Privacy Preserving Strong Simulation Queries on Large Graphs

Lyu Xu1  Jiaxin Jiang1  Byron Choi1
1Department of Computer Science
Hong Kong Baptist University, Hong Kong
{cslyuxu, jxjian, bchoi, xujl}@comp.hkbu.edu.hk

Jianliang Xu1  Sourav S Bhowmick2
2School of Computer Science and Engineering
Nanyang Technological University, Singapore
assourav@ntu.edu.sg

Abstract—This paper studies privacy preserving query services for strong simulation queries in the database outsourcing paradigm. In such a paradigm, clients send their queries to a third-party service provider (SP), who has the outsourced large graph data, and the SP computes the query answers. However, as SP may not always be trusted, the sensitive information of the clients’ queries, importantly, the query structures, should be protected. Moreover, graph pattern queries often have high complexities, whereas data graphs can be large. This paper adopts strong simulation as a practical query semantic for this paradigm. Under this semantic, queries are matched with a notion of balls, which are subgraphs related to the query diameter. We transform the core of the existing strong simulation algorithm using data-oblivious operations (ObsSSA) and propose its secure version. We show that the algorithm may encounter an overflow problem even partially homomorphic encryption (PHE) has been used. We then propose an efficient inexact algorithm EncSSA, which is secure under chosen plaintext attack (CPA). The results of privacy analysis are presented. We have conducted experiments on Twitter and Citeseer datasets, and the results show that EncSSA is both efficient and effective.

Index Terms—Graph queries, strong simulation, large graphs, data outsourcing, privacy preservation

I. INTRODUCTION

Graph pattern queries are increasingly found in emerging applications, including social networks, biology analysis, and communication networks [1]–[3]. However, some query formalisms are computationally costly, for example, the subgraph isomorphism query (sub-iso) is an NP-complete problem [4]. Subgraph homomorphism, used in the SPARQL query on the RDF data [5] with the same definition as sub-iso except that sub-iso has a bijective restriction on the matching, whereas homomorphism has an injective restriction [6]. These queries, however, can be too restricted for some applications. Graph simulation query [7] has been proposed, which can be computed in quadratic time but only support topology constraints on the children of each vertex. Similarly, dual simulation query [8], [9] preserves topological constraints on both the children and parents. Strong simulation query [9] has been then proposed to strike a balance between computation complexity and the capability to capture topological constraints.

In social search, strong simulation queries can be used to find an entity with specific types of connections or attributes. In recommender systems, such queries help individuals form collaboration networks with people having specific skills [9]–[11]. However, queries can be sensitive, as described below.

Example 1. (Privacy preserving query processing in data outsourcing) Take the social network Twitter as an example, where vertices are (virtual) entities, labels are nationalities, and edges are interactions. Fig. 1(a) is a user pattern of user Jack. Jack finds whether there exist similar labeled patterns in Twitter. In this scenario, outsourcing the searches to an SP has a number of advantages, including elasticity, high availability and cost savings, when compared to on-premise solutions. However, SP may not always be trusted. Jack does not want to expose the interactions of nationalities (flags) he searches (i.e., to protect his query structure) from the SP. Similar scenarios can be found, e.g., in collaboration networks.

Matching queries on the data graph at a large scale, especially with privacy preservation, is challenging. This paper adopts strong simulation as the query semantic because the queries are evaluated on the balls of the graph [9], which are the subgraphs being defined by their centers and radius (see details in Sec. II). Our experiments on benchmarked queries and real datasets show that balls contain several hundred vertices (Tab. IV). This opens up an opportunity for private query processing at the SP. Consider Example 1 and take “White House” as the ball center with a radius 2. Strong simulation searches the pattern in Fig. 1(a) within the ball of “White House” in Fig. 1(b), where privacy preserving computation is achievable in a ciphertext domain. In this paper, we investigate privacy preserving strong simulation query.

There have been a variety of research works on querying with privacy preservation in graph databases (see Sec. VII for details). To the best of our knowledge, strong simulation query that protects the structure information of the query in the data

Fig. 1: A strong simulation query on a simplified Twitter1 (Flags are labels. Text is for discussions but not queried.)

outsourcing paradigm has not been studied yet. This problem has the following two main technical challenges.

1) First, how to design an algorithm for strong simulation queries using data-oblivious computation [12] consisting of data access patterns that do not depend on the input?

2) Second, how to design algorithms that strike the balance between efficiency and privacy?

For the first challenge, our idea is to represent the query and each ball in the data graph using the adjacency matrix. Then, we replace the strong simulation algorithm with a series of matrix operations, which are data-oblivious. We propose an ObSSA algorithm based on the state-of-the-art [9] with only these operations. Given a query, ObSSA must carry out the same operations for each result matrix iteratively until a fixed point, where the number of iterations is bounded by the ball size.

To tackle the second challenge, we derive an encoding and adopt an encryption for the matrices. Fully homomorphic encryption (FHE) [13] cannot be adopted due to its known poor performance. Most of the existing partially homomorphic encryption (PHE) schemes (e.g., ElGamal, Paillier and Boneh-Goh-Missm) [14], [15] cannot support both additions and multiplications simultaneously more than once, which are needed by ObSSA. We adopt the encryption method, namely cyclic group based encryption scheme (CGBE), which is proposed by Fan et al. [16]. CGBE supports both the additions and multiplications simultaneously but does not allow the plaintext value of each ciphertext exceeding a public value. Using CGBE, we propose an encrypted version of ObSSA.

There is a further technical challenge from the encrypted ObSSA. Checking the definition of strong simulation requires multiple iterations that involve multiplications. Consequently, there will be an overflow in the computed ciphertext that the corresponding plaintext is larger than the given public value, e.g., a prime with 2048 bits used in [16].

To strike the balance between privacy and a practical algorithm, we propose an inexact algorithm EncSSA that consists of three efficient optimizations. (The inexact algorithm can have false positives but no false negatives.) The first idea is to exploit localized algorithms for 2-hop neighbors so that the encryption algorithm does not encounter ciphertext overflow. The second idea is to check strong simulation using a subset of possible paths derived from the query’s labels with a length $k$ larger than 2. The last idea is to check the set of labels of $h$-hop neighbors where $h$ is larger than $k$. These optimizations avoid excessive multiplications in the ciphertext domain, prune true negatives of the query results and achieve good efficiency.

Contributions. The contributions of this paper are as follows.

- We proposed the ObSSA algorithm to answer the strong simulation query under the plaintext setting, which transforms the key operations of the existing strong simulation algorithm with only data-oblivious mathematical operations.
- Based on the CGBE encryption scheme, we propose a practical and secure inexact algorithm EncSSA that comprises three privacy preserving pruning techniques.
- We present privacy analysis results of our proposed EncSSA.
- Our experiments verified the performance of EncSSA.

Organization. The preliminaries and the problem statement are introduced in Sec. II. We present the ObSSA algorithm in Sec. III and then the EncSSA algorithm in Sec. IV. The privacy analysis results are presented in Sec. V. Sec. VI presents the experimental results and Sec. VII discusses the recent related work. We conclude this paper with future work in Sec. VIII.

II. PRELIMINARIES AND PROBLEM FORMULATION

In this section, we first provide preliminaries related to strong simulation and system models for technical discussions. It then presents the problem statement of the paper.

A. Graph Data and Pattern Query Semantics

Graph. A graph is denoted by $G = (V_G, E_G, \Sigma_G, L_G)$, where $V_G$, $E_G$, $\Sigma_G$, $L_G$ are the sets of vertices, directed edges, labels and the function for matching the vertices with their labels. $(u,v)$ denotes the directed edge from vertex $u$ to $v$ and $L_G(u)$ denotes the label value of $u$. For a connected graph $G$, the distance between any 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 $G$ while the diameter of $G$, denoted by $d_G$, is the largest shortest distance between all pairs of vertices in $G$.

