OblivGM: Oblivious Attributed Subgraph Matching as a Cloud Service
Songlei Wang*, Yifeng Zheng*, Xiaohua Jia**, Fellow, IEEE, Hejiao Huang*, and Cong Wang*, Fellow, IEEE

Abstract—In recent years there has been growing popularity of leveraging cloud computing for storing and querying attributed graphs, which have been widely used to model complex structured data in various applications. Such trend of outsourced graph analytics, however, is accompanied with critical privacy concerns regarding the information-rich and proprietary attributed graph data. In light of this, we design, implement, and evaluate OblivGM, a new system aimed at oblivious graph analytics services outsourced to the cloud. OblivGM focuses on the support for attributed subgraph matching, one popular and fundamental graph query functionality aiming to retrieve from a large attributed graph subgraphs isomorphic to a small query graph. Built from a delicate synergy of insights from attributed graph modelling and advanced lightweight cryptography, OblivGM protects the confidentiality of data content associated with attributed graphs and queries, conceals the connections among vertices in attributed graphs, and hides search access patterns. Meanwhile, OblivGM flexibly supports oblivious evaluation of varying subgraph queries, which may contain equality and/or range predicates. Extensive experiments over a real-world attributed graph dataset demonstrate that while providing strong security guarantees, OblivGM achieves practically affordable performance (with query latency on the order of a few seconds).

Index Terms—Cloud-based graph analytics, attributed subgraph matching, privacy preservation, oblivious services.

I. INTRODUCTION

ATTRIBUTED graphs, as one kind of the most popular graph data models [1], have been widely used to capture the interactions between entities in various applications, such as social networks, financial services, and manufacturing industries [2]. With the widespread adoption of cloud computing [3], [4], there has been growing popularity of enterprises resorting to commercial clouds as the back-end to store and query their attributed graphs (e.g., [5], [6], to list a few). While the benefits are well-understood, deploying such graph analytics services in the public cloud also poses threats [7] to the privacy of information-rich attributed graph data and may not be good for the business interests of these enterprises as the graph data is proprietary. Hence, there is an urgent demand that security must be embedded in such cloud-backed graph analytics services from the very beginning, providing protection for the outsourced attributed graphs and queries.

As one of the most fundamental functionalities in querying attributed graphs, attributed subgraph matching, which is the focus in this paper, aims to retrieve from a large attributed graph subgraphs isomorphic to a given small query graph [1]. Attributed subgraph matching is a powerful tool in various applications, such as anti-money laundering [8], chemical compound search [9], and social network analysis [1]. A concrete example is that retrieving all users whose ego-networks are isomorphic to a given ego-network from a social network [10].

While the benefits are well-understood, deploying such graph analytics services in the public cloud also poses threats [7] to the privacy of information-rich attributed graph data and may not be good for the business interests of these enterprises as the graph data is proprietary. Hence, there is an urgent demand that security must be embedded in such cloud-backed graph analytics services from the very beginning, providing protection for the outsourced attributed graphs and queries.

In the literature, privacy-aware graph query processing has received wide attention in recent years. Most of existing works, however, focus on dealing with graph query functionalities that are different from attributed subgraph matching, like privacy-preserving shortest path queries [12], [13], [14], [15] and privacy-preserving breadth-first search [16], [17], [18]. Little work has been done for privacy-preserving attributed subgraph matching, where the state-of-the-art protocol under a similar outsourcing scenario is PGP proposed by Huang et al. [19]. The PGP protocol relies on perturbation techniques—t-closeness and k-automorphism—to protect the attributed graph and subgraph queries. Despite being a valuable design point, PGP is not satisfactory for practical use due to the following downsides.

Firstly, the construction of PGP is tailored for limited subgraph queries with equality predicates, which is far from sufficient for practical use. Indeed practical attributed subgraph
matching systems (such as ORACLE’s PGQL [10] and Amazon’s Neptune [20]) should flexibly support subgraph queries containing equality predicates as well as range predicates. Secondly, PGP relies on the notion of $t$-closeness for protecting attribute values (via generalization), which is not strong in protecting the confidentiality of data content from a cryptographic perspective, as well degrades the quality of matching results. Thirdly, PGP does not consider hiding search access patterns [21], which have been shown to be exploitable for various attacks [22], [23], [24] to learn information about queries and database contents. Therefore, how to enable privacy-preserving attributed subgraph matching is still challenging and remains to be fully explored.

In light of the above, in this paper, we design, implement, and evaluate OblivGM, a new system enabling oblivious attributed subgraph matching services outsourced to the cloud. OblivGM allows the cloud hosting an outsourced encrypted attributed graph to obliviously provide subgraph matching services, providing protection for the attributed graph, subgraph queries, and query results. OblivGM protects the confidentiality of data content associated with the attributed graphs and queries, conceals the connections among vertices in the attributed graph, and hides the search access patterns during the subgraph matching process. Besides, OblivGM supports secure and rich matching functionalities, in which a subgraph query can contain equality predicates and/or range predicates, and oblivious predicate evaluation can be effectively performed at the cloud. At a high level, OblivGM is built from a delicate synergy of insights from attributed graph modelling and advanced lightweight cryptography (such as replicated secret sharing (RSS), function secret sharing (FSS), and secure shuffle). We highlight our contributions below:

- We present OblivGM, a new system enabling oblivious attributed subgraph matching services outsourced to the cloud, with stronger security and richer functionalities over prior art.
- We show how to adequately model the attributed graph and subgraph queries to facilitate secure attributed subgraph matching, and devise custom constructions for encryption of the attributed graph and (randomized) generation of secure query tokens.
- We devise a suite of secure components to support oblivious attributed subgraph matching at the cloud, including secure predicate evaluation over candidate vertices, secure matched vertices fetching, and secure neighboring vertices accessing.
- We formally analyze the security of OblivGM, make a GPU-accelerated full-fledged prototype implementation, and conduct extensive evaluations over a real-world attributed graph dataset (with 107614 vertices and 13673453 edges). The results demonstrate that while providing strong security guarantees, OblivGM has practically affordable performance (with query latency on the order of a few seconds).

The rest of this paper is organized as follows. Section II discusses the related work. Section III introduces preliminaries. Section IV presents the problem statement. Section V gives the detail design of OblivGM. The security analysis is presented in Section VI, followed by the performance evaluation in Section VII. Finally, we conclude this paper in Section VIII.

II. RELATED WORK

A. Graph Search in the Plaintext Domain

Graphs have been widely used to model structured data in various applications (such as social networks, financial transactions, and more [25]), due to their powerful capabilities of characterizing the complex interactions among entities in the real world. As one of the most fundamental functionalities in graph data analytics, graph search has gained wide attention and various algorithms have been proposed for handling different queries on graphs, e.g., subgraph matching [1], graph similarity search [26], graph keyword search [27], breadth-first search [28], and shortest path search [29]. However, all of them consider the execution of graph search in the plaintext domain without privacy protection.

B. Privacy-Aware Graph Query Processing

In recent years, great efforts have been devoted to advancing privacy-preserving graph search. Chase and Kamara [30] propose structured encryption under the searchable encryption framework to support neighboring vertices queries. Privacy-preserving shortest path search [12], [13], [14], [15] and privacy-preserving breadth-first search [16], [17], [18] have also gained wide attention. Another line of work studies privacy-preserving subgraph matching, which is much more challenging because more complex operations are required in the ciphertext domain. Some works [31], [32] only consider structure matching and work on unattributed graphs. Others [19], [33], [34], [35], [36] study privacy-aware attributed subgraph matching, which is more sophisticated as it additionally considers the matching against vertices’ attributes and types. The works [34], [35], [36] target application scenarios different from ours. Specifically, the works [34], [35] focus on graph containment query, i.e., given a query graph $q$ and an attributed graph $G$, they just aim to output whether $q$ is a subgraph isomorphic to $G$ or not, while the work [36] considers publicly known attributed graphs.

The works that are most related to ours are [33] and [19], which aim to securely retrieve from an outsourced attributed graph subgraphs isomorphic to a given query graph, with protection for both the attributed graph and query. The state-of-the-art design is PGP [19], which makes use of $k$-automorphism for obfuscating graph structure and $t$-closeness for generalizing attribute values. As mentioned above, PGP is subject to several crucial downsides in terms of security and functionality, which greatly limit its practical usability. Compared to PGP, OblivGM is much advantageous in that it provides much stronger security, supports richer matching functionalities, and does not rely on parameter tuning for accuracy.

III. PRELIMINARIES

A. Attributed Subgraph Matching

In attributed graphs, vertices represent entities and edges represent the connections between entities. Attributed graphs
Attributed graphs can be formally defined as follows [1].

