CryptGraph: Privacy Preserving Graph Analytics on Encrypted Graph

Pengtao Xie and Eric Xing
School of Computer Science, Carnegie Mellon University

Abstract

Many graph mining and analysis services have been deployed on the cloud, which can alleviate users from the burden of implementing and maintaining graph algorithms. However, putting graph analytics on the cloud can invade users’ privacy. To solve this problem, we propose CryptGraph, which runs graph analytics on encrypted graph to preserve the privacy of both users’ graph data and the analytic results. In CryptGraph, users encrypt their graphs before uploading them to the cloud. Cloud runs graph analysis on the encrypted graphs and obtains results which are also in encrypted form that the cloud cannot decipher. The encrypted results are sent back to users and users do the decryption to get the plaintext results. In this process, users’ privacy can be strongly protected. Meanwhile, with the help of homomorphic encryption, the results analyzed from the encrypted graphs are guaranteed to be correct. In this paper, we present how to encrypt a graph using homomorphic encryption and how to query the structure of an encrypted graph by computing polynomials. To solve the problem that certain operations are not executable on encrypted graph, we propose hard computation outsourcing to seek help from users. Using two graph algorithms as examples, we show how to apply our methods to perform analytics on encrypted graphs. Experiments on two datasets demonstrate the correctness and feasibility of our methods.

Introduction

Recently, many cloud based graph analysis services have been launched, including IBM System G, Neo4j, GraphDB, Dydra, Infinity Graph, GraphLab, to name a few. In cloud based graph analytics, graph mining and analysis algorithms are deployed on cloud servers. To use the service, users upload their graphs to the cloud. Cloud runs graph algorithms over users’ graphs and sends the results back to users. Putting graph analytics on the cloud has several benefits. First, users are alleviated from the burden of implementing and maintaining graph algorithms, which are time-consuming and error-prone. Second, many graph storage and management systems have been deployed on the cloud, such as cloud based graph database. Putting graph analytics also on the cloud makes it easier to integrate with other graph management services.

Despite the benefits provided by cloud graph analytics, it suffers a severe problem: invasion of users’ privacy, which is also suffered by many other cloud based services. To use cloud graph analytics, users have to upload their graphs to the cloud, which completely exposes the graphs to malicious attackers and curious cloud owners. Besides, the results analyzed from users’ input graphs are also visible to the cloud, which is again a huge threat of users’ privacy. For some users, their graphs are so sensitive that their privacy cannot be compromised. Examples include atom graph of drugs [DTP 2004], knowledge graph of confidential documents [Poh, Mohamad, and Zaba 2012], social graph containing sensitive information of individuals [Akcora, Carminati, and Ferrari 2012; Hay et al. 2008; Liu and Terzi 2008], to name a few. How to protect users’ privacy is a vital factor and a big challenge for cloud graph analytics.

To solve this problem, we propose a solution called CryptGraph, which runs graph analysis algorithms on encrypted graphs. Users’ graphs and the analysis results are both in encrypted form that the cloud cannot decipher, which ensures that users’ privacy can never be leaked to the cloud. Figure 1 illustrates our solution. Graph algorithms are deployed on the cloud. A user wants to analyze his graph through the cloud graph analytics. However, due to privacy concern, he refuses to let the cloud see the graph. He encrypts the graph before sending it to the cloud. Cloud runs graph analytics over the encrypted graph and gets the results which are also ciphertexts that the cloud cannot decipher. The encrypted results are sent back to the user and the user does the decryption locally to get readable plaintext results.
cloud. Meanwhile, he refuses to let the cloud see the graph which leaks his privacy. He encrypts the graph and sends it to cloud servers. Cloud receives the encrypted graph, runs graph analytics over it and gets the results which are also in encrypted form that the cloud cannot decipher. The ciphertext results are sent back to the user and the user does the decryption locally to get the readable plaintext results. In this process, both the input graph and the output results are encrypted, thereby, the cloud has no chance to learn anything about the user. Users’ privacy can be strongly guaranteed. Now the problem is, with the graph encrypted, how can we run analytics over it and ensure the results are correct? This turns out to be possible with the help of homomorphic encryption. Homomorphic encryption (HE) (Gentry 2009) is an encryption scheme which allows certain computations over encrypted data. For instance, given that $x + y = z$ in the plaintext space, with HE, the equality still holds after encryption: $[x] \oplus [y] = [z]$ , where $[x]$, $[y]$, $[z]$ are the ciphertexts encrypting $x$, $y$, $z$ and $\oplus$ denotes homomorphic addition. Under homomorphic encryption, though data are encrypted, computations can still be performed and are guaranteed to be correct. Practical homomorphic encryption schemes have certain restrictions over the computations they can do. They can compute addition, subtraction, multiplication and negation, but do not support division and comparison. Many graph algorithms involve division and comparison, which makes it very challenging to perform analytics over encrypted graph.