Ball. A ball $B = (V_B, E_B, \Sigma_B, L_B, u, r)$, denoted by $G[u,r]$, is a connected subgraph of $G$ which takes vertex $u$ as center, $r$ as radius, such that, (a) $V_B = \{v | v \in V_G, \text{dis}(u,v) \leq r\}$, and (b) $E_B$ has the edges that appear in $G$ over the same vertices in $V_B$ [9]. The size of ball, $|V_B|$, is restricted by $r$. The strong simulation query searches for matches within balls, as opposed to the whole data graph.

Adjacency matrix. The adjacency matrix of graph $G$, denoted by $M_G$, is a $|V_G| \times |V_G|$ matrix. $M_G$’s transport matrix is denoted by $M_G^T$. The $i^{th}$ row vector of $M_G$ is denoted by $M_G(i)$ and the $j^{th}$ element in $M_G(i)$ is denoted by $M_G(i,j)$.

We remove the subscript $G$ when it is clear from the context. To make the definition of query semantic consistent with this paper’s notations, we introduce the vertex mapping matrix and rewrite the definition of strong simulation query.

Vertex mapping matrix. The vertex mapping matrix from a query $Q$ to a graph $G$, denoted by $P$, is a $|V_Q| \times |V_G|$ matrix. $P(i,j) = 1$ if $L_Q(v_i) = L_G(v_j)$, $v_i \in V_Q$ and $v_j \in V_G$. Otherwise, $P(i,j) = 0$. The $i^{th}$ row vector of $P$ is denoted by $P(i)$.

The vertex mapping matrix uses a 1 (a possible mapping) to denote that a vertex of the query and a vertex of the graph have the same label, and 0 otherwise. For any vertex $u$ in $Q$, if vertex $v$ in $G$ satisfies $L_Q(u) = L_G(v)$, $v$ is denoted as a candidate match of $u$.

Strong simulation query [9]. We can then rewrite the semantics of strong simulation using matrices in Def. 1.

Definition 1. A graph $G$ is a strong simulation of a connected graph $Q$, denoted as $Q \triangleright_S G$, 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.
A strong simulation query \( Q = (V, E) \) on a graph \( G = (V_G, E_G, \Sigma_Q, L_Q) \) computes whether there exists an \( S \) such that \( Q \prec_S G \). Intuitively, \( S \) can be considered as a match from \( Q \) to \( B \) that preserves the topology of both children (3(b) of Def. 1) and parents (3(c) of Def. 1).

There have been related query semantics. Dual simulation [9] and graph simulation [7] do not restrict the matching in a ball. In addition, graph simulation only requires (3(b) of Def. 1). Matching queries of the whole graph in a ciphertext domain is computationally costly.

Example 2. Fig. 2(a) are examples for query \( Q \) and graph \( G \), whereas Fig. 2(b) is the vertex mapping matrix \( P \) from \( Q \) to ball \( G[B_1, 2] \). Fig. 2(c) shows a binary matrix from \( Q \) to \( G \) such that \( Q \prec_S G \).

B. Models and Problem Statement

System model. We follow the system model that is well received in the database outsourcing literature shown in Fig. 3. We assume that the service provider (SP) is equipped with powerful computing utilities such as a cluster. The SP hosts a query service for publicly known graph data. The SP receives encrypted queries from a client, evaluates them in the encrypted domain, and returns the encrypted answers to the client. The client generates the encrypted queries, submits them to the SP, and decrypts the answers (the IDs of the balls that contain matchings). A storage server is specially introduced. If the client needs to determine the query matchings, the relevant encrypted balls are sent from the server. We assume that the server and the SP do not collude. Otherwise, the SP can infer the queries from the balls requested by the client.

The sequence of whole query processing is as follows: \( \circ \rightarrow \bullet \rightarrow \odot \rightarrow \bigcirc \rightarrow \ast \rightarrow \odot \rightarrow \bigcirc \rightarrow \circ \). The key steps of privacy preserving query processing are \( \circ \), \( \odot \), and \( \odot \), i.e., the top half of Fig. 3. \( \circ \) The client generates the encrypted messages of query graph and submits them to the SP. \( \odot \) The SP evaluates a client’s encrypted query on the graph data \( D \) and \( \odot \) returns the encrypted results to the client. \( \bullet \) The client decrypts these results to obtain the final answer.

There are steps of \( \circ \) building and encrypting balls offline, storing them in a semi-honest storage server, and \( \odot \) generating the one-time permutation to be sent to the storage server for \( \odot \) the client to retrieve some encrypted balls from the storage server for \( \bigcirc \) verification of exact results. Results are standard techniques, and omitted.

Attack model. Assume the well known semi-honest adversary model [14], [17]–[19], where the attackers are honest but curious. For presentation simplicity, we consider the SP as the attackers. Also, we assume the attackers are the eavesdroppers and adopt the chosen plaintext attack (CPA) [14], i.e., the adversaries can choose arbitrary plaintexts to obtain their ciphertexts to gain information to reduce the security.

Privacy target. To facilitate technical discussions, we assume that the privacy target is to protect the structures of the query graph \( Q \) from the SP under the attack model defined above. The structural information of \( Q \) is considered the adjacency matrices of \( Q \). More specifically, the probability that the SP correctly determines the values of the adjacency matrix of the query graph \( Q \) is guaranteed to be lower than a threshold with reference to that of random guess.

Problem statement. Given a strong simulation query \( Q \) and a large graph \( G \), the system and attack model, we compute whether \( Q \prec_S G \) by matching each ball of \( G \) to \( Q \) while preserving the privacy target.

III. OBLIVIOUS STRONG SIMULATION ALGORITHM

In this section, we first present a transformed strong simulation algorithm TSSA based on the state-of-the-art and then propose an oblivious strong simulation algorithm ObSSA under the plaintext setting. We analyze the limitation of the encrypted version of ObSSA.

A. Transformed Strong Simulation Algorithm TSSA

The current state-of-the-art for determining strong simulation [9]. SSA, starts with (candidate) matching query vertices to data vertices that have the same label and iteratively prunes such matchings that have violations.

Violation. Consider a binary match \( \langle u, v \rangle \in Q \times V_G \) such that \( L_Q(u) = L_G(v) \). There is a violation for \( \langle u, v \rangle \) in ball \( B \) if the topological constraints derived from 3(b) or 3(c) of Def. 1 (in Sec. II-A) cannot be satisfied.

1. For all \( u' \in V_Q \) where \( M_Q(u, u') = 1 \), there exists \( v' \in V_G \) such that \( M_G(v, v') = 1 \) and \( \langle u', v' \rangle \in S \).
2. For all \( u'' \in V_Q \) where \( M_Q(u'', u') = 1 \), there exists \( v'' \in V_G \) such that \( M_G(v'', v) = 1 \) and \( \langle u'', v'' \rangle \in S \).
The violation for \((u, v)\) results in pruning of \((u, v)\). When no more violations can be detected, SSA found a valid strong simulation if (i) there is at least one match for each vertex in \(Q\) and (ii) \(B\)'s center is matched to at least one vertex in \(Q\).