Definition 1: An attributed graph is defined as \( G = (V, E, T, A) \), where (1) \( V = \{ V_1, \ldots, V_N \} \) is a set of \( N \) vertices; (2) \( E = \{ e_{ij} = (V_i, V_j) : 1 \leq i, j \leq N, i \neq j \} \) is a set of edges; (3) \( T \) is a set of types and each vertex \( f \) associates with ranges.\( \forall \in \), i.e., \( \forall \) has an attribute value within the corresponding range \( q \), i.e., \( \forall f(V_i) \in A \). Specifically, prior works [19], [33] only consider exact matching (i.e., equality predicate), where the attribute of each vertex in the query \( q \) is associated with an “exact value”, and the matching is defined as that each vertex in the subgraph \( g \) has an attribute value equal to the corresponding value in \( q \), i.e., \( \forall f(V_i) = A \). OblivGM considers not only exact matching like [33] and [19] but also range matching (i.e., range predicate), where the attribute of each vertex in \( q \) is associated with a “range” (single-sided or an interval), and the matching is defined as that each vertex in \( g \) has an attribute value \textit{within} the corresponding range \( q \), i.e., \( \forall f(V_i) \in A \). In addition, OblivGM also considers and flexibly supports mixed matching, where some attributes are associated with exact values, while others are associated with ranges.

For clarity, we illustrate an attributed graph \( G \) and a query \( q \) in Fig. 1. \( G \) has three types of vertices or entities, i.e., “university (U)”, “person (P)”, and “company (C)”. The connection between different vertices implies the edge type, such as “friend (P-P)”, “work at (P-C)” and “graduate from (P-U)”. The query \( q \) represents that a user wants to retrieve two persons satisfying the following conditions: (1) both of them graduated from the same university located in “Harbin”; (2) their ages are within [30, 40]; (3) one of them is working at a software company and the other one is working at an Internet company. Then the final matching results consist of two subgraphs:

\[
g_1 := U_1 \xrightarrow{P_1} C_1, \quad g_2 := U_1 \xrightarrow{P_2} C_2.
\]
time (PPT) algorithms: (1) \((k_1, k_2) \sim \text{Gen}(1^\lambda, f)\): Given the description of \(f\) and a security parameter \(\lambda\), output two succinct FSS keys \(k_1, k_2\), each for one party: (2) \((f(x))_1 \sim \text{Eval}(k_1, x)\): Given an FSS key \(k_1\) and input \(x\), output the share \((f(x))_1\).

The security guarantee of FSS is that an adversary learning only one of the keys \(k_1\) and \(k_2\) learns no private information about the target function \(f\) and output \(f(x)\).

IV. PROBLEM STATEMENT

A. System Architecture

OblivGM is aimed at supporting oblivious and encrypted subgraph matching services in cloud computing. Fig. 2 illustrates the system architecture of OblivGM. There are three kinds of entities: the users, the graph owner as the on-premise service front-end (SF), and the cloud servers. The graph owner (e.g., an enterprise or an organization) is in possession of large amounts of information modeled as an attributed graph, and wants to leverage the power of cloud computing for storing and querying this graph. We specifically consider that the graph owner expects cloud computing services to empower subgraph matching queries made by her users (e.g., an enterprise’s employees or consumers) over the attributed graph, a popular and highly useful class of graph queries. Such cloud-empowered graph analytics service paradigm has seen wide adoption in practice (e.g., [5], [6], to list a few). However, due to privacy concerns on the proprietary attributed graph and queries, it is demanded that security must be embedded in such service paradigm from the very beginning, providing protection for the outsourced attributed graph, subgraph matching queries, and the query results. In OblivGM, the power of cloud is split into three cloud servers (referred to simply as \(\mathcal{CS}_{1,2,3}\) hereafter) from different trust domains, which can be hosted by independent cloud providers in practice. Such multi-server model has gained rising popularity in recent years for building practical secure systems in both academia [36], [39], [40], [41], [42], [43] and industry [44], [45]. OblivGM follows such trend and newly explores the support for oblivious and encrypted attributed subgraph matching services in cloud computing.

B. Threat Model and Security Guarantees

1) Threat Model: Similar to prior work using the multi-server setting for security designs [40], [41], [46], [47], [48], [49], we consider a semi-honest and non-colluding adversary model where each of \(\mathcal{CS}_{1,2,3}\) honestly follows our protocol, but may individually attempt to infer the private information. In addition, we assume that \(SF\) and the users are trustworthy parties, since \(SF\) is the owner of attributed graph, and can restrict the ranges of queries that different users are allowed to make using a standard database access control list [50].

2) Security Guarantees: Under the aforementioned semi-honest and non-colluding adversary model, OblivGM guarantees that the cloud servers cannot learn 1) each vertex’s attribute values and exact degree, and the connections between these vertices, in the attributed graph; 2) the target attribute’s value(s) associated with each vertex in subgraph queries; 3) search access patterns. To define search access patterns for oblivious attributed subgraph matching, we adapt the general definitions in searchable encryption [21], which are introduced below.

Definition 4 (Search Pattern): For two subgraph queries \(q\) and \(q’\), define \((q \preceq q’) \in \{0, 1\}\), where if the two queries are identical, \((q \preceq q’) = 1\), and otherwise \((q \preceq q’) = 0\), and “identical” means that both the structure, vertices, and attribute values of \(q\) and \(q’\) are identical. Let \(q = \{q_1, \ldots, q_n\}\) be a non-empty sequence of queries. The search pattern reveals an \(m \times m\) (symmetric) matrix with entry \((i, j) = q_i \preceq q_j\).

In practice, the search pattern implies whether a new subgraph query has been issued before.

Definition 5 (Access Pattern): Given a subgraph query \(q\) on the attributed graph \(G\), the access pattern reveals \(\{g_m = (V_m, E_m)\}\), where \(g_m\) denotes a subgraph of \(G\) isomorphic to \(q\). In practice, the access pattern reveals which vertices are “accessed”, namely, which vertices in \(G\) are matched with the vertices in \(q\). In addition, the access pattern also implicitly reveals the connections between vertices in \(G\) because \(g_m\) is isomorphic to \(q\)’s structure is public. Notably, protecting the search access patterns can defend against a large class of potential leakage-abuse attacks [22], [23], [24].

Similar to the prior works [19], [33], OblivGM considers the following information as public: 1) the schema layout parameters of the attributed graph and subgraph queries, including the number and types of vertices and edges and the types of vertex attributes; 2) the type of predicate associated with each vertex in queries, i.e., whether the predicate is an equality, single-sided range or interval range predicate; 3) the structure of queries. To make the public information of queries more concrete, we consider the query \(q\) in Fig. 1: the attacker learns that the query is

\[
U(\text{Place} = \#) \Rightarrow P(\text{Age} \in [*, *]) \rightarrow C(\text{Field} = \#).
\]  (1)

V. THE DESIGN OF OBLIVGM

From a high-level point of view, OblivGM proceeds through the following four phases: 1) attributed graph and subgraph queries modelling, 2) attributed graph encryption, 3) secure query token generation, and 4) secure attributed subgraph matching. In phase 1, \(SF\) properly models attributed graphs
and subgraph queries so as to facilitate the subsequent oblivious subgraph matching service. In phase 2, \( \mathcal{SF} \) adequately encrypts its attributed graph and then sends the resulting ciphertext to the cloud servers. In phase 3, \( \mathcal{SF} \) parses each subgraph query and generates the corresponding secure query token, followed by sending it to the cloud servers. In phase 4, the cloud servers obliviously retrieve encrypted subgraphs isomorphic to the query from the encrypted attributed graph. In what follows, we elaborate on the details of each phase.

A. Attributed Graph and Subgraph Queries Modelling

1) Attributed Graph Modelling: To represent the structure and non-structure information of an attributed graph \( \mathcal{G} \), our main insight is to delicately adapt the inverted index [51]. Specifically, given a vertex \( v_i \) in \( \mathcal{G} \), we first represent \( v_i \)’s each attribute as a tuple \( (t_j, d_j) \), \( j \in [S] \), where \( S \) is the number of \( v_i \)’s attributes (we write \([S]\) for the set \( \{1, 2, \ldots, S\} \)), \( t_j \) and \( d_j \) are the type and value of the attribute, respectively. Then \( v_i \) can be modeled as \( v_i = \{T_i, id_i, ((t_j, d_j))_{j \in [S]}\} \), where \( T_i \) is \( v_i \)’s type and \( id_i \) is \( v_i \)’s identifier (ID) (i.e., a unique number). In this paper, for clarity of presentation, we use \([\sigma_i]_{i \in [\mu]}\) to represent the set \( \{\sigma_1, \ldots, \sigma_\mu\} \), and omit the subscript \( i \in [\mu] \) when it does not affect the presentation. It is noted that the ID can be regarded as a special attribute with unique value for each vertex in \( \mathcal{G} \). Then we consider how to model the connections between vertices. Since the connection types in an attributed graph are varying, to clearly distinguish between different types, we associate each vertex \( v_i \) with several posting lists, each containing the IDs of \( v_i \)’s neighboring vertices with the same type. A posting list with \( v_i \) is represented as \( P_{v_i}^\mu = \{id_{i,j}\}_{j \in [L]} \), where \( id_{i,j}, j \in [L] \) is the ID of \( v_i \)’s each neighboring vertex with type \( T_{ne} \) and \( L \) is the number of them, i.e., \( L = |P_{v_i}^\mu| \). Therefore, the neighboring vertices of \( v_i \) can be represented as \( \{P_{v_i}^\mu\}_{T_{ne} \in T_N} \), where \( T_N \) is a set of types for the posting lists of \( v_i \).

