Technical Report 2018-635

THE GRAPH IS THE MESSAGE: DESIGN AND ANALYSIS OF AN UNCONVENTIONAL CRYPTOGRAPHIC FUNCTION

Selim G. Akl
School of Computing, Queen's University
Kingston, Ontario K7L 3N6, Canada
akl@cs.queensu.ca

May 18, 2018

Abstract

An algorithm is described for encrypting a graph to be transmitted securely from a sender to a receiver. In communications terminology, "the graph is the message": its vertices, its edges, and its edge weights are the information to be concealed. The encryption algorithm is based on an unconventional mapping, conjectured to be a trapdoor one-way function, designed for graphs. This function requires the sender and the receiver to use a secret one-time encryption/decryption key. It is claimed that a malicious eavesdropper with no knowledge of the key will be faced with a computational task requiring exponential time in the size of the input graph in order to extract the original plaintext from the ciphertext carried by the encrypted graph. A number of variants to the main algorithm are also proposed.

Keywords and phrases: cryptography, cryptanalysis, cryptology, encryption, decryption, secret key, one-way function, trapdoor one-way function, graph, multigraph, plaintext, ciphertext, one-time key, encryption algorithm, encryption scheme, public-key cryptosystem, malleability, confusion, diffusion, graph database, homomorphic encryption, social networks, unconventional cryptographic function, quantum cryptography protocol, molecular encoding.

1 Introduction

The last two decades, have seen a growing interest in linking the two fields of graph theory and cryptography. Previous work in this endeavor falls into two broad categories:

1. A number of researchers have used graphs as a tool for creating encryption keys, for producing ciphertext from plaintext, for generating digital signatures, and for constructing hash functions [2, 6, 14, 16, 21, 22, 24, 26, 27, 28, 29]. Here, the graph has no contents as such, except for its own structure, and graph traversal is the instrument for building separate cryptographic objects outside of the graph.

2. With the emergence of cloud computing, the focus has been on using homomorphic encryption [10, 15] in order to encrypt graph databases stored in an untrusted location, thus allowing them to be searched and operated upon in various ways without the need for decryption in the cloud [3, 4, 11, 12, 17, 25, 30, 31, 32]. Challenges to the security and efficiency of this type of encryption are cited in [5, 8, 9, 20].

For a survey of previous work on graphs and cryptography, see [23].

This paper has a third and distinct motivation. We are concerned with encrypting graphs that are transmitted from a sender to a receiver over an insecure channel. Specifically,

1. The difference here with the first aforementioned group of previous efforts, is that the graph is not a tool for performing other cryptographic functions. Rather, it is the graph *itself* that is being encrypted. Indeed, the graph is the message.

2. The difference with the second group, is that the encrypted graph is not stored in a database, to be repeatedly accessed and decrypted with the same key by a legitimate user, or cryptanalyzed by an enemy, over long periods of time. Rather, the secrecy of the encrypted message carried by the graph is vital only for a short period of time.

The goal is to develop an encryption function that obeys two basic properties:

1. The encryption/decryption key is difficult to break, and
2. Inverting the encryption function without knowledge of the key takes exponential time in the size of the graph.

Throughout this paper, all graphs to be encrypted are simple, undirected, and complete, for ease of presentation. Minor modifications allow the encryption algorithms presented here to handle graphs that deviate from one or more of these characteristics. Similarly, we assume that the information to be encrypted resides on the edges of the graph; the algorithms can be modified to handle the cases where the information resides in the vertices, instead of the edges, or in both the vertices and the edges.

We begin with a few definitions in Section 2. The proposed encryption algorithm for graphs is described in Section 3, and an analysis of an exhaustive attack to break it follows in Section 4. Possible applications of the algorithm are briefly outlined in Section 5. Some alternative approaches for encrypting a graph are discussed in Section 6. Concluding thoughts are offered in Section 7.

2 Definitions

Three concepts that pertain to the security and computational complexity of the proposed encryption algorithm for graphs are defined in what follows.

2.1 One-way function

A mathematical function f is said to be a *one-way function* if and only if it obeys the following two conditions:

1. Given an argument x, it is *computationally easy* to obtain the value $y = f(x)$, in the sense that the computation can be completed in an amount of time that is at most polynomial in the size of x, while
2. Given a certain value y, it is on average *computationally hard* to invert f, that is, to obtain an x such that $x = f^{-1}(y)$, in the sense that the computation can only be completed in an amount of time that is exponential in the size of y, *in the average case*.