Pseudo-code for TSSA (Alg. 1). The transformed strong simulation algorithm (TSSA) is presented in Alg. 1.² Alg. 1 returns a positive integer to show that \(Q \prec \prec G\), and it returns 0, otherwise. The details of Alg. 1 can be described as follows. For each vertex \(v \in G\) with \(L_Q(v) \in \Sigma_Q\) required by Constraint 2 of Def. 1, Alg. 1 retrieves the adjacency matrix \(M_B\) that represents the ball^2 \(B = G[v, d_Q]\) and constructs the vertex mapping matrix \(P\) from \(Q\) to \(B\) (Lines 3-4). For any vertex \(v_i\) in \(Q\) and any vertex \(v_j\) in \(B\) with the same label, \(P(i,j) = 1\), i.e., \(<i, j>\) is initially considered as a valid match. Fig. 2(b) shows an example for \(P\). \(P(3,1) = 1\), since \(L_Q(v_3) = L_G(v_1) = C\) as shown in Fig. 2(a).

In Line 5, ViolationPruning tests and prunes the invalid matches in \(P\). In particular, for any element \(P(i,j)\) in \(P\) (Line 12), if \(P(i,j) = 1\) and there is a violation of the constraint 3(b) (resp. 3(c)) because of the children (resp. parents) as Line 14 (resp. Line 17), then Line 15 (Line 18) revises \(P(i,j)\) to 0. Take Fig. 2(a) as an example. When matching \(v_1\) in \(Q\) to \(v_2\) in ball \(B_1\), \(P(1,4)\) in Fig. 2(b) equals 1 initially, and then, is revised to 0 since \(P(2)\cdot M_B^T(4) = 0\) for \(M_Q(2,1) = 1\) (Line 17), i.e., no parents of \(v_4\) in \(B_1\) can satisfy Constraint 3(c), and thus, \(v_4\) cannot be matched to \(v_1\).

The algorithm repeats the above steps (Line 12) until no more violations can be detected and removed (Line 10). Then, CheckSS in Line 6 checks (i) whether the revised matrix \(P\) still satisfies Constraint 1 of Def. 1, i.e., for the vertex \(v_i\) (the \(i\)th vertex) in \(Q\), there exists a vertex \(v_j\) (the \(j\)th vertex) in \(B\), such that \(P(i,j) = 1\) equals to 1. If there exists a row vector \(P(i)\) equals to the zero vector (Line 21), there is no vertex \(B\) that matches the \(i\)th vertex in \(Q\) and Line 22 returns 0, i.e., no match. CheckSS also checks (ii) whether the revised matrix \(P\) satisfies Constraint 2 of Def. 1, i.e., for the ball center \(v_i\), it can be matched to at least one vertex in \(Q\). If all the elements in the column for the ball center equal to zero (Line 23), there is no vertex in \(Q\) that can be matched to the ball center and Line 24 returns 0, i.e., no match. If Lines 21 and 23 do not hold, 1 is returned (Line 25). Following up with Example 2. (i) \(P\) shown in Fig. 2 does not have a row that contains all zeros and (ii) the column of the ball center (vertex 2) has a non-zero (\(P(2,3)\)). Hence, \(G\) has a strong simulation in \(B\) (shown in the dotted box of Fig. 2(a)). Finally, Line 7 returns non-zero, i.e., \(Q \prec \prec G\).

B. Oblivious Strong Simulation Algorithm (ObSSA)

In this subsection, we derive an oblivious algorithm called ObSSA from the transformed algorithm TSSA.

Violation detection. The core of TSSA is to remove the candidate matches that have the violations of Def. 1. Consider the vertex mapping matrix \(P\) from the query \(Q\) to the ball \(B\). The value of \(P(i,j)\) is modified using a series of addition and multiplication operations, called violation detector, as follows.

\[
P(i,j)' = \text{Child}(i,j) \cdot \text{Parent}(i,j), \quad \text{if} \quad P(i,j) \neq 0 \quad (1a)
\]

\[
\text{Child}(i,j) = \prod_{k=1}^{\mid V_Q \mid} M_Q(i,k) + \sum_{l=1}^{\mid V_B \mid} (M_Q(j,l) P(k,l)) \quad (1b)
\]

\[
\text{Parent}(i,j) = \prod_{k=1}^{\mid V_Q \mid} M_Q(k,j) + \sum_{l=1}^{\mid V_B \mid} (M_B(l,j) P(k,l)), \quad (1c)
\]
where \( M_Q(i,k) = 1 - M_Q(i,k) \) and \( P(i,j)' \) denotes the modified value of \( P(i,j) \) after running the violation detector.

**Pseudo-code for ObSSA (Alg. 2).** ObSSA is presented in Alg. 2. For each vertex \( v \) in \( G \) with \( L_G(v) \in \Sigma_Q \), Line 3 retrieves the adjacency matrix \( M_B \) of the ball \( B = G[v,d_Q] \) and Line 4 generates their vertex mapping matrix \( P \) from \( Q \) to \( B \). Then, ObViolationPruning() (Lines 9-17) updates the matrix \( P \) iteratively for \( |V_B| \cdot |V_Q| \) times using the violation detector (Eq. 1). Next, ObCheckSS() (Lines 18-27) computes the value prod and sum2 indicating that whether \( Q \prec_S B \). If prod or sum2 equals to 0, then \( B \subseteq G \) does not satisfy that \( Q \prec_S G \). Otherwise, \( Q \prec_S G \). Finally, Line 7 computes whether \( Q \prec_S G \) or not.

**Obliviousness of ObSSA.** Given queries of the same size and label set, ObSSA performs the same number and sequence of operations to evaluate each of them. Firstly, ObSSA traverses the same balls. For each ball and each query, we can note that the sizes of the vertex mapping matrices \( P \) used in Line 4 are the same and independent to the query structure. In addition, regardless of the query structures, (i) the number of iterations in Line 11 of ObViolationPruning() are the same, and (ii) Eq. 1 (Lines 14-15), Lines 21-26 and Line 7 take the same number and sequence of additions and multiplications.

**Correctness of ObSSA.** We prove the correctness of ObSSA using two lemmas and a proposition.

**Lemma 1.** Given a query \( Q \) to ball \( B \), ObViolationPruning(\( M_Q, M_B, P \)) of ObSSA returns an \( i' \) such that \( P'(i,j) = 0 \) iff \( \forall v \) in \( V_Q \) is not matched to \( v_j \) in \( V_B \) in a strong simulation relation.

**Proof.** (Sketch). The lemma is established by a case analysis on Child w.r.t Tab. I. Case 4 is the only case that has a violation and Child yields 0. The analysis on Parent is similar.

**Proposition 1.** ObSSA returns non-zero iff there is a strong simulation between \( Q \) and \( B \).

**Proof.** (Sketch) By Lemma 1, after ObViolationPruning, \( P(i,j) = 0 \) iff there is a violation for \( (i,j) \). In Line 24, the product of \( \text{sum}_1 \) can be non-zero only if for each \( v_j \in V_Q \), there exists a match in \( B \) (i.e., non-zero \( \text{sum}_1 \) in Line 24). In Line 26, the sum of all the elements in the column of the ball center can be non-zero only if there exists a vertex in \( Q \) that can be matched to the ball center (i.e., non-zero \( P(i,j) \) in Line 26).

**Example 3.** Following up with Example 2, Fig. 2(b) shows the initial \( P \) and Fig. 2(c) lists the non-zero entries in \( P' \) after the execution of ObViolationPruning(). A non-zero value of prod and \( \text{sum}_2 \) is returned by ObCheckSS(). Thus, ObSSA returns a non-zero result, which indicates \( Q \prec_S G \).