2) Subgraph Queries Modelling: We then consider how to properly model a subgraph query \( q \). Given a vertex \( v_i \) (named as target vertex) in \( q \), \( v_i \) has the target type \( T_i \) and target attribute \( (t_i, p_{di}) \), where \( t_i \) is the type of the target attribute and \( p_{di} \) indicates the predicate associated with the target attribute. It is noted that \( p_{di} \) can be an exact value, indicating an equality predicate, or a range (single-sided or an interval), indicating a range predicate, which are corresponding to exact matching and range matching, respectively. For simplicity, we assume that each target vertex \( v_i \) only has one target attribute, but more attributes are straightforward, which will be introduced shortly in Section V-D. Therefore, a query \( q \) can be modeled as \( q = \{|v_i = (T_i, (t_i, p_{di}))\}_{i \in [\mu]}\) where \( |q| \) is the number of target vertices in \( q \). In addition, the connections of vertices in \( q \) can be simply represented as physical connections (e.g., via pointers). To make the query modelling more concrete, we consider the query \( q \) in Fig. 1, which can be modeled as

\[
q = \{\{U, (\text{Place}, "Harbin"),
\{P, \text{Age}, "[30, 40]"\}, \{P, \text{Age}, "[30, 40]\} \}
\]

(C, (Field, "software")), (C, (Field, "Internet")),
and its structure is same as Eq. 1.

B. Attributed Graph Encryption

We now introduce how an attributed graph is encrypted in OblivGM so as to support the subsequent secure subgraph matching service. Here we need to encrypt for each vertex the values of its associated attributes and the IDs in its associated posting lists. To achieve high efficiency with lightweight secret sharing techniques, one plausible approach is to apply over each value the common 2-out-of-2 additive secret sharing technique [52]. With such technique, a secret value \( x \in \mathbb{Z}_{2^k} \) is split into two shares \((x)_1, (x)_2 \in \mathbb{Z}_{2^k}\) such that \( x = (x)_1 + (x)_2 \) in \( \mathbb{Z}_{2^k} \). However, to support multiplication over two secret-shared values, such technique requires one-round communication among the cloud servers holding the shares. In the multi-server model, it is highly desirable to make the communication among the cloud servers as little as possible. Therefore, instead of using the standard additive secret sharing technique, OblivGM builds on the technique of RSS [37] under the three-server model, which allows local operations for the cloud servers to perform a number of multiplications for secret-shared values and aggregate the results.

However, OblivGM does not directly use RSS to encrypt each value of the attributes and IDs in the posting lists. Instead, OblivGM encodes each value of the attributes and IDs in the posting lists as a one-hot vector, where all entries are “0” except for the entry at the location corresponding to the value which is set to “1”. The RSS technique is then applied over these one-hot vectors. As will be clear later in Section V-C, such encoding strategy will benefit the subsequent secure query process, allowing high communication efficiency in sending the secure query token in OblivGM.

With the above design intuition, we now describe how \( \mathcal{SF} \) encrypts the attributed graph for outsourcing. Note that the above attributed graph modeling allows \( \mathcal{SF} \) to simply perform encryption for each vertex separately. Specifically, given a vertex \( v_i \in \mathcal{G} \), \( \mathcal{SF} \) first encodes its each attribute value and each ID in the posting lists into a one-hot vector. To save the storage cost, we encode the IDs of vertices with different types separately, and thus the lengths of IDs of vertices with different types are varying. After that, \( \mathcal{SF} \) encrypts these one-hot vectors via RSS in the binary domain:

1) \( v_i = (T_i, \{[id_{i,j}] \}, \{(t_j, [d_{j,i}])\}_{j \in [S]} \) where a one-hot vector is written in bold; 2) \( \{P_{v_i}^\mu\} = \{([id_{i,j}] \} \}_{j \in [L]} \) for each posting list with type \( T_{ne} \).

It is noted that \( \mathcal{SF} \) does not protect the type information (i.e., \( T_i, \{t_j\}_{j \in [S]}, T_{ne} \)) and the number of vertices with the same type (reflected by the length of vertex IDs), because they are insensitive public information [19], [33]. More specifically, the type of a vertex indicates only the public general category of the corresponding entity. For example, it indicates that the entity is a person, a university, or a company. Vertices with the same type must have the same attribute types, e.g., all persons have the attribute of “age” and all universities have the attribute of “location”. The vertices with the same type \( T_i \) have the same length of IDs, which indicates only the number
Algorithm 1: Attributed Graph Encryption

Require: The attributed graph $G$.
Ensure: The encrypted attributed graph $[G^k]$. 

1. Initialize an empty set $\|[G^k]\|^B = \emptyset$.
2. \textbf{while} $G \neq \emptyset$ \textbf{do}
3. \hspace{1em} Select $k$ vertices $\{V_p\}_{p \in [k]}$ with the same type from $G$, and then delete $\{V_p\}_{p \in [k]}$ from $G$.
4. \hspace{1em} # Protect the degrees of $\{V_p\}_{p \in [k]}$.
5. \hspace{1em} \textbf{for} $T_{ne} \in T \mathcal{N}$ \textbf{do}
6. \hspace{2em} Blend some dummy IDs into $\{P_{V_i}^{T_{ne}}\}_{p \in [k]}$ to achieve $|P_{V_i}^{T_{ne}}| = \cdots = |P_{V_k}^{T_{ne}}|$.
7. \hspace{1em} \textbf{end for}
8. # Encrypt the content of the padded vertices:
9. \hspace{1em} Encode the private information of $V_p$, $p \in [k]$ into one-hot vectors: $v_p = \{T_p, \|id_p\|, \{(t_j, d_j)\}_{j \in [S]}\}$ and $P_{V_p}^{T_{ne}} = \{id_{p,j}\}_{j \in [L]}$, $T_{ne} \in T \mathcal{N}$, where $L$ is the length of posting lists after padding.
10. \hspace{1em} Apply binary RSS over the one-hot vectors to produce the ciphertext $v_p = \{T_p, \|id_p\|, \{(t_j, d_j)\}_{j \in [S]}\}$ and $\|P_{V_p}^{T_{ne}}\| = \{id_{p,j}\}_{j \in [L]}$, $T_{ne} \in T \mathcal{N}$.
11. \hspace{1em} $\|[G^k]\|^B \leftarrow \{v_p, \|P_{V_p}^{T_{ne}}\|\}_{T_{ne} \in T \mathcal{N}}$, $p \in [k]$.
12. \hspace{1em} \textbf{end while}
13. Output the encrypted attributed graph $[G^k]$.

of vertices having the public type $T_i$. Therefore, since the type information is common to different vertices, we consider it as public. Such treatment also appears in prior works [19], [33].

A remaining challenge here is that simply encrypting the IDs in each posting list without protecting the length information will leak the vertex degree, which could be exploited by inference attacks [53]. To tackle this challenge, OblivGM adapts the idea of $k$-automorphism [54] and has $SF$ blend some 0-vectors as dummy IDs into each $V_i$’s posting lists to make at least $k - 1$ other vertices with same type as $V_i$ have the same degree as $V_i$. More specifically, we note that each vertex with the same type has the same types of posting lists but their lengths can vary, e.g., each Person vertex has a friend list and a follower list, but the numbers of their friends and followers are varying. Therefore, given vertex $V_i$ of type $T_i$ and posting lists $\{P_{V_i}^{T_{ne}}\}_{T_{ne} \in T \mathcal{N}}$, OblivGM has $SF$ first find $k - 1$ other vertices $\{V_p\}_{p \in [k-1]}$ with type $T_i$, where each $V_p$’s posting list with type $T_{ne}$ has the similar length as that of $V_i$, i.e., $|P_{V_i}^{T_{ne}}| \approx \cdots \approx |P_{V_{k-1}}^{T_{ne}}| \approx |P_{V_k}^{T_{ne}}|$, $T_{ne} \in T \mathcal{N}$. Then $SF$ blends some 0-vectors as dummy IDs into them to achieve $|P_{V_i}^{T_{ne}}| \approx \cdots \approx |P_{V_{k-1}}^{T_{ne}}| \approx |P_{V_k}^{T_{ne}}|$, $T_{ne} \in T \mathcal{N}$. After that, OblivGM lets $SF$ apply RSS over the true and dummy IDs. Since the attribute values are also encrypted in RSS, each vertex has at least $k - 1$ other “symmetric vertices” in $G$ and the encrypted attributed graph is a $k$-automorphism graph [54].

Finally, the ciphertext of $G$ can be represented as $\|[G^k]\| = \{v_i, \|P_{V_i}^{T_{ne}}\|\}_{T_{ne} \in T \mathcal{N}}\}_{i \in [N]}$, where $T \mathcal{N}$ is a set of posting lists’ types of $V_i$ and $N$ is the number of vertices in $G$. Algorithm 1 describes how $SF$ encrypts $G$. $SF$ sends the public information and the RSS shares of $\|[G^k]\|$ to $CS_{[1,2,3]}$, respectively.