2.2 Trapdoor one-way function

A one-way function f is said to be a *trapdoor one-way function* if, when presented with a certain value y, some additional knowledge allows the computation of an x such that $x = f^{-1}(y)$ to be easy, in the sense of requiring an amount of time that is at most polynomial in the size of y.

2.3 One-time key encryption

A cryptographic system is said to use *one-time key encryption* if every plaintext is encrypted by means of an entirely new key, and that key is never used again for encryption.

We conjecture that the encryption algorithm described in the following section is a trapdoor one-way function, based on one-time key encryption. The algorithm features three graphs. The first graph stores, in plaintext form, the message to be transmitted securely. In the second graph, which is an extension of the first, one level of encryption is implemented. Finally, the second graph is compressed, completing the encryption, and the resulting third graph, holding the message in ciphertext form, is transmitted. Our claim in this paper is that a malicious eavesdropper with no knowledge of the encryption/decryption key will be faced with a computational task requiring exponential time in the size of the input graph in order to extract the original plaintext from the ciphertext carried by the encrypted graph.

3 Graph encryption and decryption

In this section we describe the processes of encrypting and decrypting a graph. The encryption process uses three distinct graphs, constructed successively. It is based on an unconventional mapping, conjectured to be a trapdoor one-way function, that is conceived especially for graph structures. Decryption only uses the last of the three graphs, from which a subgraph is calculated. Both encryption and decryption employ the same secret key.

3.1 The graph G_1

Let G_1 be a simple, complete, undirected, and weighted graph with a set of n_1 vertices,

$$V_1 = \{v_1, v_2, \ldots, v_{n_1}\},$$

and a set of m_1 edges,

$$E_1 = \{e_1, e_2, \ldots, e_{m_1}\}.$$

Note that, because G_1 is complete, $m_1 = n_1(n_1 - 1)/2$. We assume throughout this paper that all edge weights are positive numbers. The weight of the edge (v_i, v_j) connecting the two vertices v_i and v_j in G_1 is denoted by $w_{i,j}$. The graph G_1 is constructed such that, among all of its subgraphs, the structure of one particular subgraph and its edge weights represent information (a message M) that is to be sent securely from a sender A to a receiver B. A secret encryption/decryption key K is shared by A and B. It is assumed that only A and B have knowledge of K.

3.2 The graph G_2

This is the first of two stages in encrypting the graph G_1 (and consequently the message M). A set of vertices and a set of weighted edges are added to G_1, resulting in a new graph G_2. The purpose of the new vertices and the new edges is to conceal the identities of the vertices and edges of G_1, as well as the values of its edge weights. This is explained in what follows.

In order to encrypt M using K, the sender augments the graph G_1 by adding to it a set of n vertices,

$$V = \{v_{n_1+1}, v_{n_1+2}, \ldots, v_{n_1+n}\},$$

and a set of m weighted edges,

$$E = \{e_{m_1+1}, e_{m_1+2}, \ldots, e_{m_1+m}\}.$$

This yields a new graph G_2 with a set of vertices $V_2 = V_1 \cup V$ containing $n_2 = n_1 + n$ vertices, and a set of edges $E_2 = E_1 \cup E$ containing $m_2 = m_1 + m$ edges.

The key K consists of two components:

1. A sequence of quadruples

   $$t_k = (i, j, w^E_{i,j}, o_{i,j}), \quad k = 1, 2, \ldots, m,$$

   where

   (a) i and j are the indices of two vertices v_i and v_j, respectively, such that both v_i and v_j belong to V_2, and (v_i, v_j) is a new undirected edge, member of the set E, to be added to G_1 in order to obtain G_2,

   (b) $w^E_{i,j}$ is the weight of the new edge (v_i, v_j), and

   (c) $o_{i,j}$ is either equal to 0 or 1, representing addition or multiplication, respectively. The value of $o_{i,j}$ is the same for all quadruples with the same i and j. The operation $o_{i,j}$ is used in the penultimate step of encryption as explained in Section 3.3.

2. A random one-to-one mapping π, whose purpose is to hide the identities of the vertices from a malicious eavesdropper. This function is used in the final step of encryption as described in Section 3.3.

3.2.1 About the encryption/decryption key

The key K is used only once. For each message M, a new encryption/decryption key is generated in tandem by A and B using an agreed-upon uniform random-number generator and an agreed-upon seed. The seed for each new key could be an agreed-upon datum from the morning's newspaper.