In this paper, we investigate how to encrypt a graph and how to perform computations over the encrypted graph. Our major contributions are:

- Our work is the first one proposing to encrypt a graph with homomorphic encryption, which can protect all the structure information of a graph without losing the ability to perform graph analytics over it.
- We come up algorithmic and system solutions to deal with the problem that division and comparison are not computable on encrypted graph. One is querying graph structure by computing polynomials, which avoids comparison. The other is hard computation outsourcing, which solicits users to do a small fraction of operations which are hard for the cloud.
- On two important graph algorithms and two datasets, we demonstrate that analytics on encrypted graph are correct and feasible.

The rest of the paper is organized as follows. Section 2 introduces related works and Section 3 introduces homomorphic encryption. Section 4 presents our method to run graph analytics on encrypted graphs. Section 5 gives experimental results and Section 6 concludes the paper.

### Related Works

Preserving privacy of graph data has been extensively studied in many works. In general, they can be categorized into two paradigms: encryption based approaches and non-encryption based approaches. Cao et al (Cao et al. 2011), Chase and Kamara (Chase and Kamara 2010), Poh et al (Poh, Mohamad, and Zaba 2012), Yi et al (Yi, Fan, and Yin 2014) developed encryption schemes to enable searching and querying over encrypted graph. Cao et al (Cao et al. 2011) studied how to do containment query, namely whether a graph is a subgraph of another graph. Chase and Kamara (Chase and Kamara 2010) developed a structural encryption scheme which supports neighbor queries and adjacency queries on encrypted graphs. Poh et al (Poh, Mohamad, and Zaba 2012) investigated how to do searching over encrypted conceptual graphs. Yi et al (Yi, Fan, and Yin 2014) studied how to perform reachability queries: whether a vertex is reachable from another vertex. These encryption schemes are mainly designed to support graph searching and querying and they are unable to do computation-oriented graph analytics such as computing clustering coefficient and PageRank.

Non-encryption based approaches (Hay et al. 2008, Liu and Terzi 2008, Zheleva and Getoor 2008, Zhou and Pei 2008) focus on anonymizing vertexes and edges in the graph. Liu and Terzi (Liu and Terzi 2008) proposed a k-degree vertex anonymization method under which for each vertex $v$ there exist at least $k - 1$ other vertexes in the graph with the same degree as $v$. Hay et al (Hay et al. 2008) anonymizes a graph by partitioning the vertexes and then describing the graph at the level of partitions. While these methods can protect the privacy of individuals that are associated with vertexes, most of the edges are exposed, which can be used to identify a lot of structure information of the graph. On the other hand, these methods change the original structure of the graph, thereby the analytics results obtained on the anonymized graph might be quite different from those on the original graph.

### Homomorphic Encryption

In this section, we briefly introduce homomorphic encryption (Gentry 2009) which allows certain computations over encrypted data. Figure 2 illustrates how homomorphic encryption works. The user wants to compute the sum of 3 and 5 using the cloud, but he is reluctant to let the cloud know these two numbers. He encrypts 3 and 5 into ciphertexts,
Graph Analytics on Encrypted Graph

In this section, we discuss how to encrypt a graph and how to perform analytics on the encrypted graph. The major technical challenge is: graph algorithms involve division and comparison which are not computable on encrypted data. We seek two ways to solve this problem and show how they are applied in two graph algorithms: clustering coefficient and multiplication on encrypted data.