C. Secure Query Token Generation

Given a subgraph query $q$, $SF$ then generates a secure query token in a custom way. As modeled in Section V-A, a subgraph query $q$ is in the form $q = \{V_i = (T_i, (t_i, pd_i))\}_{i \in |q|}$. What should be protected is the value information $pd_i$ for the predicate because $T_i$ and $t_i$ refer to the types of vertices and attributes, which are public information [19], [33]. OblivGM flexibly supports both equality predicate and range predicate, so $pd_i$ can refer to an exact value or a range.

With the secure subgraph matching service run among the three cloud servers, OblivGM aims to minimize the communication among the cloud servers. We identify the newly developed technique—function secret sharing (FSS)—as an excellent fit for our purpose, which allows low-interaction secure evaluation of a function among multiple parties [55]. Specifically, we observe two FSS constructions as a natural fit for the two types of predicates targeted in OblivGM: distributed point functions (DPFs) [55] for equality predicates and distributed comparison functions (DCFs) [56] for range predicates. The FSS-based DPF consists of the same algorithms as Definition 3, which allows two servers to obliviously evaluate a point function $f_{a,\beta}$, outputting secret-shared $\beta$ if input $a$, otherwise, outputting secret-shared 0. Similarly, DCF is for a comparison function $g_{a,\beta}$, which outputs secret-shared $\beta$ if $x < a$, otherwise, outputs secret-shared 0. Analogously, DCF can also describe the functions $x > a$, $x \leq a$ and $x \geq a$.

In addition, constructions for interval containment (IC) build on DCFs to express functions of the form $a < x < a'$ (denoted as $g_{a,a',\beta}$). Analogously, IC can also describe the functions $a \leq x < a'$, $a < x \leq a'$ and $a \leq x \leq a'$. Applying the advanced FSS techniques in OblivGM, however, is not straightforward and needs delicate treatment. In particular, in OblivGM the values to be evaluated via the FSS technique are not in plaintext domain and each cloud server holds shares of the values. However, the FSS-based evaluation process requires the cloud servers to work on identical inputs for producing correct outputs. To address this issue, one relatively simple yet effective approach as proposed by Boyle et al. [55], [56] is to have the cloud servers open additively masked versions of secret values and tailor the generation of the FSS keys for evaluation over the masked values. While this basic approach can protect the secret value while allowing FSS-based evaluation, it has two critical limitations: i) evaluating the same private predicate on different encrypted attribute values in $\|[G^k]\|$ requires a (large number of) fresh FSS keys, imposing high computation and communication overhead over $SF$; ii) the evaluation for each secret-shared value requires the cloud servers to have one-round communication (for opening a masked version of the secret value), leading to high cloud-side communication overhead either.

As such, OblivGM does not build on the above basic approach and makes a delicate treatment for high efficiency. It is recalled that in the attributed graph encryption phase, each value demanding protection is encoded into a one-hot vector. The adoption of such encoding strategy, inspired by [42] and [39], is actually useful in providing an alternative way to avoid fresh FSS keys in OblivGM for evaluating the same
is in the form of

\[ \text{we consider the query in Fig. 1, whose secure query token} \, \text{tok}_q = \{ [T_i, (t_i, K_i)] \}_{i \in [q]} \].

\textbf{Ensure:} The encrypted matching results \([\mathbb{g}_m]\).

\begin{itemize}
  \item[1. Initialization:] \(\mathcal{CS}_{[1,2,3]}\) initialize an empty set \([\mathcal{Q}]\).
  \item[2. for \(i \in [q]\) do]
  \begin{itemize}
    \item[3.] \(V_i := (T_i, (t_i, K_i))\).
    \item[4.] if \(V_i\) is a start vertex in \(\text{tok}_q\) then
      \begin{itemize}
        \item[5.] Set \([V_c]\) as all vertices with type \(T_i\) in \([\mathbb{g}^2]\) and set \([\mathbb{id}_{V_c}], (\mathbb{d}_{V_c})\) as these vertices' IDs and values of attribute with type \(t_i\), respectively.
    \end{itemize}
  \end{itemize}
  \item[6. end if]
  \begin{itemize}
    \item[7.] \(\mathcal{X}_{V_i} = \text{secEval}(\mathcal{X}_{V_i}, K_i)\).
    \item[8.] \(\mathcal{X}_{V_i} := \text{secFetch}(\mathcal{X}_{V_i})\).
    \item[9. \(\mathcal{CS}_{[1,2,3]}\) add \(\mathcal{X}_{V_i}\) to \(\mathcal{Q}\).
    \item[10. \(\mathcal{CS}_{[1,2,3]}\) add \(\mathcal{X}_{V_i}\) to \(\mathcal{Q}\).
    \item[11. \(\mathcal{CS}_{[1,2,3]}\) reorganize \(\mathcal{Q}\) into subgraphs \([g_i]\).
  \end{itemize}
\end{itemize}

Algorithm 2: Secure Attributed Subgraph Matching secMatch

Require: The encrypted attributed graph \([\mathbb{g}^2]\); a secure query token \(\text{tok}_q\).

\(\text{where } K_1, \ldots, K_5 \text{ are generated with the same target function output } \beta = 1 \text{ in } \mathbb{Z}_2, \text{ and correspond to } f_{\text{Harbin}}^{a}, g_{\text{Harbin}}^{a} = 30, a' = 40, g_{\text{Software}}^{a}, f_{\text{Software}}^{a} \text{ and } f_{\text{Internet}}^{a}.\)

**D. Secure Attributed Subgraph Matching**

1) Overview: Upon receiving the secure query token \(\text{tok}_q\) from \(\mathcal{SF}\), the cloud servers collaboratively perform the secure subgraph matching process over the encrypted attributed graph \([\mathbb{g}^2]\) and obtain encrypted subgraphs \([\mathbb{g}_m]\) that are isomorphic to \(q\). OblivGM provides techniques that allow the cloud servers to search over the encrypted attributed graph while being oblivious to search access patterns. Our construction is comprised of three components: secure predicate evaluation over candidate vertices (denoted as \(\text{secEval}\)), secure matched vertices fetching (denoted as \(\text{secFetch}\)), and secure neighboring vertices accessing (denoted as \(\text{secAccess}\)).

At a high level, secure subgraph matching proceeds as follows at the cloud in OblivGM. Given a current target vertex \(V_i \in \text{tok}_q\), OblivGM provides \(\text{secEval}\) to have the cloud servers first perform secure predicate evaluation over candidate vertices in the encrypted attributed graph (i.e., vertices with the same type as \(V_i\)) and produce encrypted predicate evaluation results. Then, based on the encrypted evaluation results, OblivGM provides \(\text{secFetch}\) to allow the cloud servers to obliviously fetch the encrypted matched vertices which satisfy the predicates based on the encrypted predicate evaluation results produced from \(\text{secEval}\). Afterwards, based on each matched vertex’s encrypted ID, OblivGM then needs to allow the cloud servers to obliviously access the IDs and attribute values of each matched vertex’s neighboring vertices, which are used as the candidate vertices of the next-hop target vertex in \(\text{tok}_q\). The above process runs iteratively until all target vertices in \(\text{tok}_q\) are processed. Finally, \(\mathcal{CS}_{[1,2,3]}\) reorganize the matched vertices into subgraphs based on the public structure of \(\text{tok}_q\), and delete incomplete subgraphs who do not have the complete structure as \(\text{tok}_q\), and then output the final encrypted matching results \([g_m]\). In Algorithm 2, we give OblivGM’s complete construction for the secure subgraph matching process at the cloud, which relies on the coordination of the three components: \(\text{secEval}\), \(\text{secFetch}\), and \(\text{secAccess}\), following the aforementioned workflow. For clarity, we illustrate the secure attributed subgraph matching process in Fig. 3. In what follows, we elaborate on the design of each component.

2) Secure Predicate Evaluation Over Candidate Vertices:

For simplicity of presentation, we start with introducing how to allow \(\mathcal{CS}_{[1,2,3]}\) to evaluate a single predicate over the candidate vertices for a target vertex in the secure query token.

Given a target vertex \(V_i = (T_i, (t_i, K_i)) \in \text{tok}_q\), \(\mathcal{CS}_{[1,2,3]}\) need to first retrieve its candidate vertices \(\{V_c\}\)'s \([\mathbb{id}_{V_c}]\) (i.e., IDs) and \([\mathbb{d}_{V_c}]\) (i.e., the values of the attribute with type \(t_i\)) from \([\mathbb{g}^2]\). We note that there are two cases here that need to be treated separately: 1) If \(V_i\) is a start vertex in the query and has no antecedent vertices (e.g., vertex \(U\) of query \(q\) in Fig. 1), then \(\{V_c\}\) are the vertices with type \(T_i\) in \([\mathbb{g}^2]\). \(\mathcal{CS}_{[1,2,3]}\) can locally set \(\{V_c\}'\) IDs and the values of attribute with type \(t_i\)

OblivGM oblivious GM

Algorithm 3 Secure Predicate Evaluation secEval
Require: The candidate vertices’ attribute values \([\{d_v\}_c]\) and the encrypted predicate \(K_i\).
Ensure: The encrypted evaluation results \([\{x_{v_c}\}_c]\).