**Time complexity.** Let \( N_{\text{ball}} \) denote the number of balls computed in Line 2. In ObViolationPruning(), Line 11 is repeated for \( |V_B| \cdot |V_Q| \) times. Assume that both the addition and multiplication operations take \( O(1) \) time. Then, Eq. 1 takes \( O(|V_Q| \cdot |V_B|) \) time. Since there are \( |V_Q| \cdot |V_B| \) elements in matrix \( P \) to be computed using Eq. 1, ObViolationPruning() needs \( O(|V_Q|^3 \cdot |V_B|^3) \) time. For ObCheckSS(), computing prod and \( \text{sum}_2 \) needs \( O(|V_Q| \cdot |V_B|) \) time. Hence, the total time complexity is \( O(N_{\text{ball}} \cdot (|V_Q|^3 \cdot |V_B|^3)) \).

**C. Encoding and Encryption**

To make ObSSA secure, we present the encoding for the matrices and adopt a partial homomorphic encryption scheme (PHE) to facilitate secure matrix computations.

**Encoding for \( M_Q \).** \( \forall i,j \in [1, |V_Q|], M_Q(i,j) \) is encoded as follows.

\[
M_Q^e(i,j) = \begin{cases} 
q, & \text{if } M_Q(i,j) = 0; \\
1, & \text{otherwise},
\end{cases}
\]

where \( q \) is a large prime number. Based on this encoding, we use a symmetric encryption scheme called cyclic group based encryption (CGBE) [20] to encrypted the encoding for \( M_Q \). Fully homomorphic encryption scheme (FHE) [13] is not adopted because of their efficiency problem. CGBE is a partially homomorphic encryption scheme supporting both additions and multiplications but the scheme is correct only when the computed ciphertext corresponds to a plaintext that does not exceed a predefined limit.

**Encryption.** We now recall the definition of CGBE, namely the key generation \( \text{Gen} \), encryption \( \text{Enc} \), decryption \( \text{Dec} \) functions as follows.

- \( \text{Gen} \) generates a cyclic group \( (g) = \{g^i | i \in \mathbb{Z}_p, g^i \in \mathbb{Z}_n \} \) with the generator \( g \) and order \( p \) (\( p \gg q \)). Note that \( p \) should be a large prime number. Moreover, \( \text{Gen} \) generates a uniformly random secret key \( x \in [1, p] \). It outputs \( p \) as the public value for the computation on \( SP \) and \( (x, g) \) as the private keys.
- \( \text{Enc} \). \( \text{Enc} \) takes a message \( m \) and the secret key \( (x, g) \) as input, chooses a random value \( r \) and produces the ciphertext \( c \) as output, as follows,

\[
c = mrg^x \pmod{p}.
\]

- \( \text{Dec} \). \( \text{Dec} \) takes a ciphertext \( c \) and the secret key \( (x, g) \) as input and computes the decrypted message as output, as follows,

\[
mr = cg^{-x} \pmod{p}.
\]

Note that \( \text{Dec} \) only decrypts the ciphertext \( c \) as a product of message \( m \) and a random value \( r \), where \( r \) is different for each ciphertext. We use the following to encrypt the encoding of \( M_Q \).

\[
mr = \begin{cases} 
0 \pmod{q}, & \text{which encrypts the plaintext 0;} \\
\mathbb{Z}_q^+, & \text{which encrypts the plaintext 1.}
\end{cases}
\]

As discussed in [20], there is a negligible chance of false positives that \( \mathbb{Z}_q^+ = 0 \pmod{q} \), since \( q \) is a large prime number. Moreover, CGBE is correct only if the computed ciphertext corresponds to a plaintext not larger than the order \( p \). Otherwise, there is an overflow.

**D. Encrypted ObSSA and its Limitations**

Using the encoding and encryption schemes, we describe an encrypted ObSSA algorithm, and analyze its limitations.
Encrypted ObSSA. In encrypted ObSSA, the adjacency matrix of $Q$ is encrypted by the client. $G$ is encoded for correct computation. The encrypted ObSSA, the ball and the vertex mapping matrix are generated by the SP as in ObSSA. In Line 5 (Lines 9-17), each element $P(i,j) \in P$ is computed iteratively $|V_B| \cdot |V_Q|$ times. If the size $|V_B|$ of ball $B$ is large, it not only takes a long runtime but also makes the value of the plaintext for the encoding $P(i,j)$ large. The following steps illustrate the first two iterations of ObViolationPruning. Denote $P_i$ to be the computed vertex mapping matrix $P$ after the $i$-th iteration, and $P_0$ is $P$.

1) The encrypted query $\overrightarrow{M_{Q,n}}$, denoted as $\overrightarrow{M_{Q,enc}}$, contains $g^q$. To achieve homomorphic computation on Eq. 1 in each iteration (Lines 11-16) for correct decryption in the end (Line 17), each polynomial of $\sum_{i=1}^{|V_Q|} (M_B(j,l) \cdot P_0(k,l))$ must contain the same power of $g^x$ with $\overrightarrow{M_{Q,enc}}$, where $x$ is a value derived from the iteration number.

2) The first iteration of ObViolationPruning is different from the others. Note that the initial matrix $P_0$ contains plaintext $SP$ can replace the possibly large plaintext value of $\sum_{i=1}^{|V_Q|} (M_B(j,l) \cdot P_0(k,l))$ with the chosen ciphertexts of 0 and 1, denoted by $c_0$ and $c_1$, respectively, provided by the client. If $\sum_{i=1}^{|V_Q|} (M_B(j,l) \cdot P_0(k,l))$ is a non-zero, $SP$ replaces it with $c_1$, and otherwise, $c_0$.

3) The remaining iterations are done in the ciphertext domain.

**Overview in ciphertext for its plaintext value.**

1) The largest possible value of $P_1$ can be analyzed as follows. Recall $Enc$ of $CGBE$. The ciphertext is either $g^x r^q$ or $g^x r^q g^x$, which corresponds to 1 or 0, and each ciphertext contains the secret key $g^x$. W.l.o.g, we analyze the computation for Child in Eq. 1. Consider the largest possible value of Child in the first iteration, i.e., each ciphertext has the largest value $g^x r^q$. Then, the value of Child is as follows.

$$\prod_{k=0}^{|V_Q|} (g^x r^q + g^x r^q) \cdot (r^q)^{|V_Q|}$$

2) In the second iteration, we ensure the two components of Child (Eq. 1b) have the same order of $g^x$, so that they can be correctly added. First, $\sum_{i=1}^{|V_Q|} (M_B(j,l) \cdot P_1(k,l))$ contains the power of the private key $(g^x)^{2|V_Q|}$. Second, we replace $\overrightarrow{M_{Q,enc}}(i,j)$ with $\overrightarrow{M_{Q,enc}}(i,j)$. As a consequence, after the second iteration, the largest possible value of $P_2(i,j)$ is the following.

$$P_2(i,j) = [(g^x)^{2|V_Q|} \cdot (r^q)^{|V_Q|} + |V_B| \cdot (2r^q)^2|V_Q|] \cdot (r^q)^{|V_Q|}$$

where $[(r^q)^2|V_Q| + |V_B| \cdot (2r^q)^2|V_Q|] \cdot (r^q)^{|V_Q|}$ corresponds to the plaintext of the largest possible value for $P_2(i,j)$, which is larger than $(2r^q)^2|V_Q|$. For example, consider the experimental settings of [20] and [16]. We assume that both $q$ and $r$ are of 32 bits. If the public value $p$ is of 4096 bits and the size $|V_Q|$ of the query is 5, $(2rq)^{2|V_Q|}$ needs 6500 bits.

3) Similar to the second iteration, we can analyze that Child of the $n$th iteration contains a factor $(g^x)^{2|V_Q|}$.

From the above analysis, we can observe that in practice, the value of the plaintext can be larger than the order $p$ of $CGBE$ and there can be an overflow.

