both the query and data graphs. To achieve this, the asymmetric scalar-product-preserving encryption scheme (*ASPE*) [54] was used in [17]. However, [60] has proved that *ASPE* is insecure under the known plaintext attack (*KPA*). Boneh-Goh-Nissim (*BGN*) [6] is a cryptosystem that supports unlimited number of addition operations and at most one multiplication to achieve a secure inner product computation between two encrypted vectors. However, the performance of the state-of-the-art *BGN* [23] is still impractical in our case. Thus, the inner product approach cannot yet be adopted. We proposed a twiglet pruning technique by applying *CGBE* [17] to aggregate the key ciphertexts since *CGBE* supports efficient homomorphic operations.

Secure query framework. Gentry et al. [20] described the first plausible construction for a fully homomorphic encryption (*FHE*) that supports both addition and multiplication on ciphertexts. Due to the known poor performance, *FHE* cannot be adopted. An oblivious *RAM* simulator (*ORAMs*), introduced by Goldreich et al. [21], is a compiler that transforms algorithms in a way that the resulting algorithms preserve the input-output behavior of the original algorithm but the distribution of memory access pattern of the transformed algorithm is independent to the memory access pattern of the original algorithm. However, *ORAMs* cannot be adopted since the query cannot be known to SP. *GraphSC* [42], *GraphSE*^{2} [31] and *PeGraph* [52] are frameworks for privacy preserving query on encrypted graph data, which also need the query structure for the matching process. The trusted execution environment (*TEE*) is a recent solution for databases [51, 56, 63]. The security is guaranteed by the hardware. This work is the first to use a *TEE* (*SGX*) for graph queries.

8 CONCLUSION

This paper proposes LGPQ* for the problem of privacy preserving *localized graph pattern query* (*LGPQ*) processing in a cloud computing paradigm. LGPQ* 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, LGPQ* 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, LGPQ* 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 are presented. The experimental results have shown the efficiency of LGPQ*. As for future work, we plan to apply LGPQ* to other *LGPQ*s, such as *p-homomorphism* query [47] and *conditional graph pattern* (*CGP*) query [16].

REFERENCES

[1] Anonymity. 2022. https://anonymous.4open.science/r/LGPQ-0129.

[2] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'keeffe, Mark L Stillwell, et al. 2016. {SCONE}: Secure linux containers with intel {SGX}. In *OSDI*.

[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 *FAST*.

[4] Johes Bater, Yongjoo Park, Xi He, Xiao Wang, and Jennie Rogers. 2020. Saqe: practical privacy-preserving approximate query processing for data federations. *PVLDB* (2020).

[5] Dan Bogdanov, Sven Laur, and Jan Willemson. 2008. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*.

[6] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. 2005. Evaluating 2-DNF formulas on ciphertexts. In *TCC*.

[7] Angela Bonifati, Wim Martens, and Thomas Timm. 2020. An analytical study of large SPARQL query logs. *VLDBJ* (2020).

[8] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure:{SGX} cache attacks are practical. In *WOOT*.

[9] Jianneng Cao, Fang-Yu Rao, Mehmet Kuzu, Elisa Bertino, and Murat Kantarcioglu. 2013. Efficient tree pattern queries on encrypted xml documents. In *EDBT/ICDT*.

[10] Ning Cao, Zhenyu Yang, Cong Wang, Kui Ren, and Wenjing Lou. 2011. Privacy-preserving query over encrypted graph-structured data in cloud computing. In *ICDCS*.

[11] Zhao Chang, Lei Zou, and Feifei Li. 2016. Privacy preserving subgraph matching on large graphs in cloud. In *SIGMOD*.

[12] Stephen A Cook. 1971. The complexity of theorem-proving procedures. In *STOC*.

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

[14] 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 *ICDE*.

[15] Yousef Elmehdwi, Bharath K Samanthula, and Wei Jiang. 2014. Secure k-nearest neighbor query over encrypted data in outsourced environments. In *ICDE*.

[16] Grace Fan, Wenfei Fan, Yuanhao Li, Ping Lu, Chao Tian, and Jingren Zhou. 2020. Extending graph patterns with conditions. In *SIGMOD*.