1: for \([d_{v_c}] \in [\{d_v\}_c]\) do
2: \(CS_{[1,2,3]}(d_{v_c})\) locally evaluate \(K_i\) on \([\{d_v\}_c]\) by Eq. 3.
3: \(CS_{[1,2,3]}(d_{v_c})\) re-share the results to achieve \([x_{v_c}]\) in RSS.
4: end for
5: \(CS_{[1,2,3]}(d_{v_c})\) output the encrypted results \([x_{v_c}]\).

\[
\begin{align*}
\text{Fig. 3. Illustration of secure attributed subgraph matching.}
\end{align*}
\]
A shuffle is performed without the cloud servers knowing the permutation. Since the candidate vertices are shuffled, we can safely open the predicate evaluation results and identify which shuffled vertices are the matched ones. Here what we need is a technique that can perform secure shuffling in the secret sharing domain. In particular, given a secret-shared dataset with an ordered set of records $[\mathbf{D}] = ([\mathbf{r}_1], \ldots, [\mathbf{r}_n])$ (named as table; each record $[\mathbf{r}_i]$ is a row in $[\mathbf{D}]$ and can denote each candidate vertex’s encrypted information in our context), we need a secret-shared shuffle protocol that allows the parties holding the shares to jointly shuffle the records in $[\mathbf{D}]$ and produce secret shares of the result $[\pi(\mathbf{D})]$, while no party can learn the permutation $\pi(\cdot)$. We identify that the state-of-the-art protocol from [18] is well suited for our purpose, as it allows secret-shared shuffling in the RSS domain. Algorithm 5 shows the secret-shared shuffle protocol, and we write $[\mathbf{D}] = \text{secShuffle}(\mathbf{D})$ to denote the protocol. OblivGM adapts secShuffle to instantiate the above idea for handling the case of multiple matched vertices.

4) Secure Neighboring Vertices Accessing: With the encrypted ID $[[\mathbf{id}_{ne}]]$ produced for each matched vertex $V_m$, $CS_{[1,2,3]}$ then need to access information of each matched vertex’s neighboring vertex. The support for multiple target neighboring vertices is straightforward, where $CS_{[1,2,3]}$ handle with each of them independently. We first introduce how $CS_{[1,2,3]}$ obliviously fetch each $V_m$’s neighboring vertices’ IDs $[[\mathbf{id}_{ne}]]$ via $V_m$’s ID $[[\mathbf{id}_{m}]]$. It is noted that the type of neighboring vertices $V_{ne}$ is $T_{ne}$, i.e., the type of the next-hop target vertex in $\text{tok}_q$. Therefore, $V_m$’s posting list $[[P_{T_{ne},m}]]$ with type $T_{ne}$ contains the needed $[[\mathbf{id}_{ne}]]$. $CS_{[1,2,3]}$ should obliviously fetch $[[P_{T_{ne},m}]]$ from all candidate vertices’ $V_c$’s posting lists $[[P_{T_{ne},c}]]$. Our key insight is to utilize the benefits that $V_m$’s ID $[[\mathbf{id}_{m}]]$ is encoded as a one-hot vector and protected via RSS. Specifically, OblivGM lets $CS_{[1,2,3]}$ obliviously and each bit $[[\mathbf{id}_{m}][c]], c \in [C]$ by each candidate vertex $V_c$’s posting list $[[P_{T_{ne},c}]]$, and then XOR the AND operation results to obtain $V_m$’s posting list $[[P_{T_{ne},m}]]$. Formally, $CS_{[1,2,3]}$ perform the following:

$$
[[P_{T_{ne},m}]] = \bigoplus_{c=1}^{C} \left([\mathbf{id}_{m}][c]\right) \otimes \left([\mathbf{id}_{c}]ight) \in [L_{max}],
$$

Algorithm 5 Building Block: Secret Shuffling secShuffle [18]

Require: The ordered set of records $[[\mathbf{D}]]$ in binary RSS; the seed of random value generator; $CS_1$ and $CS_2$ hold $s_{12}$; $CS_2$ and $CS_3$ hold $s_{23}$; $CS_3$ and $CS_1$ hold $s_{31}$.

Ensure: The shuffled records $[[\pi(\mathbf{D})]]$.

1. $CS_1$ and $CS_2$ use $s_{12}$ locally generate $\pi_{12}$, $T_{12}$ and $R_2$.
2. $CS_2$ and $CS_3$ use $s_{23}$ locally generate $\pi_{23}$ and $T_{23}$.
3. $CS_3$ and $CS_1$ use $s_{31}$ locally generate $\pi_{31}$, $T_{31}$ and $R_1$.
4. $CS_1$: $X = \pi_{31}(\pi_{12}(\mathbf{D}_{1} \oplus \mathbf{D}_{2} \oplus T_{12}) \oplus T_{31});$ sends $X$ to $CS_2$.
5. $CS_2$: $Y = \pi_{12}(\mathbf{D}_3 \oplus T_{12});$ $C_1 = \pi_{23}(X \oplus T_{23}) \oplus R_2;$ sends $Y$ and $C_1$ to $CS_3$.
6. $CS_3$: $C_2 = \pi_{23}(X \oplus T_{31} \oplus T_{23}) \oplus R_1;$ $R_3 = C_1 \oplus C_2;$ sends $R_3$ to $CS_2$.
7. $CS_1$ holds $(R_1, R_2), CS_2$ holds $(R_2, R_3),$ and $CS_3$ holds $(R_3, R_1)$ as the final secret shares of $[[\pi(\mathbf{D})]]$.

Algorithm 6 Secure Neighboring Vertices Accessing secAccess

Require: The matched vertices’ IDs $[[\mathbf{id}_{ne}]]$; the neighboring vertices’ type $T_{ne}$ and attribute type $a_{ne}$.

Ensure: Neighboring vertices’ $[[\mathbf{id}_{ne}]]$, $[[\mathbf{d}_{ne}]]$.

1. for $V_{ne} \in \{V_m\}$ do
2. $[[P_{T_{ne},m}]] = \bigoplus_{c=1}^{C} \left([\mathbf{id}_{m}][c]\right) \otimes \left([\mathbf{id}_{c}]/\otimes[[L_{max}]]ight)$.
3. $CS_{[1,2,3]}$ regard $[[P_{T_{ne},m}]]$ as a table $[[\mathbf{D}]]$.
4. $[[\mathbf{D}]] = \text{secShuffle}(\mathbf{D})$.
5. for $[[\mathbf{id}_{ne}]] \in [[\mathbf{D}]]$ do
6. Open $[[V_{ne}]] = \bigoplus_{x=1}^{X} \left([\mathbf{id}_{ne}][x]\right)$.
7. if $y_{ne} == 1$ then
8. Add $[[\mathbf{id}_{ne}]]$ into the outputs $[[\mathbf{d}_{ne}]]$.
9. $[[\mathbf{d}_{ne}]] = \bigoplus_{x=1}^{X} \left([\mathbf{id}_{ne}][x]\right) \otimes \left([\mathbf{id}_{c}]/\otimes[[L_{max}]]ight)$.
10. Add $[[\mathbf{d}_{ne}]]$ into the outputs $[[\mathbf{d}_{ne}]]$.
11. end if
12. end for
13. end for
14. $CS_{[1,2,3]}$ output $[[\mathbf{id}_{ne}]]$ and $[[\mathbf{d}_{ne}]]$. 

Algorithm 4 Secure Matched Vertices Fetching secFetch

Require: $\{V_c\}$’s $[[\mathbf{X}_c]]$, $[[\mathbf{y}_m]]$, and $[[\mathbf{d}_{vm}]]$.

Ensure: Matched vertices’ $\{V_{ne}\}$’s $[[\mathbf{id}_{ne}]]$ and $[[\mathbf{d}_{vm}]]$.

1. $CS_{[1,2,3]}$ initialize empty sets $[[\mathbf{id}_{ne}]]$, $[[\mathbf{d}_{vm}]]$.
2. if Case I then
3. $[[\mathbf{d}_{vm}]] = \bigoplus_{c=1}^{C} \left([\mathbf{id}_{vm}][c]\right) \otimes \left([\mathbf{X}_c]\right)$.
4. $[[\mathbf{d}_{vm}]] = \bigoplus_{c=1}^{C} \left([\mathbf{X}_c]\right) \otimes \left([\mathbf{y}_m]\right)$.
5. end if
6. if Case II then
7. $CS_{[1,2,3]}$ regard the inputs as a secret-shared table $[[\mathbf{D}]] = ([\mathbf{X}_c], [\mathbf{y}_m], [\mathbf{d}_{vm}])$.
8. $[[\mathbf{D}]] = \text{secShuffle}([[\mathbf{D}]]).$
9. $CS_{[1,2,3]}$ open $[\mathbf{x}_c] \in [[\mathbf{D}]], c \in \{1, C\}$.
10. for $c \in \{1, C\}$ do
11. if $\mathbf{x}_c$ then
12. $([\mathbf{id}_{ne}].\text{add}([\mathbf{d}_{ne}].\text{add}([\mathbf{d}_{vm}])))$.
13. end if
14. end for
15. end if
16. $CS_{[1,2,3]}$ output $[[\mathbf{id}_{ne}]]$ and $[[\mathbf{d}_{vm}]]$. 

where \([\text{id}_{v_e}^l]\) is the \(l\)-th ID in \(v_e\)’s posting list \([P_{v_e}^{T_{ne}}]\) and \(L_{\text{max}}\) is the maximum length of all candidate vertices’ posting lists. Correctness holds since there is only one 1 in the one-hot vector \(\text{id}_{v_e}^l\), whose location corresponds to \(v_e\)’s location in \(G\), and thus only the IDs in \(v_e\)’s \(P_{v_e}^{T_{ne}}\) will be kept.