Graph Representation and Encryption

To enable management and computation of a graph on the cloud, it is necessary to let the cloud servers know certain information about the graph. Meanwhile, we protect the key information and prohibit the cloud servers from accessing it. In general, we expose the following information to the cloud: number of vertexes, number of edges, degree of vertexes, vertices or between an edge and a vertex, we can use an incidence matrix to represent neighborhood relationships between two vertexes. To encrypt a graph, we encrypt each element of the incidence matrix and weight matrix independently using homomorphic encryption. Let $\tilde{I}$ denote the encrypted incidence matrix and $\tilde{W}$ denote the encrypted weight matrix. All the cloud can see are $\tilde{I}$ and $\tilde{W}$, where the elements are ciphertexts which make no sense to the cloud. Cloud has no way to know whether an edge exists between two vertexes or the direction and weight of an edge, thereby it knows nothing about the structure of the graph.

Query Graph Structure by Computing Polynomials

Querying the structure of a graph is a major operation in graph algorithms, such as selecting the neighboring vertexes of a given vertex, selecting the inbound or outbound edges of a vertex, judging whether a group of vertexes form a strong clique or not, etc. On unencrypted graphs, this can be easily done. To represent neighborhood relationships between two vertexes or between an edge and a vertex, we can use an adjacency matrix. To judge whether a group of vertexes are interconnected, we can use an if-else statement to inspect the connectivities among these vertexes. However, on encrypted graphs, this becomes very hard. First, an adjacency matrix cannot be used since it leaks the structure information of a graph. Second, if-else judgment cannot be performed since comparison is not executable over encrypted data.

To solve this problem, we propose to rewrite all queries into polynomial computations. Polynomials involve addition and multiplication, which are computable by the homomorphic encryption scheme. We assume all queries are boolean, and comparison is not executable over encrypted data. If-else statements to inspect the graph algorithms, such as selecting the neighboring vertexes of a given vertex, selecting the inbound or outbound edges of a vertex, judging whether a group of vertexes form a strong clique or not, etc. On unencrypted graphs, this can be easily done. To represent neighborhood relationships between two vertexes or between an edge and a vertex, we can use an adjacency matrix. To judge whether a group of vertexes are interconnected, we can use an if-else statement to inspect the connectivities among these vertexes. However, on encrypted graphs, this becomes very hard. First, an adjacency matrix cannot be used since it leaks the structure information of a graph. Second, if-else judgment cannot be performed since comparison is not executable over encrypted data.