[17] Zhe Fan, Byron Choi, Qian Chen, Jianliang Xu, Haibo Hu, and Sourav S. Bhowmick. 2015. Structure-preserving subgraph query services. *TKDE* (2015).

[18] Zhe Fan, Byron Choi, Jianliang Xu, and Sourav S. Bhowmick. 2015. Asymmetric structure-preserving subgraph queries for large graphs. In *ICDE*.

[19] 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 *DASFAA*.

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

[21] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *JACM* (1996).

[22] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache attacks on Intel SGX. In *EUROSEC*.

[23] Vincent Herbert, Bhaskar Biswas, and Caroline Fontaine. 2019. Design and implementation of low-depth pairing-based homomorphic encryption scheme. *JCEN* (2019).

[24] Haibo Hu, Jianliang Xu, Qian Chen, and Ziwei Yang. 2012. Authenticating location-based services without compromising location privacy. In *SIGMOD*.

[25] Kai Huang, Haibo Hu, Shuigeng Zhou, Jihong Guan, Qingqing Ye, and Xiaofang Zhou. 2021. Privacy and efficiency guaranteed social subgraph matching. *VLDBJ* (2021).

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

[27] 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 *SIGMOD*.

[28] 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. *VLDBJ* (2022).

[29] Hyeong-Il Kim, Hyeong-Jin Kim, and Jae-Woo Chang. 2019. A secure kNN query processing algorithm using homomorphic encryption on outsourced database. *DKE* (2019).

[30] Jinha Kim, Hyungyu Shin, Wook-Shin Han, Sungpack Hong, and Hassan Chafi. 2015. Taming subgraph isomorphism for RDF query processing. *PVLDB* (2015).

[31] 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 *AsiaCCS*.

[32] Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. 2012. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *PVLDB* (2012).

[33] 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 *ICDE*.

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

[35] 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 *SIGMOD*.

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

[37] An Liu, Kai Zhengy, Lu Liz, Guanfeng Liu, Lei Zhao, and Xiaofang Zhou. 2015. Efficient secure similarity computation on encrypted trajectory data. In *ICDE*.

[38] Jinfei Liu, Juncheng Yang, Li Xiong, and Jian Pei. 2017. Secure skyline queries on cloud platform. In *ICDE*.

[39] Rongxing Lu, Xiaodong Lin, Zhiguo Shi, and Jun Shao. 2014. PLAM: A privacy-preserving framework for local-area mobile social networks. In *INFOCOM*.

[40] Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai, and Tianyu Wo. 2014. Strong simulation: Capturing topology in graph pattern matching. *TODS* (2014).

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

[42] Kartik Nayak, Xiao Shaun Wang, Stratis Ioannidis, Udi Weinsberg, Nina Taft, and Elaine Shi. 2015. GraphSC: Parallel secure computation made easy. In *S&P*.

[43] Miles Ohlrich, Carl Ebeling, Eka Ginting, and Lisa Sather. 1993. Subgemini: Identifying subcircuits using a fast subgraph isomorphism algorithm. In *DAC*.

[44] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*.

[45] Claude E Shannon. 1949. Communication theory of secrecy systems. *Bell Syst. Tech. J.* (1949).

[46] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. 2016. Preventing page faults from telling your secrets. In *AsiaCCS*.

[47] 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 *SIGMOD*.

[48] Einat Sprinzak, Shmuel Sattath, and Hanah Margalit. 2003. How reliable are experimental protein–protein interaction data? *JMB* (2003).

[49] Yuanyuan Tian and Jignesh M. Patel. 2008. Tale: A tool for approximate large graph matching. In *ICDE*.

[50] Julian R Ullmann. 1976. An algorithm for subgraph isomorphism. *JACM* (1976).

[51] 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. *PVLDB* (2022).

[52] Songlei Wang, Yifeng Zheng, Xiaohua Jia, and Xun Yi. 2022. PeGraph: A system for privacy-preserving and efficient search over encrypted social graphs. *TIFS* (2022).

[53] Songlei Wang, Yifeng Zheng, Xiaohua Jia, and Xun Yi. 2022. Privacy-preserving analytics on decentralized social graphs: The case of eigendecomposition. *TKDE* (2022).

[54] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. 2009. Secure knn computation on encrypted databases. In *SIGMOD*.

[55] 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 *ICDE*.