**IV. A PRACTICAL INEXACT SOLUTION EncSSA**

Due to the limitation discussed in Sec. III-D, we propose a practical, inexact algorithm, called EncSSA, which comprises three pruning ideas, namely, localized pruning, neighbor-label pruning, and path pruning.

**A. Localized Violation Pruning**

Localized violation pruning (presented in Alg. 3 TwoIterPruning) contains two main techniques. Firstly, we conduct violation detection for a bounded number of times. Secondly, we replace large ciphertext with small ciphertext to allow more violation detections before overflow happens.

We illustrate the techniques with an example of Child, shown in Fig. 4(a). (The case of Parent is similar.) We also denote $G[v_2, 2]$ as $B$. Alg. 3 checks whether a match from $u_1 \in V_Q$ to $v_2 \in V_B$ violates Def. 1, i.e., conducting the violation detector on $P_0(1,2)$.

In the first iteration (Lines 3-8) of Alg. 3, Line 5 computes the value of Child(1,2) in Eq. 1(b). Then, Line 6 searches for the value of Child(1,2) in the column of the original ciphertext in the replacement table shown in Fig. 4(b) and updates Child(1,2) with the corresponding ciphertext for replacement. The rows of the replacement table in Fig. 4(b) list all the possible cases of Child. (The replacement table has $2|V_Q| - 1$ rows.) As the term $\overrightarrow{M_{Q,enc}}(i,j)$ of Eq. 1(b) is computed and provided by the client to $SP$, the client can also precompute a ciphertext that corresponds to a small plaintext for replacement.

The second iteration (Lines 9-12) conducts violation detection for the second time. It takes the encrypted vertex mapping matrix $P_1$ as input. Hence, Alg. 3 performs violation pruning in the encrypted domain to yield $P_2$.

**Analysis of query size and overflow.** In Fig. 4(b), the value of Child is at most $g^x r^q$. Therefore, the value of each element in $P_1$ is at most $g^{2x}(rq)^2$ (Lines 7-8). With Eq. 1, we know that the value of each element in $P_2$ is:

$$P_2(i,j) \leq [(g^x)^{2|V_Q|} \cdot (r^q)^2 + |V_B| \cdot (2r^q)^2] \cdot (r^q)^{|V_Q|}$$

Assume that both $q$ and $r$ are of 32 bits and the public order $p$ of $CGBE$ is of 4096 bits. If $|V_B|$ needs 20 bits, then,
Algorithm 3: Localized violation pruning TwoIterPruning

Input: The replacement tables $T_B$ of $Q$, the initial vertex mapping matrix $P_0$, and the adjacency matrix $M_B$

Output: The vertex mapping matrix $R_1$ after two iterations of violation detections (cf. ObViolationPruning of Alg. 2)

1. generate two null matrices $P_0$ and $R_1$ of the same size as $P_0$;
2. foreach element $P_0(i, j)$ do // 1st iteration
   3. if $P_0(i, j) ≠ 0$ then
      4. compute the values of Child and Parent: $\text{Eq}.$1b and 1c
      5. Search the replacement tables $T_B$ for each $v_i$'s children and parents, then replace Child and Parent with small ciphertext;
      6. $P_1(i, j) ← \text{Child}$; $P_1(i, j) ← \text{Parent}$;
3. return $R_1$;

Fig. 4: (a) The example query, ball and matching; and (b) the replacement table $T_R$ of large ciphertext with small ciphertext

(a) $Q$, the ball, and checking matching from $u_4$ to $v_2$

(b) Replacement table for $u_4$’s children

Prop. 2 is then used to prune balls that must contain violations due to the balls’ centers. We illustrate the steps of pruning with 3-path as follows.

- **Client**: For each vertex $u$ of $Q$, the client enumerates all possible $k$-paths starting from $u$ using $|\Sigma_Q|$ labels. For example, given $Q$ as shown in Fig. 2(a), the client computes an encrypted 3-path table $T_B$ of all the $|V_Q| \cdot A_2^2 = 6$ possible 3-paths, where $A$ is the permutation symbol, as Tab. II shows some of them. For each 3-path $p$ in $T_B$, there is a ciphertext corresponding to a plaintext indicating that whether $p$ exists in $Q$. Then, the client sends $T_B$ to the SP.

- **SP**: After receiving table $T_B$, for each ball $B$, the SP runs PathPruning (Alg. 4). Line 3 first uses DFS to traverse ball $B$ and find all the 3-paths starting from the center $v_j$ of $B$. For each vertex $u_i$ of $Q$ that potentially matches $v_j$, Lines 5-11 aggregate the ciphertexts in $C_i$ for pruning $B$.

Specifically, for each $k$-path $p$ starting from $u_i$ (Line 6), if there exists $k$-path $p$ starting at $v_j$ (Line 7), then, Line 8 multiplies $C_i$ with the ciphertext of 1, which is used to ensure the power of the private key $g^{x \cdot A_{|\Sigma_Q|}^{-1}}$ for the correctness of decryption. Otherwise, in Line 10, by Prop. 2, $p$ leads to violation if $u_i$ has $p$. Hence, $C_i$ is multiplied by the ciphertext $c_p$ of $p$ in $T_B$ (Line 10). Line 11 aggregates the ciphertexts $C_i$ of all possible matching vertices into $R_2$. If $\text{Dec}(R_2) = 0$, i.e., $B$’s center $v_j$ cannot be matched to any vertices in $Q$, then $B$ is pruned. The generalization of path-based pruning including $k$-paths ending at each vertex is immediate.

**Analysis.** Given queries with the same $|V_Q|$, $\Sigma_Q$ and $L_Q$, Alg. 4 conducts the same operations without exploiting the query structure. Thus, Alg. 4 is oblivious. Regarding the value of $k$, the pruning of PathPruning when $k = 2$ is covered by TwoIterPruning. Hence, we assume $k > 2$, but $k ≤ |V_Q|$.

Regarding the overflow problem due to CGBE, we remark that Lines 8 and 10 in Alg. 4 use multiplications for $A_{|\Sigma_Q|}^{-1}$ times in total. In the experimental settings in Sec. VI, both $q$ and $r$ are of 32 bits while the public value $p$ is of 4096 bits. Hence, $4096/64 = 64$ times of multiplication are supported. If $A_{|\Sigma_Q|}^{-1} > 64$, we can do the multiplications in batches.

Alg. 4 uses $O(|V_B| \cdot |E_B|)$ time for DFS and $O(|V_Q| \cdot A_{|\Sigma_Q|}^{-1} \cdot k \cdot D_B)$ time for ciphertext aggregation in the worst case, where $D_B$ is the maximum degree of $B$.