The common encryption/decryption key is created by the sender and the receiver synchronously but consecutively. It is first produced by A when initiating the process of encrypting G_1. The same key is later produced by B upon receipt of the encrypted graph.

The process of creating K begins by generating the following quantities m times (each iteration produces one of the m quadruples):

1. Two positive integers i and j, $i \neq j$, from the set $\{1, 2, \ldots, n_2\}$ (note that the same pair (i, j) may be generated by the random number generator for another quadruple, during another iteration of this step, as called for by the algorithm),
2. A positive integer $w^E_{i,j}$, and
3. A 0 or a 1 for $o_{i,j}$ (if the present pair (i, j) had already been generated for another quadruple, then $o_{i,j}$ takes the same value, 0 or 1, assigned to $o_{i,j}$ in that previous quadruple).

The second and final step in creating K is to generate a set of random positive integers $U = \{u_1, u_2, \ldots, u_{n_2}\}$ for use in the mapping π.

Note that if A and B had never met before commencing to communicate and exchange encrypted messages with the help of a secret key, then a public-key cryptosystem [13], or even (for increased security) a quantum cryptography protocol [19], can be used initially to establish *once and for all* all the agreed-upon parameters (namely, the random number generator, the method for generating seeds, and so on for all variables of the encryption algorithm).

3.2.2 The multigraph

The addition of the n vertices V aims to hide the vertices of V_1 in a larger set V_2. Adding the m edges E is designed to yield a *multigraph* G_2, that is, a graph in which two vertices may be connected by multiple edges; in fact, by adding m edges to G_1, every pair of vertices in the resulting graph G_2 is *intended* to be connected by several edges. In order to achieve these two objectives, we take n and m to be sufficiently large, but typically only a polynomial in n_1; for example, $n = (\alpha n_1) + \beta$ and $m = (\gamma n_2(n_2 - 1)/2) + \delta$, where α, β, γ, and δ are agreed-upon positive integers.

The weights of the edges added by K to G_1 to create G_2 are arbitrary (they are generated by a random process) but of the same type and size as the original weights in G_1.

3.3 The graph G_3

Once multigraph G_2 is constructed, the second stage of encryption begins. A new simple graph G_3 is obtained from G_2 as follows. Every pair of vertices (v_i, v_j) in G_2 are now connected in G_3 by *one* edge whose weight is either the sum (if $o_{i,j} = 0$) or the product (if $o_{i,j} = 1$) of the weights of all the edges connecting these two vertices in the multigraph G_2. In other words, all the edges between v_i and v_j in G_2 are collapsed into exactly one edge in G_3, and the weight of that edge, denoted by $W_{i,j}$, encapsulates the collective weights of all the edges it has now replaced. Note that graph G_3 is a simple, undirected, and complete graph, that is, every pair of its vertices are connected by exactly one edge. Its set of vertices is V_3, where

$$V_3 = V_2 = \{v_1, v_2, \ldots, v_{n_1}, v_{n_1+1}, v_{n_1+2}, \ldots, v_{n_1+n}\},$$

that is, G_3 has $n_3 = n_2 = n_1 + n$ vertices, and its set of edges E_3 consists of $m_3 = n_2(n_2 - 1)/2$ edges.

The final step in encrypting G_3 is to disguise its vertices. This is done using the one-to-one function π which maps every index i, $1 \leq i \leq n_3$, of a vertex in V_3, to a distinct element in the set of random positive integers U. (It is worth observing that by obscuring the identity of each vertex v_i, we are also hiding the adjacency list of v_i, that is, the identities of v_i's immediate neighbors in G_1. This property of the algorithm gains even more relevance in those cases where the assumption made throughout this paper–that G_1 is a complete graph–does not hold.) Since the purpose of the mapping π is to confuse the eavesdropper, and not the reader of this paper, we shall henceforth continue to refer to the vertices of G_3 with their original indices, namely, $v_1, v_2, \ldots, v_{n_3}$ (as they are known to A and eventually recognized by B). The graph G_3 is now sent to the receiver B.