[56] Cheng Xu, Ce Zhang, Jianliang Xu, and Jian Pei. 2021. SlimChain: scaling blockchain transactions through off-chain storage and parallel processing. *PVLDB* (2021).

[57] Lyu Xu, Jiaxin Jiang, Byron Choi, Jianliang Xu, and Sourav S Bhowmick. 2021. Privacy preserving strong simulation queries on large graphs. In *ICDE*.

[58] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *S&P*.

[59] Xifeng Yan, Philip S Yu, and Jiawei Han. 2004. Graph indexing: a frequent structure-based approach. In *SIGMOD*.

[60] Bin Yao, Feifei Li, and Xiaokui Xiao. 2013. Secure nearest neighbor revisited. In *ICDE*.

[61] 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* (2020).

[62] 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. *TIFS* (2022).

[63] Wenchao Zhou, Yifan Cai, Yanqing Peng, Sheng Wang, Ke Ma, and Feifei Li. 2021. Veridb: An sgx-based verifiable database. In *SIGMOD*.

[64] Lei Zou, Lei Chen, and M Tamer Özsu. 2009. Distance-join: Pattern match query in a large graph database. *PVLDB* (2009).

A DEFINITIONS AND PROOFS

A.1 Definitions of *Strong Simulation*

DEFINITION 4. *(**Strong simulation** (**ssim**)) Given a connected query graph Q and a data graph G, G is a* strong simulation *of Q, if G has a ball $B = G[v_s, d_Q]$, where $v_s \in V_G$, such that there exists a binary relation $S \subseteq V_Q \times V_B$, denoted by $\langle\cdot,\cdot\rangle$, satisfying the following conditions.*

(1) $\forall u \in V_Q$, $\exists v \in V_B$ such that $\langle u, v\rangle \in S$
(2) $\exists u \in V_Q$, such that $\langle u, v_s\rangle \in S$, and
(3) $\forall \langle u, v\rangle \in S$,
(a) $L_Q(u) = L_B(v)$, i.e., $P(u, v) = 1$,
(b) for all $u' \in V_Q$ where $M_Q(u, u') = 1$, there exists v' in V_B such that $M_B(v, v') = 1$ and $\langle u', v'\rangle \in S$, and
(c) for all $u'' \in V_Q$ where $M_Q(u'', u) = 1$, there exists v'' in V_B such that $M_B(v'', v) = 1$ and $\langle u'', v''\rangle \in S$. □

A.2 Proofs

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) $v \in G_s$.*

PROOF. W.l.o.g, assume vertex u in Q that satisfies $L_Q(u) = l$. Since G_s is a matching subgraph for Q under *hom* with match function $\mathcal{H}$, there exists vertex v in G_s that $\mathcal{H}(u) = v$ and hence $L_{G_s}(v) = l$. For any vertex v' in G_s, we have $\text{dis}(v, v') \leq d_{G_s} \leq d_Q$ that $V_{G_s} \subseteq V_{G[v,d_Q]}$. Since $G[v, d_Q]$ (*resp.* G_s) is the induced subgraph of G based on vertex set $V_{G[v,d_Q]}$ (*resp.* V_{G_s}), G_s is a subgraph of $G[v, d_Q]$ and v is the target vertex of Prop. 1. □

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) $L_G(w') = l$.*

PROOF. Given a query Q and a label l ($l \in \Sigma_Q$), assume that the subgraph B_s of ball $B = G[w, d_Q]$ satisfies i) B_s is a matching subgraph for Q under *hom*, and ii) $w \notin V_{B_s}$. W.l.o.g, assume vertex u of Q satisfies that $L_Q(u) = l$. Since B_s is a matching subgraph for Q under *hom* with match function $\mathcal{H}$, we have $\text{dis}\big(v, \mathcal{H}(u)\big) \leq d_{B_s} \leq d_Q$, $v \in V_{B_s}$. Therefore, ball $G[\mathcal{H}(u), d_Q]$ satisfies i) B_s is a subgraph of $G[\mathcal{H}(u), d_Q]$, ii) $\mathcal{H}(u) \in V_{B_s}$, and iii) $L_G\big(\mathcal{H}(u)\big) = l$. Prop. 2 holds. □