To solve this problem, we propose to rewrite all queries into polynomial computations. Polynomials involve addition and multiplication, which are computable by the homomorphic encryption scheme. We assume all queries are boolean, and comparison is not executable over encrypted data. If-else statements to inspect the connectivities among these vertexes. However, on encrypted graphs, this becomes very hard. First, an adjacency matrix cannot be used since it leaks the structure information of a graph. Second, if-else judgment cannot be performed since comparison is not executable over encrypted data.
vertex $i$? (equivalently, is there a directed edge from $j$ to $i$?) 3, is vertex $j$ the outbound neighbor of vertex $i$? (equivalently, is there a directed edge from $i$ to $j$). Conjunction queries can be constructed by conjuncting atom queries. For instance, “are vertex $i, j, k$ in a triangle?”. This can be equivalently written as “is $i$ and $j$ connected? AND is $j$ and $k$ connected? AND is $i$ and $j$ connected?”. The answer of a conjunction query is true if and only if the answers of all atom queries are true. On encrypted graphs, we construct polynomials $p : S \rightarrow \{\bar{0}, \bar{1}\}$ to answer these queries, where $S$ is a subset of elements in the incidence matrix $I$. Note that we use the notation $\bar{1}, \bar{0}$ to emphasize that the outputs of polynomials are also encrypted and the cloud has no way to decipher them. In the rest of the paper, if not explicitly stated, $\bar{a}$ denotes the encrypted ciphertext of $a$. For atom queries involving vertex $i$ and $j$, $S = \{\bar{I}_{ij}\}$. For conjunction queries involving a set of vertices $A, S = \big\{\bar{I}_{ij} | i \in A \land j \in A \land i \neq j\big\}$. For the first atom query “is vertex $i$ or vertex $j$ connected with an undirected edge?” it can be directly answered from the incidence matrix $I$ of an undirected graph. $\bar{I}_{ij} = \bar{1}$ means there is an edge between $i$ and $j$; $\bar{I}_{ij} = \bar{0}$, otherwise. So the polynomial would simply be $p_1(x) = x$. The other two queries can be answered by evaluating polynomials over the incidence matrix $I$ of directed graphs. To answer “is vertex $j$ the inbound neighbor of vertex $i$?”, we evaluate $p_2(x) = \frac{1}{2}x^2 + \frac{1}{2}x$ over $\bar{I}_{ij}$. Recall that $\bar{I}_{ij} = \bar{1}$ denotes there is a directed edge from $j$ to $i$ and $\bar{I}_{ij} = \bar{0}$ otherwise. $p_2(x)$ maps $\bar{1}$ to $\bar{1}$ and maps $\bar{0}$ and $\bar{1}$ to $\bar{0}$. Thereby, $p_2(x)$ is capable to pick out the inbound edges. Similarly, to answer “is vertex $j$ the outbound neighbor of vertex $i$?”, we evaluate $p_3(x) = \frac{1}{2}x^2 - \frac{1}{2}x$ over $\bar{I}_{ij}$. This polynomial is compatible with the representation of outbound edges. We call these polynomials answering atom queries as atom polynomials. Knowing how to answer atom queries, we can answer conjunction queries by multiplying atom polynomials. A conjunction query consisting $K$ atom queries $Q^{(c)} = Q_1^{(a)} \land Q_2^{(a)} \land \cdots \land Q_K^{(a)}$ can be answered with a polynomial $p^{(c)} = p_1^{(a)} \cdot p_2^{(a)} \cdots p_K^{(a)}$, which is the product of $K$ atom polynomials. $p^{(c)}$ is called conjunction polynomial. $p^{(c)}$ equals to $\bar{1}$ if and only if all atom polynomials equal to $\bar{1}$, this is consistent with the fact that the answer of a conjunction query is true if and only if the answers of all atom queries are true.

Graph algorithms usually involve operations over a set where elements satisfy certain conditions. For example, to count the triangles containing vertex $i$, we need to calculate the cardinality of a set $\{(i, j, k) | \text{vertexes } i, j, k \text{ form a triangle}\}$. In PageRank, to update the PageRank value for a vertex $i$, we need to sum up all the weighted PageRank values of the inbound neighbor set $\{ j | j \text{ is the inbound neighbor of } i \}$. Cloud can answer the queries “do vertexes $i, j, k$ form a triangle?” and “is $j$ the inbound neighbor of $i$?” by calculating polynomials. However, cloud does not know what the answers are since they are also encrypted. Thereby, cloud has no way to create a set whose elements meet certain conditions. To solve this problem, we propose to do masked computations over elements in the universal set. Operation on each element is masked with the authenticity that whether the element satisfies a certain condition. If the authenticity is $\bar{1}$, the element contributes to the final result. If the authenticity is $\bar{0}$, the element has no influence of the final result. For example, in PageRank, to update the PageRank value of a vertex $i$, one needs to sum up the weighted PageRank values $w_j$ of its inbound neighbors $N(i)$. Since the cloud cannot construct $N(i)$, we can do the masked summation $\sum_{j \in V} \bar{w}_j \cdot p_2(\bar{I}_{ij})$ over all vertexes $V$, where $p_2(x)$ is the atomic polynomial selecting inbound neighbors. If $j$ is an inbound neighbor of $i$, $p_2(\bar{I}_{ij})$ equals to $\bar{1}$ and the contribution $\bar{w}_j$ of vertex $j$ will be counted into the final sum. If $j$ is not an inbound neighbor of $i$, $p_2(\bar{I}_{ij}) = \bar{0}$ and $\bar{w}_j$ will not be contributed to the final sum.