**C. Neighbor-Label Pruning**

In previous subsections, we use Child and Parent to check two-hop neighbors and $k$-path to check neighbors that are further away. Here, we propose to use simpler information (just labels) to detect violations due to the neighbors even further away, and yet we do not run into the overflow problem.
Algorithm 4: Path-based pruning PathPruning

Input : The length k, the encrypted k-path table \( P \), the adjacency matrix \( M_B \) and the vertex mapping matrix \( P \)
Output: The encrypted messages \( R_B \) for pruning ball \( B \)

1 Procedure PathPruning \((P, M_B)\):
2   \( R_B \leftarrow 0 \)
3   start a DFS from \( B \)'s center \( v_L \) to compute all \( k \)-paths;
4   foreach \( u_i \) in \( Q \) where \( P(i, j) = 1 \) do
5     \( C_i \leftarrow 1 \);
6     foreach \( k \)-path \( p \) in \( P \) starting at \( u_i \) do
7       if \( p \) is found that starts at \( v_L \), then
8          \( C_i \leftarrow C_i \cdot C_i, c_i; \) // aggregate ciphertext of \( 1 \)
9        if \( C_i \neq 0 \) then
10           \( R_B \leftarrow R_B + C_i; \) // violation if \( u_i \) has \( p \)
11       return \( R_B \);

Algorithm 5: Neighbor-label pruning NLPruning

Input : The hop \( h \), the NL-index \( N L^h \) of \( B \), the center \( v \) of \( B \) and the encrypted NL-matrix \( E M_Q^h \)
Output: The encrypted messages \( R_B \) for pruning ball \( B \)

1 Procedure NLPruning \((E M_Q^h, N L^h)\):
2   foreach \( u_i \) in \( Q \) with \( L_Q(u_i) = L_Q(v) \) do
3       // detect violation using Prop. 3
4       \( C_i \leftarrow 1 \);
5       for \( i \) \( 1 \leq i \leq \Sigma_{Q} \) do
6         if \( u_i \notin L_Q(v) \) then
7            \( C_i \leftarrow C_i \cdot C_i, c_i; \) // aggregate ciphertext of \( 1 \)
8         if \( C_i \neq 0 \) then
9            \( R_B \leftarrow R_B + C_i; \) // violation if \( i \notin N L^h(v) \)
10        return \( R_B \);

NL-index. Given a vertex \( u \) in \( G \) and a hop number \( h \), we precompute the \( h \)-hop neighbors \( N = \{v | \text{dis}(u, v) = h\} \). Then, the \( h \)-hop NL-index of \( u \), denoted by \( N L^h(u) \), is the labels of the \( h \)-hop neighbors, i.e., \( N L^h(u) = \{L_Q(v) | v \in N\} \).

Example 4. Consider the graph \( G \) in Fig. 2(a). \( N L^h(v_2) = \{L_Q(v_3), L_Q(v_8), L_Q(v_7)\} = \{A, B, C\} \).

NL-matrix. Given a query \( Q \) and a hop number \( h \), the NL-matrix of \( Q \) for \( h \)-hop neighbors (exclude cycles), denoted as \( M_Q^h \), is a \( |Q| \times |Q| \) matrix defined as follows: \( M_Q^h(i, j) = 0 \) if the vertex \( v_i \) has an \( h \)-hop neighbor; and \( 1 \) otherwise.

Based on NL-index and NL-matrix, we propose the following proposition to detect violations.

Proposition 3. Consider a query \( Q \) and a ball \( B \) with the center \( v \) and any vertex \( u \) in \( Q \), where \( L_Q(u) = L_Q(v) \). If \( u \) has an \( h \)-hop neighbor that has a label \( l \), but \( l \notin N L^h(v) \), then \( u \) cannot be matched to \( v \).

Since the hop in NL-matrix excludes cycles whereas the \( h \)-hops in NL-index includes cycles, Prop. 3 holds and it can be easily proved by following Def. 1. We propose the NLPruning procedure (Alg. 5). NLPruning takes the hop number \( h \), the NL-index and the encrypted NL-matrix as inputs, and outputs encrypted messages \( R_B \) as the violation detection result.

For each vertex \( u_i \) in \( Q \) having the same label as the ball center \( v \) (Line 2), Lines 4-8 aggregate the ciphertexts in \( C_i \) for pruning \( B \). Specifically, for each label \( l \) in \( \Sigma_Q \) (Line 4), if \( v \) can reach an \( h \)-hop neighbor that has label \( l \) (Line 5), then Line 6 multiplies \( C_i \) by the ciphertext of \( 1 \), to ensure correctness of decryption. Otherwise, in Line 8, there may be a violation for the unreachable \( h \)-hop label \( l \) by Prop. 3. Assumed that \( l \) is the \( j \)-th label in \( \Sigma_Q \). Line 8 multiplies \( C_i \) by \( E M_Q^{j-1}(i, j) \). If \( u_i \) has an \( h \)-hop neighbor with label \( l \), then \( Dec(E M_Q^{j-1}(i, j)) = 0 \) and hence, \( Dec(C_i) = 0 \). Otherwise, \( Dec(C_i) = Z^* \). Line 9 aggregates the ciphertexts \( C_i \)s into \( R_B \). If \( Dec(R_B) = 0 \), i.e., \( B \)'s center \( v \) cannot be matched to any vertices in \( Q \), then \( B \) can be pruned.

Analysis. The value of \( h \) for Alg. 5 should be larger than \( k \) for \( k \)-path in Sec. IV-B to further prune the balls that cannot be pruned by Alg. 4. The analysis of the obliviousness of Alg. 5 and the overflow in Alg. 5 are similar to the one of Alg. 4. Regarding the worst-case time complexity, Alg. 5 takes \( O(|V_Q| \cdot \Sigma_Q^{|NL^h(v)|}) \).

D. EncSSA Algorithm

In this subsection, to answer the strong simulation query, we propose an encrypted inexact algorithm, called EncSSA, using the pruning techniques of Sec. IV-A, IV-B and IV-C.

Pseudo-code for EncSSA (Alg. 6). Taking the data graph \( G \), the encrypted adjacency matrix \( E M_Q \), the diameter \( d_Q \), portion \( p \) and the inputs of Alg. 3, 4, 5 as inputs, the EncSSA
algorithm outputs a superset $R_S$ of balls that contains the strong simulation results for $Q$. At the SP side, for each vertex $v$ in $G$, Line 2 retrieves the adjacency matrix $M_B$ for the ball $B = G[v, d_Q]$ and Line 3 generates the corresponding vertex mapping matrix $R_0$ from $Q$ to $B$. Then, the BallPruning() procedure (Line 5) computes the ciphertexts for pruning $B$. Specifically, Line 8 conducts the violation detection on all vertices in $B$ for only one iteration (i.e., Lines 3-8 in Alg. 3), which is denoted as OneIterPruning(). Then, Line 9 randomly chooses a portion $p (0 < p \leq 1)$ of vertices among $|V_B|$ and conducts Alg. 3 on these vertices. Line 10 conducts Alg. 4 and Line 11 conducts Alg. 5. Since $R_0$ and $R_1$ obtained by Lines 8-9 are matrices, the ResultAggregation() procedure (i) combines the ciphertexts in each row of both matrices by addition (Lines 12-14), and (ii) combines the ciphertexts in the column for the ball center by addition (Lines 15-16). Finally, the SP sends the combined ciphertexts together with $R_2$ and $R_3$ to the client (Line 7).

At the client side, after receiving the ciphertexts, Decryption() first decrypts the ciphertext $R_2$ ($R_3$) and prunes $B$ based on Alg. 4 (Alg. 5) in Line 18 (Line 20). Then, similar to ObCheckSS() in Alg. 2, (i) Line 22 checks whether there exists at least one valid match in $B$ for each vertex in $Q$ and (ii) Line 24 checks whether there exists at least one vertex in $Q$ that can be matched to the ball center. If $B$ cannot be pruned, Line 26 adds $B$ into $R_S$.

Analysis. As introduced in Sec. IV-A, IV-B and IV-C, EncSSA can handle the situation of overflow. Moreover, the computations at the SP side consist of Alg. 3-5, which are oblivious. Then, we analyze the scale of ciphertexts need in EncSSA. Let $N_{ball}$ denote the number of balls computed in Line 1. For each ball, Lines 10-11 (Lines 15-16) both generate $O(1)$ ciphertexts while Lines 13-14 both generate $|V_Q|$ ciphertexts. Therefore, EncSSA needs to transmit in total $N_{ball}(a|V_Q|+b)$ ciphertexts, where $1 \leq a, b \leq 2$, in practice. Regarding time complexities, EncSSA needs $O(N_{ball}|V_Q|^2 |V_B|^2+|E_B|+|V_Q|A^{k-1}_{|V_Q|-1}b\cdot D_B+|V_Q|\Sigma_Q\max((|NL^k(v)|)))$ time at the SP side, and $O(N_{ball}|V_Q|\cdot Dec_k)$ at the client side, where $Dec_k$ is the time for decrypting a ciphertext.