Query-obliviousness of Alg 1. Given a query Q and a ball B, Alg. 1 enumerates all CMMs of B by using only V_Q, Σ_Q and L_Q, where the pruning of redundant CMMs is independent of E_Q. Given any query Q' that i) $V_{Q'} = V_Q$, ii) $\Sigma_{Q'} = \Sigma_Q$, and iii) $L_{Q'} = L_Q$, Alg. 1 returns the same set *CM* for Q'. Therefore, Alg. 1 is query-oblivious that preserves the query privacy. □

Correctness of Alg. 2. Assume the CMM C represents a match function $\mathcal{H}_C$: $V_Q \rightarrow V_G$. According to CMM's definition (Sec. 3.1), H_C satisfies condition (1) of Def. 1. Hence, matching violations can only be found for unsatisfaction on condition (2) of Def. 1. That is, $\exists\ i, j \in [i, |V_Q|]$ that $M_Q(i, j) = 1$ and $M_p(i, j) = 0$ since $M_p = C \cdot M_G \cdot C^{\mathrm{T}}$ is the projected adjacency matrix of G by C. To detect this unsatisfaction, we consider only two cases where $M_p(i, j) = 0$.

1. $\forall i, j$ that $M_p(i, j) = 0$, $M_Q(i, j) = 0$.
2. $\exists i, j$ that $M_p(i, j) = 0$, $M_Q(i, j) = 1$.

For Case 1, assume $\mathcal{H}_C$ does not satisfy condition (2) of Def. 1. Then, there must exist vertices $u, v \in V_Q$ that $M_Q(u, v) = 1$ and $M_G(\mathcal{H}_C(u), \mathcal{H}_C(v)) = C(u) \cdot M_G \cdot C^{\mathrm{T}}(v) = M_p(u, v) = 0$, This is contradictory to Case 1 and hence $\mathcal{H}_C$ satisfies condition (2) of Def. 1 that $\mathcal{H}_C$ is a valid match function under *hom*. In this case, no encodings are aggregated into r.

For Case 2, $\mathcal{H}_C$ cannot satisfy condition (2) of Def. 1 and hence H_C is *not* a valid match function under *hom*. In this case, there exist i and j that i) $M_p(i, j) = 0$, and ii) $M_Q(i, j)$'s encoding $M_{Q_e}(i, j) = q$. Therefore, Line 6 aggregates factor q into r by multiplication.

In Alg. 2, if C represents a valid match function under *hom*, only Case 1 exists that Line 7 returns r with an initial value 1. Otherwise, Case 2 also exists that r aggregates a factor q by Line 6. □

Query-obliviousness of Alg. 2. Consider a graph G and any two distinct queries Q_1, Q_2 that i) $V_{Q_1} = V_{Q_2}$, ii) $\Sigma_{Q_1} = \Sigma_{Q_2}$, and iii) $L_{Q_1} = L_{Q_2}$. Given a CMM for matching V_{Q_1} and V_{Q_2} to V_G, Lines 3-6 of Alg. 2 aggregate the encodings of elements located on the same positions in M_{Q_1} and M_{Q_2}. Therefore, the access pattern of encodings M_{Q_e} in Alg. 2 is independent of the query's edge set, *i.e.*, Alg. 2 is query-oblivious that preserves the query privacy. □

PROPOSITION 3. *Consider a height h, a query Q, and a ball B with the center w. If there exists at least one $T^i_{v,h}$, $i \in \{\text{i}, ..., \text{x}\}$ 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, then w cannot be matched to v under* sub-iso, hom *and* ssim.

PROOF. Prop. 3 can be proved by contradiction. W.l.o.g, we take *hom* as an example when the cases of *sub-iso* and *ssim* can be proved in a similar way. Given a match function $\mathcal{H}$ of *hom* for matching V_Q to V_B. Assume that center w of B can be matched to vertex v of Q. If there exists at least one $T^i_{v,h}$, $i \in \{\text{i}, ..., \text{x}\}$ but there does not exist $T^i_{w,h}$ such that $T^i_{v,h}$ and $T^i_{w,h}$ are isomorphic, then there is a contradiction that we can project a $T^i_{w,h}$ isomorphic to $T^i_{v,h}$ by using the vertices of B in $\{\mathcal{H}(u) | u \in \mathcal{V}\}$, where $\mathcal{V} = \{u | u \in V_Q$ and u is projected to generate $T^i_{v,h}\}$. Therefore, w cannot be matched v. □