Using the polynomial based graph structure querying methods, we can compute the basic quantities on a graph. To obtain the number of edges in an undirected graph, we can do homomorphic summation of all elements in the encrypted incidence matrix $I$ and multiply the encrypted sum by 0.5 and send the result back to the user. The user does the local decryption and gets the plaintext result. To obtain the degree of vertex $i$ in an undirected graph, we do homomorphic summation of all elements of the $i$th row of $I$. To get the quantities on directed graph, we first evaluate polynomials on the incidence matrix to get encrypted binary matrices representing inbound edges and outbound edges. Then the number of inbound edges, outbound edges, total edges and the indegree and outdegree of a vertex can be easily computed by summing up elements in the binary output matrices.

**Hard Computation Outsourcing**

Recall that homomorphic encryption cannot support division and comparison, which we refer to as hard computations. These two types of computations exist extensively in graph algorithm. For example, comparison is needed in single source shortest paths (SSSP) and division is required in PageRank. While these two operations are hard for homomorphic encryption, they are easy for users. Thereby, we propose to outsource the hard computations back to the users who own the graphs. Whenever the cloud encounters a hard computation $O$ over some data $D$, it sends $\bar{D}$ back to the graph owner and solicits the owner to perform $O$ on $\bar{D}$. Note that, $\bar{D}$ are intermediate results which are in encrypted form. On users’ side, there is a client process in charge of processing the hard computations for the cloud. The client first decrypts $\bar{D}$ into plaintext data $D$, then performs $O$ over $D$ and gets a result $\bar{R}$, $\bar{R}$ is then encrypted into $\bar{\bar{R}}$ which is sent back to the cloud. Cloud receives $\bar{\bar{R}}$ and resumes the computation. Figure 5 illustrates the idea of hard computation outsourcing. Cloud wants to compute the division between two ciphertexts. Unfortunately, division is not computable over encrypted data. Cloud sends the two ciphertexts back to the user and asks the user to do a division. The user decrypts the ciphertexts and gets two numbers 6 and 3. User divides 6 by 3 and gets 2, which is very easy to compute.
The result 2 is encrypted and sent back to the cloud. Cloud gets the encrypted 2 and resumes the computation.

The drawback of hard computation outsourcing is that users are involved into the computation. This in some sense violates the motivation of cloud graph analytics, which is to alleviate users’ burden of implementing and maintaining their own graph algorithms. To mitigate this drawback, we propose two principles. First, hard computations should be a small proportion of the whole computations in a graph algorithm. In other words, most of the computations should happen on the cloud side and users only need to perform a few computations that the cloud cannot do. Second, the computations that users help to do should not be algorithm-specific. Users only need to provide a set of basic computations which are common to all graph algorithms. This ensures that the computations on users’ side are stable no matter how significantly the graph algorithms change on the cloud side.

Applications
In this section, we show how to perform graph analytics on encrypted graph with the above mentioned two techniques. Specifically, we study two graph algorithms: computing clustering coefficient and PageRank.

Clustering Coefficient
Clustering coefficient \( cc(i) \) of a vertex \( i \) in an undirected graph measures how tightly-knit the community is around the vertex. It is defined as

\[
cc(i) = \frac{|\{(j,k) \in E| j \in N(i) \land k \in N(i)\}|}{0.5 \cdot d_i \cdot (d_i - 1)} \quad (1)
\]

where \( N(i) \) is the neighbor set of vertex \( i \) and \( d_i \) is the degree of vertex \( i \). The numerator is basically the count of triangles containing vertex \( i \). To compute the clustering coefficients of vertexes on an encrypted graph, the cloud first computes the degree \( d_i \) of each vertex, which is the sum of all elements of the \( i \)th row in \( \tilde{I} \). Then it computes the denominator \( 0.5 \cdot d_i \cdot (d_i - 1) \). Since division is not supported, cloud can use hard computation outsourcing to compute the reciprocal of \( 0.5 \cdot d_i \cdot (d_i - 1) \). In this algorithm, users only need to compute \( N \) divisions, where \( N \) is the number of vertexes. This amount of computation is light-weighed and meets the first principle of hard computation outsourcing.