However, since the lengths of different candidate vertices’ posting lists are varying and there are also some dummy IDs in some posting lists (to achieve \(k\)-automorphism when encrypting the attributed graph as introduced in Section V-B), the fetched \([P_{v_e}^{T_{ne}}]\) may contain some invalid IDs, which will incur undesirable performance overheads. Therefore, OblivGM lets \(CS_{[1,2,3]}\) further obliviously refine \([P_{v_e}^{T_{ne}}]\) to filter out these invalid IDs. We observe that the invalid IDs are 0-vectors, and thus OblivGM lets \(CS_{[1,2,3]}\) first locally XOR each bit of \([\text{id}_{v_e}^l]\) \(\in\) \([P_{v_e}^{T_{ne}}]\):

\[
\{y_{v_e}\} = \bigoplus_{x=1}^{X} [\text{id}_{v_e}^l[x]],
\]

where \(X\) is the length of \(\text{id}_{v_e}^l\), which is the number of vertices with type \(T_{ne}\) in \([G^k]\). \(y_{v_e} = 0\) indicates that \(\text{id}_{v_e}^l\) is a 0-vector and an invalid ID. After that, a naive method is to let \(CS_{[1,2,3]}\) obliviously permute the encrypted IDs in \([P_{v_e}^{T_{ne}}]\) before opening \([\{y_{v_e}\}]\). Since the encrypted IDs are shuffled, we can safely open \([\{y_{v_e}\}]\) and identify which shuffled \([\text{id}_{v_e}^l]\) are the invalid IDs. Specifically, OblivGM lets \(CS_{[1,2,3]}\) regard \([P_{v_e}^{T_{ne}}]\) as a table, where each \([\text{id}_{v_e}^l]\) \(\in\) \([P_{v_e}^{T_{ne}}]\) is a record, and then obliviously shuffle \([P_{v_e}^{T_{ne}}]\) via \(\text{secShuffle}\) followed by opening the XOR operation results \([\{y_{v_e}\}]\) to filter out the invalid IDs. By this way, \(CS_{[1,2,3]}\) can obliviously obtain the accurate \([P_{v_e}^{T_{ne}}]\), without knowing the search pattern.

After that, \(CS_{[1,2,3]}\) should obliviously fetch the encrypted value \([\text{d}_{v_e}^l]\) of each neighboring vertex \(v_e\)’s attribute with type \(T_{ne}\) via its ID \([\text{id}_{v_e}^l]\), where \(T_{ne}\) is the type of the target attribute associated with the next-hop target vertex in \(tk_{g_e}\). Our key insight is to utilize the benefits that \([\text{id}_{v_e}^l]\) \(\in\) \([P_{v_e}^{T_{ne}}]\) is encoded as a one-hot vector and protected via RSS as before. Specifically, OblivGM lets \(CS_{[1,2,3]}\) first locally retrieve the vertices with type \(T_{ne}\) from \([G^k]\), and then locally retrieve the values of their attributes with type \(T_{ne}\), denoted as \([\{\text{d}_{v_e}^l\}]_{x \in [X]}\). After that, \(CS_{[1,2,3]}\) obliviously AND each bit \([\text{id}_{v_e}^l[x]]\), \(x \in [X]\) by \([\text{d}_{v_e}^l]\), and then XOR the AND operation results to obtain \(v_e\)’s attribute value \([\text{d}_{v_e}^l]\). Formally, \(CS_{[1,2,3]}\) perform the following:

\[
[\text{d}_{v_e}^l] = \bigoplus_{x=1}^{X} [\text{id}_{v_e}^l[x]] \otimes [\text{d}_{v_e}^l].
\]

Then all matched vertices’ neighboring vertices’ IDs and attribute values compose \([\{\text{id}_{v_e}^l\}]\) and \([\{\text{d}_{v_e}^l\}]\). Finally, \(CS_{[1,2,3]}\) set \([\{\text{id}_{v_e}^l\}]\) and \([\{\text{d}_{v_e}^l\}]\) as new candidate vertices’ \([\{\text{id}_{V_l}\}]\) and \([\{\text{d}_{V_l}\}]\) for the use in the next-hop target vertex matching.

VI. Security Analysis

We follow the simulation-based paradigm [57] to prove the security guarantees of OblivGM. We start with defining the ideal functionality \(F\) for oblivious and encrypted attributed subgraph matching, which comprises the following parts:

- **Input.** \(SF\) submits the attributed graph \(G\) and a subgraph query \(q\) to \(F\).
- **Computation.** Upon receiving \(G\) and \(q\) from \(SF\), \(F\) retrieves the subgraphs \(\{g_m\}\) isomorphic to \(G\) from \(SF\).
- **Output.** \(F\) returns subgraphs \(\{g_m\}\) to \(SF\).

We allow \(F\) to leak \(\text{leak}(F) = (\text{schema}^G, q, \text{struct}^q)\) as defined in Section IV-B, where \(\text{schema}^G\) and \(\text{struct}^q\) are the schema layout parameters of \(G\) and \(q\) and \(\text{struct}^q\) is the structure of \(q\). Let \(\Pi\) denote a protocol for secure attributed subgraph matching realizing the ideal functionality \(F\). \(\Pi\)’s security is formally defined as follows.

**Definition 6:** Let \(A\) be an adversary who observes the view of a corrupted server during \(\Pi\)’s execution. Let \(\text{View}_{\text{Real}}^{\text{Real}}[\{\Lambda\}]\) denote \(A\)’s view in the real world experiment. In the ideal world, a simulator \(S\) generates a simulated view \(\text{View}_{\text{Ideal}}^{\text{Ideal}}[\{\Lambda\}]\) to \(A\) given only the leakage \(\text{leak}(F)\). After that, \(A\) can be a PPT adversary \(\exists\) a PPT simulator \(S\) s.t. \(\text{View}_{\text{Real}}^{\text{Real}}[\{\Lambda\}] \approx \text{View}_{\text{Ideal}}^{\text{Ideal}}[\{\Lambda\}]\).

**Theorem 1:** According to Definition 6, OblivGM can securely realize the ideal functionality \(F\) when instantiated with secure DPFs, DCFs, secret shuffling and a pseudo-random function, assuming a semi-honest and non-colluding adversary model.

**Proof:** OblivGM consists of three secure subroutines: 1) attributed graph encryption \(\text{encGraph}\); 2) secure query token generation \(\text{genToken}\); 3) secure attributed graph matching \(\text{secMatch}\). Each subroutine in OblivGM is invoked in order as per the processing pipeline and their inputs are secret shares. Therefore, if the simulator for each subroutine exists, then the complete protocol is secure [58], [59], [60]. Since the roles of \(CS_{[1,2,3]}\) in these subroutines are symmetric, it suffices to show the existence of simulators for \(CS_1\).

- **Simulator for \(CS_1\) in \(\text{encGraph}\).** Since \(CS_1\) only receives the RSS-based secret shares during \(\text{encGraph}\), the simulator for \(\text{encGraph}\) can be trivially constructed by invoking the RSS simulator. Therefore, from the security of RSS [37], the simulator for \(\text{encGraph}\) exists.
- **Simulator for \(CS_1\) in \(\text{genToken}\).** Since \(CS_1\) only receives FSS keys (i.e., \(\{K_i\}\)) apart from the public information, the simulator for \(\text{genToken}\) can be trivially constructed by invoking the FSS simulator. Therefore, from the security of FSS [55], [56], the simulator for \(\text{genToken}\) exists.
- **Simulator for \(CS_1\) in \(\text{secMatch}\).** It is noted that \(\text{secMatch}\) (i.e., Algorithm 2) consists of three components and each component is invoked in order as per the processing pipeline. We analyze the existence of their simulators in turn:
  - **Simulator for \(CS_1\) in \(\text{secEval}\).** Since in each function loop of \(\text{secEval}\) (i.e., Algorithm 3), \(CS_1\) evaluates FSS keys on the independent secret shares, we only analyze the existence of simulator for one
function loop. At the beginning of a function loop, \( CS_1 \) has two FSS keys \( \langle k_1^1, k_1^2 \rangle \) and secret shares \( \langle d_{V,1}, 1 \rangle, \langle d_{V,2}, 2 \rangle \), later outputs the evaluation results \( x_1^1, x_1^2 \). Since these information \( CS_1 \) views is all legitimate in FSS, the simulator for the evaluation can be trivially constructed by invoking the simulator of FSS. After that, \( CS_1 \) receives a secret share from \( CS_2 \), i.e., re-sharing in RSS, and thus the simulator for the re-sharing can be trivially constructed by invoking the RSS simulator. Therefore, from the security of FSS \( [55], [56] \) and RSS \( [37] \), the simulator for \( secEval \) exists.