PROPOSITION 4. *Consider a query Q and a ball B with 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.*

PROOF. Prop. 4 can be proved by contradiction. W.l.o.g, we take *hom* as an example when the cases of *sub-iso* and *ssim* can be proved in a similar way. Given a match function $\mathcal{H}$ of *hom* for matching V_Q to V_B. Assume that center w of B can be matched to vertex u of Q. If there exists one h-twiglet $t = \big[L_Q(u), l_1, ..., [l_{h-2}, l_{h-1}]\big]$ in Q projected by vertices $\big[u, v_1, ..., [v_{h-2}, v_{h-1}]\big]$ but there does not exist t in B starting from w, then there is a contradiction that the h-twiglet in B projected by vertices $\big[\mathcal{H}(u), \mathcal{H}(v_1), ..., [\mathcal{H}(v_{h-2}), \mathcal{H}(v_{h-1})]\big]$ is identical to t. That is, t does not exist. Therefore, w cannot be matched u. □

B DETAILED PRIVACY ANALYSIS

In this section, we analyze the privacy of LGPQ*, including i) the privacy of encrypted messages, ii) the LGPQ framework, iii) the BF pruning and the query-oblivious twiglet pruning techniques, and iv) the ball retrieval scheme. As introduced in Sec. 2.3, we assume Both Dealer and Players are semi-honest [36], and *Player*s do not collude with each other [35]. Moreover, we assume the collude-resistant model that *Dealer* and *Player* do not collude [14, 29, 37, 38].

B.1 Privacy Analysis of the Encrypted Messages

We adopt the asymmetric encryption scheme CGBE [17] and *AES256* to encrypt the queries in LGPQ*. The encrypted data of query Q include i) the encrypted encodings $M^E_{Q_e}$ of M_{Q_e} and the h-twiglet tables $\mathcal{T}$s that are encrypted by CGBE, and ii) the encrypted encodings of 2-label binary trees of Q's vertices that are encrypted by *AES256*. Then, we have the following proposition.

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.*

PROOF. For the Player server, after receiving query Q from User, Players cannot break the ciphertexts in $M^E_{Q_e}$ and $\mathcal{T}$s since they are secure against *CPA* [17]. The encodings of 2-label binary trees of Q's vertices are encrypted by *AES256* and sent into Players' SGX's enclaves that Players cannot obtain the plaintexts of these encodings. Since Dealer and Player do not collude, Prop. 5 holds. □

B.2 Privacy Analysis of the LGPQ Framework

The LGPQ framework has three steps, namely *candidate enumeration*, *query verification* and *query matching*. The analysis of their access pattern privacy are as follows.

- **Candidate enumeration.** As analysis in Sec. 3.1 shows, in the candidate enumeration algorithm (Alg. 1), all CMMs of a ball B are enumerated by using only V_Q, Σ_Q and L_Q of query Q, where the pruning of redundant CMMs is independent of E_Q. Given any query Q' that i) $V_{Q'} = V_Q$, ii) $\Sigma_{Q'} = \Sigma_Q$, and iii) $L_{Q'} = L_Q$, Alg. 1 returns the same set CM for Q'. Therefore, Alg. 1 is query-oblivious.
- **Query verification.** As analysis in Sec. 3.2 shows, in the query verification algorithm (Alg. 2), the access pattern of encodings M_{Q_e} (or $M^E_{Q_e}$ after encrypted) is independent of query's edge set. Consider a graph G and any two distinct queries Q_1, Q_2 that i) $V_{Q_1} = V_{Q_2}$, ii) $\Sigma_{Q_1} = \Sigma_{Q_2}$, and iii) $L_{Q_1} = L_{Q_2}$. Given a CMM for matching V_{Q_1} and V_{Q_2} to V_G, Lines 3-6 of Alg. 2 aggregate the encodings of elements located on the same positions in M_{Q_1} and M_{Q_2}. Therefore, Alg. 2 is query-oblivious.
- **Query matching.** For the query matching step, User decrypts the result set R from Player and then retrieves the target encrypted balls from Dealer. Although Dealer knows the specific balls retrieved by User, Dealer cannot infer the edge information of the query from the encrypted ball data and hence the query matching step is query-oblivious under our system and security model.