Using the polynomial based structure querying method, the triangle count on the numerator can be equivalently written into

\[
|\{(j,k) \in E| j \in N(i) \land k \in N(i)\}| = \sum_{j=1}^{|V|} \sum_{k=j+1}^{|V|} \tilde{I}_{ij} \cdot \tilde{I}_{ik} \cdot \tilde{I}_{jk} \quad (2)
\]

where the equality holds subject to encryption and decryption. The conjunction query \( \{j, k \in E \land j \in N(i) \land k \in N(i)\} \) is evaluated with a conjunction polynomial \( p(x) = x \cdot x \cdot x \). This equation basically examines all possible \((j, k)\) pairs to see whether they form a triangle with \( i \). \((i, j, k)\) forms a triangle if and only if \( \tilde{I}_{ij} = 1, \tilde{I}_{ik} = 1 \) and \( \tilde{I}_{jk} = 1 \), equivalently, \( \tilde{I}_{ij} \cdot \tilde{I}_{ik} \cdot \tilde{I}_{jk} = 1 \).

PageRank
PageRank [Page et al., 1999] is an algorithm to compute the importance of vertexes in a directed and unweighted graph. The underlying assumption is that more important vertexes are likely to receive more links (inbound edges) from other vertexes. The PageRank value \( PR(i) \) of vertex \( i \) can be computed as

\[
PR(i) = 1 - d \cdot \frac{\sum_{j \in N(i)} DO(j)}{N} \cdot PR(j) \quad (3)
\]

where \( N \) is the total number of vertexes. This equation is iteratively computed over each vertex until the PageRank value of each vertex converges.

To run PageRank over an encrypted graph, cloud needs to compute Eq. (3) under the homomorphic encryption scheme. In this equation, \( (1 - d) / N \) and \( 1 / DO(j) \) involve divisions, which are not supported by the encryption scheme. Since cloud knows the fixed parameter \( d \) and the number of vertexes \( N \), it can compute \((1 - d) / N \) on plaintext data. To compute \( 1 / DO(j) \), cloud uses hard computation outsourcing. Cloud first computes \( DO(j) \) for each vertex, then sends \( DO(j) \) to the graph owner and solicits a reciprocal operation. Users decrypt \( DO(j) \) into plaintext value \( DO(j) \), compute the reciprocal \( 1 / DO(j) \), encrypt the reciprocal and send the ciphertext back to cloud. In the whole process, users only need to compute the reciprocal of \( N \) numbers once, which is not a big burden. Computing \( \sum_{j \in N(i)} \frac{1}{DO(j)} PR(j) \) requires to pick out the inbound neighbors of \( i \). To do this, cloud evaluates the atom polynomial \( p(x) = \frac{1}{2} x^2 + \frac{1}{2} x \) on each element \( \tilde{I}_{ij} \) of the \( i \)th row of the incidence matrix \( \tilde{I} \). If \( p(\tilde{I}_{ij}) = 1 \), \( j \) is an inbound neighbor of \( i \). Since cloud cannot know the output of \( p(\tilde{I}_{ij}) \), it does the masked summation over all vertexes

\[
\sum_{j \in N(i)} \frac{1}{DO(j)} PR(j) = \sum_{j=1}^N p(\tilde{I}_{ij}) \frac{1}{DO(j)} PR(j) \quad (4)
\]
Table 1: Clustering coefficients (CC) of Dolphin Social Network

| Graph         | Avg CC | Max CC | Min CC | L2 Dist |
|---------------|--------|--------|--------|---------|
| Unencrypted   | 0.2590 | 0.6667 | 0      | -       |
| Encrypted     | 0.2589 | 0.6666 | 0      | 1.7e-9  |