- Simulator for \( CS_1 \) in \( secFetch \). It is noted that there are two cases in \( secFetch \) (i.e., Algorithm 4). Since case I consists of basic operations (i.e., \( \oplus \) and \( \otimes \)) in RSS, the simulator for it can be trivially constructed by invoking the simulator of RSS. At the beginning of case II, \( CS_1 \) has secret shares \( \{ (x_{V,1})_1 \mid \{ |d_{V,1} \}_1 \} \), \( (x_{V,2})_2 \mid \{ |d_{V,2} \}_2 \), and later receives secret shares in secure shuffling and secret shares \( \{ (x_{V,1})_2 \} \) from \( CS_2 \) to recover \( \hat{x}_{V,1} \). Therefore, from the security of secret shuffling \( [18] \) and RSS \( [37] \), the simulator for \( secFetch \) exists.

- Simulator for \( CS_1 \) in \( secAccess \). Similar to the analysis for \( secFetch \), the simulator for \( secAccess \) (i.e., Algorithm 6) can be trivially constructed by invoking the simulator of RSS and the simulator of secure shuffle \( secShuffle \). Therefore, from the security of secret shuffling \( [18] \) and RSS \( [37] \), the simulator for \( secAccess \) exists.

We now explicitly analyze why OblivGM can hide search access patterns as follows.

- **Hiding the search pattern.** Given a query token, each \( CS_{1,2,3} \) only receives the FSS keys (i.e., \( \{ K_i \} \)) apart from the public schema layout parameters and the structure of the query. The security of FSS guarantees that even encrypting the same value multiple times will result in different FSS keys indistinguishable from uniformly random values. Therefore, from the security of FSS \( [55], [56] \), \( CS_{1,2,3} \) cannot determine whether a new query has been issued before (except knowing whether the public structure was used before). In addition, in the process of secure subgraph matching, OblivGM lets \( CS_{1,2,3} \) shuffle the (binary) evaluation results \( \{ |x_{V,1} \} \) and \( \{ |y_{V,2} \} \) before opening them. From the security of secure shuffle \( [18] \), even processing the same queries multiple times will result in different orders of the opened results. Since \( \{ x_{V,1} \} \) and \( \{ y_{V,2} \} \) are bit-strings, secure shuffle ensures that even processing the same query multiple times will result in different opened bit-strings at each time. So these opened binary evaluation results will not indicate whether two queries are the same or not. Therefore, OblivGM can hide the search pattern.

- **Hiding the access pattern.** As per Definition 5, the access pattern in fact indicates whether a vertex in the encrypted attributed graph \( \mathbb{G}^k \) is a matched vertex, namely, whether it will appear in the matching results \( \{ |G_m \} \). Since the matched vertices are (obliviously) determined in \( secFetch \), we only need to analyze \( secFetch \). Recall there are two cases in \( secFetch \). For case I, it does not leak the access pattern apparently since all processing is in secret sharing domain and nothing is opened. For case II, before opening the evaluation results \( \{ |x_{V,1} \} \) of candidate vertices, OblivGM lets \( CS_{1,2,3} \) obliviously shuffle \( \{ |x_{V,1} \} \| \{ |d_{V,1} \} \| \{ |d_{V,2} \} \) which breaks the mapping relationship between the candidate vertices and \( \mathbb{G}^k \). Therefore, from the security of secure shuffle \( [18] \), OblivGM can hide the access pattern.

The proof of Theorem 1 is completed.

**Discussion.** Attacks exploiting search access patterns have received wide attention and hiding these patterns is crucially important, which OblivGM ambitiously explores and provides corresponding guarantees. Additionally, we note that there are emerging volume-based attacks \( [61], [62], [63] \) exploiting the volume of results. However, our dedicated assumptions make the attacks ineffective in our context. Specifically, the works \( [61], [62] \) assume that the database is dense, i.e., there is at least one record for every possible value of the plaintext domain, which obviously cannot be achieved in attributed graphs. The work \( [63] \) assumes that the adversary issues independent and identically distributed queries with respect to a fixed query distribution and also does not address encrypted databases for high-dimensional data, which also does not stand in attributed graphs because of the heterogeneity.

**VII. Performance Evaluation**

**A. Setup**

We implement a prototype system of OblivGM in C++. Our prototype implementation comprises \( \sim 1500 \) lines of code (excluding the code of libraries). We also implement a test module with another \( \sim 300 \) lines of code. Three Alibaba Cloud ECS c8g1.2xlarge instances are used to act as \( CS_{1,2,3} \), each has a NVIDIA Tesla V100 GPU with 16 GB memory. All of the instances run Ubuntu 20.04 and have 8 Intel Platinum 8163 CPU cores and 32 GB of RAM. In addition, a Macbook Air with 8 GB RAM acts as \( SF \) to generate and send query tokens. For the adopted cloud environment, the network bandwidth is 2.5 Gbps with an average latency of 0.2 ms.

1) Protocol Instantiation: Note that the all private data in OblivGM is encrypted in binary RSS, and thus we can store these data in bit-strings \( [0,1]^n \). However, bool data (i.e., \{true, false\}) in C++ is stored as an 8-bit data, which will incur undesirable storage overheads when storing bit-strings. Therefore, we divide each private bit-string into 32-bit sub-vectors, and store each sub-vector in a 32-bit unsigned int to save the storage. In addition, for DPFs and DCFs, we set the security parameter \( \lambda \) to 128.

2) Performance Boost From GPU: We note that the overall design of OblivGM is highly parallelizable, which enables us to take advantage of GPU for parallel processing to
achieve a performance boost. Specifically, the GPU architecture is optimized for performing a large number of simple computations on blocks of values, which means that operations like component-wise addition and multiplication of vectors/matrices on GPU can be executed fast [64]. It is noted that the main operations in OblivGM are the addition and multiplication between (one-hot) vectors/mats/pes on GPU can be executed fast [64].

B. Evaluation on Attributed Graph Encryption

Recall that $SF$ needs to model the attributed graph, add some dummy IDs into posting lists, and split the private information into binary RSS. The time and storage cost of encrypting the dataset under different $k \in \{2, 4, 6\}$ (i.e., $k$-automorphism) are $\{54, 66, 69\}$ minutes and $\{161, 179, 198\}$ GB, respectively. It is crucial to notice that such pre-processing cost is one-off and does not affect the online service quality.

C. Evaluation on Secure Query Token Generation

Recall that $SF$ needs to parse a subgraph query into the corresponding secure query token. In particular, $SF$ should generate FSS keys for the value information of each private predicate in the query. We conduct experiments with varying predicate types (i.e., $=$, $<$ and $\leq$) and the number of target vertices in subgraph queries (i.e., $|q| \in \{2, 4, 8\}$), and summarize the time cost and token size in Table I.

1https://docs.nvidia.com/cuda/cudand/index.html

| $|q| = 2$ | $|q| = 4$ | $|q| = 8$ | $|q| = 2$ | $|q| = 4$ | $|q| = 8$ |
|---|---|---|---|---|---|
| $=$ | 0.03 | 0.06 | 0.12 | 0.07 | 0.14 | 0.28 |
| $<$ | 0.04 | 0.08 | 0.16 | 0.07 | 0.14 | 0.28 |
| $\leq$ | 0.08 | 0.16 | 0.32 | 0.14 | 0.28 | 0.56 |

D. Performance Benchmarks on Sub-Protocols

We provide the performance benchmarks of OblivGM’s three sub-protocols under different data sizes, i.e., $secEval$, $secFetch$ and $secAccess$.

1) Computational Efficiency: We first evaluate the computational efficiency of each sub-protocol, with results provided in Table II. From the results, it can be observed that the time cost of the three modules are all not linear in the data size. This benefits from the high parallelizability of OblivGM, which enables different GPU cores to perform independent sub-tasks simultaneously, e.g., securely evaluating the same private predicate on different encrypted attribute values or securely accessing different matched vertices’ neighboring vertices.

2) Communication Efficiency: Fig. 4 illustrates the communication of each sub-protocol. Specifically, the left figure reports the communication of $secEval$ under predicates with type $=$, $<=$, $<$ and the number of target vertices $\in \{1000, 10000\}$. It is noted that the communication of predicates with type $=$ and $<$ is identical since both of them require $CS_{1,2,3}$ to communicate one bit (i.e., re-share $[x_v]$) for each candidate vertex $V_c$, but the communication of predicates with type $\leq$ is $2x$ that of predicates with type $=,$ $<$ since predicates with type $\leq$ consist of two predicates with type $<$. The middle figure reports the communication of $secFetch$ under two cases and the number of candidate vertices $\in \{1000, 10000\}$. It is noted that the communication of case I is kept invariable, irrespective of the number of candidate vertices. That is because case I mainly requires local computation, and $CS_{1,2,3}$ only need to re-share two secret-shared vectors to achieve them in RSS (recall

TABLE I

| Time cost (s) | Token size (MB) |
|---|---|
| $|q| = 2$ | $|q| = 4$ | $|q| = 8$ | $|q| = 2$ | $|q| = 4$ | $|q| = 8$ |
| $=$ | 0.03 | 0.06 | 0.12 | 0.07 | 0.14 | 0.28 |
| $<$ | 0.04 | 0.08 | 0.16 | 0.07 | 0.14 | 0.28 |
| $\leq$ | 0.08 | 0.16 | 0.32 | 0.14 | 0.28 | 0.56 |

TABLE II

| Size | $secEval$ | $secFetch$ | $secAccess$ |
|---|---|---|---|
| 1000 | 1.4 | 1.6 | 1.8 | Case I | 0.1 | 1.2 | 2.1 | ~ |
| 5000 | 2.1 | 2.3 | 2.5 | 0.1 | 1.3 | 2.5 | |
| 10000 | 2.6 | 2.9 | 3.4 | 0.3 | 1.6 | 2.9 | |

Fig. 4. Communication of sub-protocols under different data sizes.