With the analysis above, the access pattern privacy is preserved by the LGPQ framework. For the inputs in plaintext, remark that i) V_Q, Σ_Q, L_Q and the ball B have no relation to the query privacy, and ii) d_Q does not lead to the query privacy leakage [18]. The ciphertexts used in LGPQ on Players only are encrypted by CGBE and hence secure under *CPA* that the query privacy is preserved from Players. Since Dealer and Player do not collude, we have the following proposition.

PROPOSITION 6. *The privacy target is preserved by the* LGPQ *framework from* SP *against the attack model.*

B.3 Privacy Analysis of the Pruning Techniques

BF **pruning.** For the BF pruning in Sec. 4.1, we have the Prop. 7.

PROPOSITION 7. *The privacy target is preserved by the* BF *pruning from* SP *against the attack model.*

PROOF. On the User side, η (η is used to ensure the obliviousness and is set as 256 in our experiments) encodings of 2-label binary trees of topologies vii-x in the query Q is computed without privacy leakage. On the Player side, the BF of each candidate ball is constructed, which does not expose the query privacy since the BF construction process are independent to the query. The encodings of Q are encrypted and sent into SGX's enclave via a secure channel established between User and each Player's SGX while the BF of candidate balls are transmitted into SGX's enclave directly on the Player side. As shown in Sec. 4.1.2, η encodings of Q are tested by the BF in a query-oblivious manner inside the enclave, where the tested results are also aggregated into an integer in a query-oblivious manner. The integer is encrypted as c_{sgx} inside the enclave. Currently, SGX suffers from three types of leakages.

(1) The access pattern leakages through extensive observations and statistics in the untrusted memory access locations. Since the encodings of Q are tested by the *BF* in a query-oblivious manner, this kind of access pattern is preserved.

(2) The access pattern leakages inside the enclave in a manner of page fault attacks [58]. Such attacks can be mitigated by a software-based defense solution [46]. Hence, we do not consider this kind of access leakages.

(3) The access pattern leakages using cache attacks [8, 22]. As our experimental settings show, the value of parameter m for the BF pruning (by Eqa. 1) is $25K$ that the size of each BF is smaller than $4KB$ that can be addressed within one page. As the granularity of the attacks becomes finer, the attacks become harder to get valid information [26].

With the analysis above, the access pattern privacy is preserved by the BF pruning from Players. Moreover, Q's encodings are obliviously tested by *BF*s inside SGX's enclave that the query privacy is also preserved from Players. Since Dealer and Players do not collude, Prop. 7 holds. □

Twiglet pruning. Let $\mathcal{G}(\mathcal{T}, i)$ (*resp.* $\mathcal{G}(r)$) 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 analyze the possibilities of $\mathcal{G}(r) = 1$ and derive the following proposition.

PROPOSITION 8. *After running* `TwigletPrune`, $Pr[\mathcal{G}(r) = 1] \leq 1/2 + \epsilon$, *where ϵ is negligible.*

Proof. Given a query Q and a ball B with center w, let $V_w = \{u | u \in V_Q$ and $L_Q(u) = L_B(w)\}$ and $W_w = \{h$-twiglet $t | t \in \mathcal{T}$ and t starting from vertex u, $u \in V_w\}$. For any h-twiglet $t_i \in \mathcal{T}$, since *CGBE* is secure against *CPA* [17], we have $Pr[\mathcal{G}(\mathcal{T}, i) = 1] \leq 1/2 + \epsilon$. For any two distinct h−*twiglet* $t_i, t_j \in \mathcal{T}$, we can know from the encoding and encryption scheme in Sec. 3 that their ciphertexts c_{t_i} and c_{t_j} are independent. Therefore,

$$Pr[\mathcal{G}(r) = 1] = \prod_{t_i \in W_w} Pr[\mathcal{G}(\mathcal{T}, i) = 1] \leq \frac{1}{2} + \epsilon$$

Moreover, `TwigletPrune` (Alg. 5) is oblivious that Players cannot infer any information of c_{t_i} or r from the access pattern of Alg. 5. Since Dealer and Players do not collude, Prop. 8 holds. □

From Prop. 8, we can know that `TwigletPrune` preserves the query privacy from SP. With i) `TwigletPrune` is oblivious that preserves the access pattern privacy from Players, and ii) Dealer and Players do not collude, we derive Prop. 9.