V. PRIVACY ANALYSIS

Due to space restrictions, we present the main ideas of the privacy analysis, but present the detailed derivations in [21].

Lemma 2. CGBE [20] is secure against CPA. $EM_Q$ and $EM_P^p$ is preserved from SP against the attack model.

Proposition 4. The structure of query encrypted by CGBE is preserved from SP against the attack model.

With Lemma 2 and Prop. 4, SP cannot attack the query’s ciphertexts. Then, we analyze Alg. 3-Alg. 5.

For Alg. 3, assume that the vertex mapping matrix has $n$ elements with value 1. Let $A(Q)$ be a function that returns 1 if SP can determine the existence of an edge of $Q$, and 0 otherwise. Then, since Alg. 3 conducts oblivious computation on ciphertexts encrypted by CGBE, we can yield Prop. 5.

TABLE III: Statistics of the real-world datasets

| Graph | $|V_G|$ | $|E_G|$ | $\Sigma_Q$ |
|-------|--------|--------|----------|
| Slashdot | 82,168 | 948,464 | 72, 64, 128 |
| DBLP | 317,080 | 1,049,866 | 32, 64, 128 |
| Twitter | 81,306 | 1,768,149 | 32, 64, 128 |

Proposition 5. After running TwoIterPruning, $Pr[A(Q) = 1] \leq 2^{-n}$.

Prop. 5 states that there is a negligible probability that SP can attack the vertex mapping matrix after conducting the localized violation pruning.

For Alg. 4 (resp. Alg. 5), let $G(R_2)$ (resp. $K(R_3)$) returns 1 if SP can compute the plaintext of the output ciphertext $R_2$ (resp. $R_3$), and 0, otherwise. We analyze the possibilities of $G(R_2)=1$ (resp. $K(R_3)=1$) and yield Prop. 6 (resp. Prop. 7).

Proposition 6. After running PathPruning, $Pr[G(R_2) = 1] \leq 1/2 + \epsilon$, where $\epsilon$ is negligible.

Prop. 6 (resp. Prop. 7) states that there is a negligible probability SP can do so after conducting the path-based pruning (resp. neighbor-label pruning). In addition, we yield Prop. 8 to show that SP cannot infer the plaintext of the structural information of the query from the relations between the encrypted messages of different hops (e.g., the 1-hop adjacency matrix can yield the 2-hop adjacency matrix) needed for neighbor-label pruning. Note that no relations exist in the case for localized violation pruning (path-based pruning).

Proposition 8. Given the encrypted matrices $EM_Q$ and $EM_P^p$, the neighbor information of the query is preserved from SP against the attack model under CGBE.

Putting Props. 5, 6, 7 and 8 together, we have Thm. 1 since the three pruning techniques used independently in EncSSA.

Theorem 1. EncSSA (Alg. 6) preserves the privacy of the query structure against CPA.

VI. EXPERIMENTAL RESULTS

We conducted detailed experiments to investigate the efficiency and effectiveness of our proposed algorithms.

Platform. The prototype used in the experiment is implemented in C++. We used a machine with an Intel Xeon E5-2630 2.2GHz CPU and 256GB RAM running CentOS 7.7 for both the SP and client. We used the GMP libraries to implement CGBE encryption scheme.

Datasets. We used three real-world datasets, namely Slashdot, DBLP and Twitter [22]. Some characteristics of the datasets can be found in Tab. III. Since the vertices in these datasets do not have labels, similar to [9], we generated random labels for them. We use the subscript $|\Sigma_Q|$ to denote the generated datasets, i.e., Slashdot$|\Sigma_Q|$, DBLP$|\Sigma_Q|$ and Twitter$|\Sigma_Q|$.

Query sets. Given a query size $|V_Q|$ and a diameter $d_Q$, the
query generator $QGen^4$ derived a radius $\gamma$. $QGen$ randomly chose a vertex $v$ in the data graph and obtained the induced graph of $v$'s $1$ to $\gamma$-hop neighbors. After random edge deletions, $QGen$ output the maximum connected component if it had a size $|V_Q|$ and diameter $d_Q$.

**Default parameters.** The parameters are described as follows:
- **$CGBE$.** The encoding $q$ and random number $r$ for $CGBE$ were both of 32 bits. The public value was of 4096 bits.
- **Query and graph.** The query sizes $|V_Q|$ varied from 6 to 10. The default value of $|V_Q|$ was 8. According to the analysis in Sec. IV-A, the parameters for $CGBE$ can support $|V_Q| \leq 13$ without overflow. The query diameter $d_Q$ varied from 3 to 5, where the default value of $d_Q$ was 3. The label size $|\Sigma_G| = 32, 64$ or 128 and the default value was 64.
- **Neighbor-label pruning (NL).** The value of hop $h$ varied from 4 to 6. The default value was 4.
- **Path-based pruning (Path).** The value of length $k$ for $k$-path varied from 3 to 5. The default value was 3.
- **Two-iteration violation pruning (twoIter).** We chose vertices randomly in each ball with portion $p$ ($p = 0.1, 0.3$, or $0.5$) for twoIter. This investigated the balance between efficiency and effectiveness of the pruning. The corresponding algorithm was denoted as $twoIter_p$. The default value was 0.3.
- **One-iteration violation pruning (oneIter).** We conducted one iteration on all the vertices for violation pruning.

**Balls.** Tab. IV shows some statistics of the balls generated from random queries under the default setting. *In a nutshell, each query can lead to several thousand balls and each ball contains hundreds of vertices.

**A. Experiment on Efficiency**

**EXP-0. Performance at the client.** Given a query, the client generated the encrypted messages for $EncSSA$ and decrypted the ciphertexts returned by the $SP$ to obtain the results.

1) **Preprocessing.** Given a query $Q$, the client generated the adjacency matrix $M_Q$, the replacement tables $T_R$, the path tables $T_P$, and the NL-matrices $M_Q^h$. The total preprocessing times are all less than 0.15s.

2) **Encryption.** The messages to be encrypted were $M_Q$, $T_R$, $T_P$, and the chosen ciphertexts $c_0$ and $c_1$. Based on the encrypted $M_Q$, i.e., $EM_Q$, the client generated the replacement tables $T_R$ for twoIter. The total encryption times are all less than 0.1s.

3) **Decryption.** The client decrypted the messages generated from Path, NL and twoIter. The total runtimes for decryption of $EncSSA$ in our experiment are all less than 1s.

4) **Message sizes.** As analyzed in Sec. IV-D, the message size is $N_{ball}(a:|V_Q|+b)$, $1 \leq a, b \leq 2$, where $N_{ball}$ is the number of computed balls. The size of each ciphertext for $CGBE$ of 4096 bits is 512 bytes. Take the query for Twitter in the next **EXP-1** as an example. The message size is 24MB, whose transmission time is less than 0.5s in a 100Mbps Ethernet.

**EXP-1. Overall runtimes under the default setting.** For the ease of exhibition, we used boxplots. In $x$-axis, we grouped the balls according to their sizes, whose definition is shown in the figures. Only 1% of balls were beyond $x$ range. We reported the performance as runtimes per ball, and for brevity, we simply called them runtimes. 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. The total runtime of each technique was presented in the parentheses in the subcaption of each figure.