TABLE III

| Equality ($=$) | Range ($<$) | Range ($\leq$) |
|---|---|---|
| $k$ | $|q|$ | Case I | Case II |
|---|---|---|---|
| 2 | 0.7 | 2.1 | 2.5 | 1 | 3.4 | 4.0 | 1.5 | 4.1 | 4.5 |
| 4 | 1.2 | 3.0 | 3.2 | 1.4 | 4.5 | 5.3 | 1.8 | 4.4 | 4.9 |
| 6 | 1.5 | 3.3 | 3.9 | 2.1 | 7.4 | 7.6 | 3.5 | 6.1 | 7.5 |
Algorithm 4). The right figure shows the communication of secAccess under different number of neighboring vertices.

E. Evaluation on Query Latency

We now report the query latency, namely, given a query token, how long it takes to obliviously execute subgraph matching on the encrypted attributed graph and output encrypted matching results. In particular, we first report the overall latency for different queries, and then report the breakdown of the overall query latency.

1) Overall Query Latency: For simplicity, we conduct experiment using 2-hop subgraph queries, with varying predicate types (i.e., =, < and ≪), the number of target vertices (i.e., |q| ∈ {2, 4, 8}) and k-automorphism (i.e., k ∈ {2, 4, 6}), and summarize the experiment results in Table III. It can be observed that the query latency is not linear in |q|. The reason is that different target vertices in the same hop can be evaluated in parallel, which enables us to allocate the secure predicate evaluation of each target vertex on independent GPU cores, and they work simultaneously. To better understand the query latency of OblivGM, we next report the breakdown of the overall query latency.

2) Breakdown of Query Latency: For future work, it would be interesting to explore how to extend our initial research effort to support oblivious attributed subgraph matching services outsourced to the cloud, with stronger security and richer functionalities over prior art. At the core of OblivGM is a delicate synergy of attributed graph modelling and lightweight cryptographic techniques like FSS, RSS, and secret-shared shuffling. Extensive experiments over a real-world attributed graph dataset in the real cloud environment demonstrate that OblivGM achieves practically affordable performance.

For future work, it would be interesting to explore how to extend our initial research effort to support oblivious attributed subgraph matching with malicious security. Other directions for future work are to investigate the support for more complex scenarios such as dynamic graphs, as well as the possibility of leveraging the recent advances in trusted hardware for performance speedup.

F. Evaluation on Server-Side Communication

1) Overall Communication: We evaluate the same queries as that in the above experiments and provide the results in Table IV. It can be observed that similar to the above experiments, the communication is not linear in |q|. To better understand the communication of OblivGM, we next report the breakdown of the overall communication.

2) Breakdown of Communication: Fig. 6 illustrates the breakdown of the overall communication. It is observed that the communication of secEval is inconspicuous in Fig. 6. Because in secEval, $CS_{1,2,3}$ only need to re-share one bit (i.e., $\|x_{c,}\|$) for each candidate vertex $V_c$. The majority of communication is due to secFetch and secAccess since they require secure shuffle.

VIII. CONCLUSION AND FUTURE WORK

We design, implement, and evaluate OblivGM, a new system enabling oblivious attributed subgraph matching services outsourced to the cloud, with stronger security and richer functionalities over prior art. At the core of OblivGM is a delicate synergy of attributed graph modelling and lightweight cryptographic techniques like FSS, RSS, and secret-shared shuffling. Extensive experiments over a real-world attributed graph dataset in the real cloud environment demonstrate that OblivGM achieves practically affordable performance.

We now report the query latency, namely, given a query token, how long it takes $CS_{1,2,3}$ to obliviously execute subgraph matching on the encrypted attributed graph and output encrypted matching results. In particular, we first report the overall latency for different queries, and then report the breakdown of the overall query latency.

## Table IV

| $k$ | $|q| = 2$ | $|q| = 4$ | $|q| = 8$ | $|q| = 2$ | $|q| = 4$ | $|q| = 8$ |
|-----|-----------|-----------|-----------|-----------|-----------|-----------|
| 2   | 64.74     | 87.15     | 103.65    | 72.4      | 94.15     | 112.9     |
| 4   | 78.15     | 99.25     | 120.45    | 98.15     | 119.25    | 140.45    |
| 6   | 91.24     | 117.34    | 137.53    | 111.2     | 137.3     | 147.5     |

Fig. 5. Breakdown of query latency (s) under different values of $k$ (k-automorphism) and $|q|$ (number of target vertices).

Fig. 6. Breakdown of communication under different values of $k$ (k-automorphism) and $|q|$ (number of target vertices).

REFERENCES

[1] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, “Efficient subgraph matching by postponing Cartesian products,” in Proc. ACM SIGMOD, Jun. 2016, pp. 1199–1214.

[2] ORACLE. (2021). 17 Use Cases for Graph Databases and Graph Analytics. Accessed: Apr. 15, 2022. [Online]. Available: https://www.oracle.com/a/ocom/docs/graph-database-use-cases-book.pdf

[3] J.-N. Liu et al., “Enabling efficient, secure and privacy-preserving mobile cloud storage,” IEEE Trans. Dependable Secure Comput., vol. 19, no. 3, pp. 1518–1531, May 2022.
Songlei Wang received the B.E. degree in internet of things from the China University of Petroleum (East China), Qingdao, China, in 2018, and the M.E. degree in computer technology from the Harbin Institute of Technology, Shenzhen, China, in 2021, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Technology. His research interests include cloud computing security and secure machine learning.

Yifeng Zheng received the Ph.D. degree in computer science from the City University of Hong Kong, Hong Kong, in 2019. He is currently an Assistant Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. He worked as a Post-doctoral Researcher with the Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia, and the City University of Hong Kong. His work has appeared in prestigious conferences (such as ESORICS, DSN, ACM AsiaCCS, IEEE PERCOM, IEEE INFOCOM, and IEEE ICDCS), as well as journals (such as IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, and IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS). His current research interests are focused on security and privacy related to cloud computing, the IoT, machine learning, and multimedia. He received the Best Paper Award in the European Symposium on Research in Computer Security (ESORICS) 2021. He has served as a program committee member for more than ten international conferences/workshops, and the TPC Co-Chair for the special session on Private, Secure, and Trust Data Analytics at IEEE DSAA 2022.

Xiaohua Jia (Fellow, IEEE) received the B.Sc. and M.Eng. degrees from the University of Science and Technology of China in 1984 and 1987, respectively, and the D. Sc. degree in information science from the University of Tokyo in 1991. He is currently the Chair Professor with Department of Computer Science, City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, and mobile computing. He is an Editor of IEEE INTERNET OF THINGS JOURNAL, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (2006–2009), Wireless Networks, World Wide Web Journal, and Journal of Combinatorial Optimization. He is the General Chair of ACM MobiHoc 2008, the TPC Co-Chair of IEEE GLOBECOM 2010 Ad Hoc and Sensor Networking Symposium, and the Area-Chair of IEEE INFOCOM 2010 and 2015.

Hejiao Huang received the Ph.D. degree in computer science from the City University of Hong Kong in 2004. She is currently a Professor with the Harbin Institute of Technology, Shenzhen, China. Previously, she was an Invited Professor at INRIA, France. Her research interests include cloud computing, trustworthy computing, formal methods for system design, and wireless networks.

Cong Wang (Fellow, IEEE) is currently a Professor with the Department of Computer Science, City University of Hong Kong. His research interests include data and network security, blockchain and decentralized applications, and privacy-enhancing technologies. He has been one of the Founding Members of the Young Academy of Sciences of Hong Kong since 2017, and has been conferred the RGC Research Fellow in 2021. He received the Outstanding Researcher Award (Junior Faculty) in 2019, the Outstanding Supervisor Award in 2017, and the President’s Awards in 2016 and 2019, all from the City University of Hong Kong. He was a Co-Recipient of the Best Paper Award of IEEE ICDCS 2020, ICPADS 2018, MSN 2015, the IEEE INFOCOM Test of Time Paper Award 2020, and the Best Student Paper Award of IEEE ICDCS 2017. His research has been supported by multiple Government Research Fund Agencies, including the National Natural Science Foundation of China, Hong Kong Research Grants Council, and Hong Kong Innovation and Technology Commission. He served as the TPC co-chairs for a number of IEEE conferences and workshops. He has served as an Associate Editor for IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING (TDS), IEEE TRANSACTIONS ON SERVICES COMPUTING (TSC), IEEE INTERNET OF THINGS JOURNAL (IoT-J), IEEE NETWORKING LETTERS, and the Journal of Blockchain Research.