Proposition 9. *The privacy target is preserved by the twiglet pruning from* SP *against the attack model.*

B.4 Privacy Analysis of the Ball Retrieval Scheme

For the SSG scheme in Sec. 4.3, we have Prop. 10.

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

Proof. Given k Players, let $\mathcal{K}(i, j, \mathcal{S}_i)$ be a function that returns 1 if Player$_i$ ($1 \leq i \leq k$) can determine whether the j^{th} ball in sequence $\mathcal{S}_i$ is negative and returns 0, otherwise. Then, we prove Prop. 10 as follows.

(1) **The access pattern privacy is preserved by** RSG **from** Players. We analyze the possibilities of $\mathcal{K}(i, j, \mathcal{S}_i) = 1$. Given a ball identifier set S with $|S| \cdot \theta$ positives, RSG randomly assigns the positives in S to sequences $\mathcal{S}_i$ for Player$_i$, $1 \leq i \leq k$. Assume $|S| \cdot \theta \cdot \beta_i$ positives are assigned to $\mathcal{S}_i$. Then, we have,

$$Pr[\mathcal{K}(i, j, \mathcal{S}_i) = 1] = \beta_i \quad (2)$$

According to the *Shannon's maxim* [45], Player$_i$ is assumed to know the sequence partition scheme without the value of θ and β_i, $1 \leq i \leq k$. Therefore, Player$_i$ cannot know whether a ball in $\mathcal{S}_i$ is negative and the access pattern privacy is preserved by RSG from Players.

(2) **The access pattern privacy is preserved by** SSG **from** Players. Let C_i denote that SSG generates sequences $\mathcal{S}_i$, $1 \leq i \leq k$ according to Case i (Sec. 4.3). Given a ball identifier set S with $|S| \cdot \theta$ positives, the value of θ depends on the query structure and is unknown to Players. Therefore, the probability P_1 (*resp.* P_2) that Dealer generates sequences according to Case 1 (*resp.* Case 2) of SSG is equal to 1/2. In the following, we analyze these two cases separately.

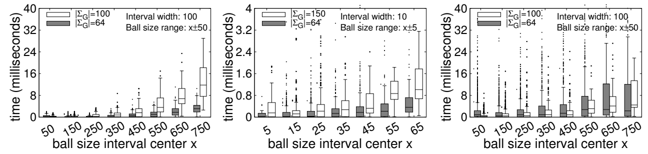
Case 1 ($\theta < 1/2$)**.** $\forall\, i \in [1, k]$ and $j \in [1, |S_i|]$,

– j **is not larger than the** SCP ($0 \leq j \leq \lceil 2|S| \cdot \theta / k \rceil$)**.** According to SSG, the numbers of positives and negatives before the SCP are the same. The balls before SCP are randomly permuted as shown in Fig. 8, Therefore, we have,

$$Pr[\mathcal{K}(i, j, \mathcal{S}_i) = 1 | C_1] = Pr[\mathcal{K}(i, j, \mathcal{S}_i) = 1] \cdot P_1 = \frac{1}{2} \cdot \frac{1}{2} < \frac{1}{2} + \epsilon \quad (3)$$

– j **is larger than the** SCP ($\lceil 2|S| \cdot \theta / k \rceil < j \leq 2|S|/k$)**.** For the balls in $\mathcal{S}_i$ after the SCP, there are $\alpha = \lceil |S| \cdot \theta / k \rceil$ positives and $2|S|/k - 3\alpha$ negatives that are permuted randomly. Therefore, we have,

$$Pr[\mathcal{K}(i, j, \mathcal{S}_i) = 1 | C_1] = Pr[\mathcal{K}(i, j, \mathcal{S}_i) = 1] \cdot P_1 = \frac{\frac{2|S|}{k} - 3\alpha}{\frac{2|S|}{k} - 2\alpha} \cdot \frac{1}{2} < \frac{1}{2} + \epsilon \quad (4)$$



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

Figure 18: Per-ball runtimes of `TreeBF`$_{15}$ **on three datasets**

Case 2 ($\theta \geq 1/2$)**.** SSG generates the early sequence $\mathcal{E}_i$ and dummy sequence $\mathcal{D}_i$ of sequence $\mathcal{S}_i$, $i \leq i \leq k$ by using RSG. According to Eqa. 2, we have,

$$P[\mathcal{K}(i, j, \mathcal{S}_i) = 1 | C_2] = P[\mathcal{K}(i, j, \mathcal{S}_i) = 1] \cdot P_2 = \beta_i \cdot \frac{1}{2} \leq \frac{1}{2} + \epsilon \quad (5)$$

Since Players do not collude with each other, Player$_i$ can learn nothing from the dummy sequence $\mathcal{D}_i$.