We first investigated the algorithms on the three datasets under the default setting. The results of $Slashdot_{64}$ are presented in Fig. 5. Fig. 5(a) shows that the runtimes of $twoIter_{0.3}$ increase as the ball sizes $|V_B|$ increase. $twoIter_{0.3}$ takes roughly 300ms even for large balls. We remark that oneIter is faster than twoIter_{0.5} but slower than twoIter_{0.3}.

As expected, NL is efficient and not sensitive to $|V_B|$ but the NL-index sizes. Path is also efficient but its runtimes increase as $|V_B|$ increases. This is because Path involved traversing the ball to check all the $k$-paths starting from the ball center.

The results of $DBLP_{64}$ and $Twitter_{64}$ are presented in Fig. 6 and Fig. 7. Similar trends can be observed with the exception of the runtimes of Path of $DBLP_{64}$. The reason is that the
Finally, we report in Fig. 8 the runtimes of strong simulation in the plaintext domain to give a reference to the overhead of privacy preserving computation. We implemented Match() of [9] in our codebase. From the total runtimes indicated in the figures’ subcaptions, the private algorithm is around 2.3x, 17x and 2.7x slower than Match().

**EXP-2. Runtimes under different settings.** We ran the algorithms on Slashdot and vary a parameter at a time. In the following figures, there are only fewer than 1% of outlier performances that cannot be displayed.

**2.1. Varying |VQ|.** Fig. 9(a) shows the results for twoIter0.3 on Slashdot for Q_6, Q_8 and Q_10. It can be observed that the runtime increases as either |VQ| or |V_B| increases. Moreover, the trends of the runtime of twoIter0.3 are slightly superlinear in practice. The runtime variations are larger when either |VQ| or |V_B| becomes larger. These show twoIter0.3 can always process at least 2 balls per second under various settings.

Recall that two-Iter is oblivious, where the runtime depends only on |V_Q| and |V_B|. Hence, we further investigated |V_B| and |VQ|. Fig. 10(a) shows the total number of balls of different |VQ|s. The number of balls for Q_6-Q_10 ranged from 2400 to 4800. Fig. 10(b) shows the variations of |V_B| and Fig. 10(c) reports that many balls have hundreds of vertices. twoIter0.3 can process more than 10 such balls in 1s.

The trends of NL and Path when varying |VQ| are shown in Figs. 9(c) and 9(b). It can be observed that a larger |VQ| leads to a larger runtime. For NL, a larger query has a larger NL-matrix. For Path, more labels also make the path tables larger. Therefore, NL and Path become slower.

**2.2. Varying p for twoIter_p.** In Fig. 11, it can be observed that the runtimes of twoIter_p increase as p increases. When p=1, the runtime of two-Iter1 on large balls can be large. We can tune the runtime by tuning p, but introducing false positives (see Sec. VI-B).

**2.3. Varying k for Path.** Due to space restriction, the detailed results are presented in a technical report [21]. In a nutshell, the runtimes increase when k increases. This is because Path traverses each ball from the center to check all the k-paths. We do not observe the increase in runtimes of DBLP since DBLP balls of DBLP are very sparse, the performance differences due to the traversals for k-paths cannot be observed.

**2.4. Varying h for NL.** The runtimes are generally very small and they increase as h increases [21].

**B. Experiments on Effectiveness**

EncSSA computes a superset of the solution of strong simulation query, in which there can be false positives. Thus, we investigated the accuracy of the results, defined as \(\frac{TP + TN}{TP + TN + FP + FN}\), where \(TP (TN)\) denotes true positive (true negative), and \(FP (FN)\) denotes false positive (false negative), respectively. We remark \(FN=0\).

We tested the effectiveness of oneIter, twoIter, NL, Path and EncSSA independently. We also tested the effectiveness of specific combinations of them on different datasets. For Slashdot and Twitter, EncSSA used twoIter, NL and Path. For DBLP, EncSSA used oneIter, NL and Path. All results are the average from 10 random queries.

**EXP-1. Overall effectiveness under the default setting.** Fig. 12 shows the number of matches obtained by each method and the corresponding accuracy. It can be observed that almost all the methods have accuracies higher than or close to 90%, with only one exception. DBLP is sparse and there are fewer paths with length larger than 3, which reduces the pruning power of Path. Moreover, EncSSA has the highest accuracy since its result is the intersection set of the results obtained by oneIter, twoIter, NL and Path. For EncSSA*, its accuracy is slightly lower than EncSSA’s but its runtime is shorter than EncSSA’s by turning off some specific technique(s).

**EXP-2. Effectiveness under different settings.** Fig. 13 shows the results when varying p = 0.1, 0.3, 0.5, 0.7 and 0.9 for twoIter_p. As expected, a larger p leads to fewer false positives but more runtimes are needed (Fig. 11). Hence, we can observe a trade-off between efficiency and effectiveness in choosing p. We further ran Path and NL. A larger k for Path leads to few false positives. The improvement from DBLP is
not obvious since there are few $k$-paths for $k \geq 4$. In NL, when $h \geq 4$, the accuracy is larger than 80%. However, it does not improve further since the center of each computed ball does not have neighbors larger than 4 hops.

VII. RELATED WORK

There have been works on privacy preserving query processing [23]–[26] in the recent three years. This section includes only graph queries and secure query framework.

**Privacy preserving graph queries.** Cao et al. [17] studied tree pattern queries on encrypted XML documents by pre-determining the traversal order for each query. Cao et al. [18] proposed a filtering and verification method to solve the privacy preserving subgraph isomorphism query (sub-iso) over encrypted graph-structured data in cloud computing. To solve sub-iso in cloud computing, Fan et al. [20] transformed the classic solution into matrix operations. They studied privacy preserving sub-iso under two different models, i.e., the structure information of both query and graph are preserved [20] and only query is preserved [16]. Chang et al. [27] also solved the privacy preserving sub-iso by using the $k$-automorphic graph to protect the structure information of data graphs. Gao et al. [11] studied the privacy preserving strong simulation query [9] in cloud. They used $k$-automorphic graph to protect the structure information of data graphs. However, the structure information of query is not preserved.

**General secure query framework.** Gentry et al. [13] described the first plausible construction for a fully homomorphic encryption (FHE) that supports both addition and multiplication operations on ciphertexts. However, due to the known poor performance, FHE cannot be adopted for this paper. An oblivious RAM simulator (ORAMs), introduced by Goldreich and Ostrovsky [28], is a compiler that transforms algorithms in such a way that the resulting algorithms preserve the input-output behavior of the original algorithms 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 applied in this paper since the query is not known to the SP. Nayak et al. [29] proposed a framework called GraphSC to provide a programming paradigm that allows non-cryptography experts to write secure graph-based algorithm with parallel secure oblivious implementations. Following the Pregel/GraphLab programming paradigm, GraphSC uses three primitives as interfaces, i.e., scatter, gather and apply. However, GraphSC still needs the query structure for violation detection.

VIII. CONCLUSION

This paper investigates the problem of privacy preserving strong simulation queries for large graphs. This paper adopts strong simulation query as it strikes a good balance between matching flexibility and query efficiency. This paper presents an oblivious algorithm for strong simulation queries under the plaintext settings. Then, the paper proposes its encrypted version and several optimizations for an inexact efficient secure algorithm EncSSA. Privacy analysis results are presented. The experimental results have shown the algorithms are efficient and effective. As for future work, we plan to integrate query answer authentication into this work.

ACKNOWLEDGMENT

This work is supported by HKRGC GRF 12232716, 12201518, and 12200819, and CRF C6030-18GF.

REFERENCES