Table 2: PageRank (PR) values of Political Blogs

| Graph         | Avg PR | Max PR | Min PR | L2 Dist |
|---------------|--------|--------|--------|---------|
| unencrypted   | 0.00044| 0.0878 | 0.0001 | -       |
| encrypted     | 0.00043| 0.0875 | 0.0001 | 1.0e-7  |

In this equation, only inbound neighbors where \( p(\tilde{I}_{ij}) = 1 \) contributes to the final sum.

Experiments

In this section, we evaluate the correctness and speed of graph analytics on two encrypted graphs.

Datasets

We use two datasets in the experiments. The first one is the Dolphin Social Network [Lusseau et al., 2003] which is an undirected social network of frequent associations between 62 dolphins in a community living off Doubtful Sound, New Zealand. The graph has 62 nodes and 159 edges. The second one is the Political Blogs [Adamic and Glance, 2005] which is a directed network of hyperlinks between weblogs on US politics. This graph has 1490 nodes and 16718 edges.

Result

On each of the datasets, we perform graph analytics on the plaintext graph and the encrypted graph and compare their difference. The graph analytics we performed are: 1, compute the clustering coefficient of each vertex in the Dolphin Social Network; 2, compute the PageRank value of each vertex in the Political Blogs graph. Table 1 shows the average, max and min clustering coefficients on the plaintext and encrypted Dolphin Social Network. We also show the L2 distance \( \frac{1}{N} \sum_{n=1}^{N} ||CC^{(p)}_{n} - CC^{(c)}_{n}||^2 \) between the clustering coefficients \( \{CC^{(p)}_{n}\}_{n=1}^{N} \) computed on plaintext graph and \( \{CC^{(c)}_{n}\}_{n=1}^{N} \) obtained from encrypted graph, where \( N \) is the number of vertexes. From Table 1, we can see that the clustering coefficients on plaintext graph and encrypted graph are very close to those obtained from plaintext graph. Through the graph is encrypted, clustering coefficients can still be computed and guaranteed to be correct. Encrypting the graph does not incur significant performance loss. The difference between the clustering coefficients on plaintext graph and encrypted graph mainly comes from the conversion of real numbers to integers when doing the computations on encrypted data.

Table 2 shows the average, max and min PageRank values on the unencrypted and encrypted Political Blogs graph. We also compute the L2 distance \( \frac{1}{N} \sum_{n=1}^{N} ||PR^{(p)}_{n} - PR^{(c)}_{n}||^2 \) between \( \{PR^{(p)}_{n}\}_{n=1}^{N} \) and \( \{PR^{(c)}_{n}\}_{n=1}^{N} \) which are the PageRank values computed from plaintext graph and encrypted graph respectively and \( N \) is the number of vertexes. From Table 2, we can see that the PageRank values obtained from encrypted graph are very close to those obtained from plaintext graph. Again, encrypting the graph does not degrade the performance of PageRank. Table 1 and Table 2 indicate that graph analytics on encrypted graphs are guaranteed to be correct.

To check the speed of graph analytics on encrypted graph, we measure the time of the key operations in each analytic task. In computing clustering coefficients, the key operation is triangle counting. Table 3 shows the average time (in seconds) of triangle counting for each vertex in the Dolphin Social Network. In PageRank, updating PageRank value for each vertex using Eq.(5) and Eq.(6) is the key operation. Table 4 shows the average time (in seconds) of updating PageRank value for each vertex in Political Blogs. Not surprisingly, the computations on encrypted graph are much slower than those on the plaintext graph. However, considering that most graph analytics do not need to be done in realtime, the time spent on encrypted graph is acceptable. Moreover, computations on encrypted graphs can be easily parallelized on the multi-core and distributed computing facilities on cloud platforms, which can improve the speed in great manner and make encrypted graph analytics more feasible.