With Eqa. 3-5, it can be obtained that the access pattern privacy is preserved by SSG from Players.

With the analysis above, the access pattern privacy is preserved by SSG from Players. Since the sequences $\mathcal{S}_i$, $1 \leq i \leq k$ contain no information about the query privacy, the query privacy and hence the privacy target are preserved by SSG from Players. Moreover, Dealer cannot know any information about the query privacy from the encrypted ball data and the decrypted *PM*s sent by User. Although the access pattern for generating the sequences $\mathcal{S}_i$, $1 \leq i \leq k$ is not oblivious but related to the *PM*s, Dealer cannot infer any information about the query privacy from the access pattern with the encrypted ball data. Therefore, the access pattern privacy is preserved by SSG from Dealer and then the privacy target is preserved by SSG from Dealer. Since Dealer and Players do not collude, Prop. 10 holds. □

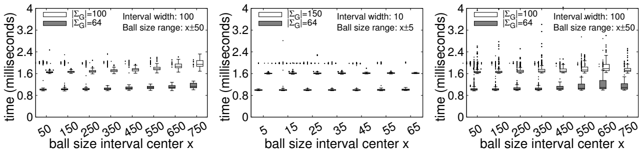
In ⑨ of Fig. 3, although Dealer knows the identifiers of encrypted balls retrieved by User, Dealer can know nothing about the query privacy from the encrypted ball data. Then, by putting Prop.s 5, 6, 7, 9 and 10 together, we have Theorem. 1.

Theorem 1. LGPQ* *preserves the privacy target from* SP *against the attack model.*

C OVERALL RUNTIMES OF `TreeBF` & `Twiglet`

2.1. Overall runtime of `TreeBF`$_{15}$**.** We investigate the BF pruning on the three datasets under the default setting. Fig. 18 shows `TreeBF`$_{15}$'s runtime for each ball, including the runtimes of online BF construction, data transmission into SGX's enclave, and oblivious checking on BF inside the enclave. We can see that the runtimes of `TreeBF`$_{15}$ on all datasets increase as the ball sizes increase. This is because the complexity of subtrees' enumeration (Alg. 4) depends on the maximum degree of vertices of the ball, which often increases when the ball sizes increase. According to the statistics of candidate balls in Table 4, `TreeBF`$_{15}$ may take hundreds of milliseconds for *hom* queries and few seconds for *ssim* queries.

2.2. Overall runtime of `Twiglet`$_3$**.** Fig. 19 shows the per-ball runtimes of `Twiglet`$_3$. It can be seen that the runtimes of `Twiglet`$_3$ on *Slashdot* and *Twitter* increase slightly as the ball sizes increase.



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

Figure 19: Per-ball runtimes of $\texttt{Twiglet}_3$ on three datasets

This is because $\texttt{Twiglet}_3$ involved *DFS* (Line 3 of Alg. 5) starting from the ball center to enumerate *h*-twiglets. For *DBLP*, $\texttt{Twiglet}_3$'s runtime is not sensitive to the ball size since *DBLP* is sparse that the ball sizes are not as large as the ones of *Slashdot* or *Twitter*, where the *DFS* visits few nodes. For the balls of graph G with a larger label set $|\Sigma_G|$, $\texttt{Twiglet}_3$ took more time to find a matching violation (Lines 6-11 of Alg. 5) due to more distinct labels of each vertex's neighbors. The relative performance of the techniques may vary with datasets. Compared to $\texttt{TreeBF}_{15}$, $\texttt{Twiglet}_3$ took less time for *LPGM* queries on *Slashdot* and *Twitter*, but took more time when querying on the sparse dataset *DBLP*.