Conclusion

In this paper, we study how to perform graph analytics on encrypted graph, with the aim to protect users’ privacy. We investigate how to encrypt the graph such that its structural information is protected from leakage. Given the encrypted graph, we put forward to query its structure by computing polynomials. Considering that division and comparison are not computable by homomorphic encryption, we propose hard computation outsourcing, which solicits graph owners to help compute a small fraction of operations that the cloud cannot do. Using two graph algorithms as examples, we show how to leverage polynomial based structure querying method and hard computation outsourcing to conduct graph analysis on encrypted graphs. Experiments on two real datasets demonstrate that analytics on encrypted graph are correct and feasible.
References

Adamic, L. A., and Glance, N. 2005. The political blogosphere and the 2004 us election: divided they blog. In Proceedings of the 3rd international workshop on Link discovery, 36–43. ACM.

Akcora, C. G.; Carminati, B.; and Ferrari, E. 2012. Privacy in social networks: How risky is your social graph? In Data Engineering (ICDE), 2012 IEEE 28th International Conference on, 9–19. IEEE.

Bos, J. W.; Lauter, K.; Loftus, J.; and Naehrig, M. 2013. Improved security for a ring-based fully homomorphic encryption scheme. In Cryptography and Coding. Springer. 45–64.

Brakerski, Z.; Gentry, C.; and Vaikuntanathan, V. 2012. (level- ed) fully homomorphic encryption without bootstrapping. In Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, 309–325. ACM.

Brakerski, Z. 2012. Fully homomorphic encryption without modulus switching from classical gapsvp. In Advances in Cryptology–CRYPTO 2012. Springer. 868–886.

Cao, N.; Yang, Z.; Wang, C.; Ren, K.; and Lou, W. 2011. Privacy-preserving query over encrypted graph-structured data in cloud computing. In Distributed Computing Systems (ICDCS), 2011 31st International Conference on, 393–402. IEEE.

Chase, M., and Kamara, S. 2010. Structured encryption and controlled disclosure. Advances in Cryptology–ASIACRYPT 2010 577–594.

DTP. 2004. Aids antiviral screen (2004).

Gentry, C. 2009. Fully homomorphic encryption using ideal lattices. In STOC, volume 9, 169–178.

Hay, M.; Miklau, G.; Jensen, D.; Towsley, D.; and Weis, P. 2008. Resisting structural re-identification in anonymized social networks. Proceedings of the VLDB Endowment 1(1):102–114.

Liu, K., and Terzi, E. 2008. Towards identity anonymization on graphs. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 93–106. ACM.

Lusseau, D.; Schneider, K.; Boisseau, O. J.; Haase, P.; Slooten, E.; and Dawson, S. M. 2003. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. Behavioral Ecology and Sociobiology 54(4):396–405.

Lyubashevsky, V.; Peikert, C.; and Regev, O. 2013. On ideal lattices and learning with errors over rings. Journal of the ACM (JACM) 60(6):43.

Page, L.; Brin, S.; Motwani, R.; and Winograd, T. 1999. The pagerank citation ranking: Bringing order to the web.

Poh, G. S.; Mohamad, M. S.; and Zaba, M. R. 2012. Structured encryption for conceptual graphs. In Advances in Information and Computer Security. Springer. 105–122.

Suri, S., and Vassilvitskii, S. 2011. Counting triangles and the curse of the last reducer. In Proceedings of the 20th international conference on World wide web, 607–614. ACM.

Yi, P.; Fan, Z.; and Yin, S. 2014. Privacy-preserving reachability query services for sparse graphs. In Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on, 32–35. IEEE.

Zheleva, E., and Getoor, L. 2008. Preserving the privacy of sensitive relationships in graph data. In Privacy, security, and trust in KDD. Springer. 153–171.

Zhou, B., and Pei, J. 2008. Preserving privacy in social networks against neighborhood attacks. In Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on, 506–515. IEEE.