We also note in passing that, for simplicity, we have assigned to the variable $o_{i,j}$ only two interpretations, these being addition and multiplication. More generally, $o_{i,j}$ can denote any number of arithmetic transformations when the edges of G_2 connecting every pair of vertices (v_i, v_j) are collapsed to one edge in G_3. Specifically, when $o_{i,j} = 2$, for example, $W_{i,j}$ for (v_i, v_j) is to be computed from

$$W_{i,j} = W^0_{i,j} + (W^1_{i,j} \times w_{i,j}), \text{ if } v_i \in V_1 \text{ and } v_j \in V_1,$$

$$\text{and } W_{i,j} = W^0_{i,j} + W^1_{i,j}, \text{ otherwise},$$

where $W^0_{i,j}$ is the sum of the weights of the edges in E connecting the two vertices v_i and v_j, that is,

$$W^0_{i,j} = \sum_{(v_i,v_j)} w^E_{i,j},$$

and $W^1_{i,j}$ is the product of the weights of the edges in E connecting the two vertices v_i and v_j, that is,

$$W^1_{i,j} = \prod_{(v_i,v_j)} w^E_{i,j}.$$

More advanced transformations, including modular arithmetic, for example, are also possible, but require a more involved process for creating the encryption/decryption key. This is particularly true given the present context of one-time key encryption. In the next section we show how the receiver obtains the original message M from G_3.

3.4 Decryption

When G_3 is received by B, the latter begins by deriving the value of n_1 from $n_3 = n_2 = n_1 + n = n_1 + (\alpha n_1) + \beta$, and the value of m from $m = (\gamma n_2(n_2 - 1)/2) + \delta$. The receiver can now generate the encryption/decryption key K. Then B proceeds to recover G_1 from G_3 by applying the following steps, guided by K:

1. The mapping π restores to the vertices their initial identities. All new vertices added by K, and their associated edges, are discarded from G_3. The vertices remaining are the n_1 vertices of G_1, namely, $V_1 = \{v_1, v_2, \ldots, v_{n_1}\}$.
2. The original weight of the original edge connecting a pair of vertices (v_i, v_j) in G_1 is recovered by computing
   (a) $w_{i,j} = W_{i,j} - W^0_{i,j}$, if $o_{i,j} = 0$, or
   (b) $w_{i,j} = W_{i,j}/W^1_{i,j}$, if $o_{i,j} = 1$.

Once G_1 is recovered, a weighted subgraph of it is obtained using an agreed-upon graph algorithm, whose running time is polynomial in n_1. This could be, for example, an algorithm for computing the minimum spanning tree of G_1, or the shortest path between two vertices in G_1, and so on. The resulting weighted subgraph is guaranteed to be unique by construction of G_1. It carries the information (the message M) that the sender A intends to communicate to the receiver B. The exact nature of the message M is of secondary interest in this paper.

4 Cryptanalysis

It is straightforward to see that the computations involved in both the encryption and decryption steps of Section 3 require a running time of $O(n_1^2)$, that is, a polynomial in the size of the input graph G_1. In this section we analyze the computational complexity of the task faced by the cryptanalyst (also referred to as the malicious eavesdropper, or "the enemy") in attempting to obtain the message M from the graph G_3 without knowledge of the key K.

In the real world, cryptanalysts often have at their disposal some domain-dependent information about the content of an encrypted message. This information may help them, on occasion, to extract part, if not all, of the plaintext from the ciphertext. For example, it would be of great assistance to the enemy to learn that every private communication between two parties always begins with the two words "TOP SECRET". In a theoretical analysis, however, such intelligence is too nebulous to quantify, too imprecise to express mathematically in a general setting. For the purpose of this study we assume, therefore, that the malicious eavesdropper, while likely to be familiar with the context of the communication, does not possess any side knowledge when attempting to obtain the exact plaintext message M from the ciphertext graph G_3.

Since K is never used more than once, and the availability of ancillary information is precluded, exhaustive search appears to be the only option available to the cryptanalyst. The latter can reasonably assume that the original message is hidden in (possibly a subgraph of) the plaintext graph (G_1), which may itself be a subgraph of the ciphertext graph (G_3). The only option then is to enumerate all (not necessarily complete) subgraphs of G_3, and from each subgraph, considered a candidate for being G_1, attempt to pry out a meaningful message. Since every subgraph potentially holds a message which is valid in some sense, testing a few subgraphs at random will not do. All subgraphs must be examined; none can be overlooked, none can be missed, for it may contain the intended message M. Only when all such messages have been generated, can one be selected which, when compared to all other messages considered, is without any doubt the correct M.

Enumerating all possible subgraphs of G_3, that is,

$$\sum_{x=1}^{m_3} \binom{m_3}{x} = 2^{m_3} - 1 = 2^{n_3(n_3-1)/2} - 1$$

subgraphs, is a computation requiring exponential time in the size of the input. To this must be added the time taken to generate a message from each subgraph enumerated. We do not attempt an analysis of the computational complexity of this step which is very much dependent on the particular application.

Based on this analysis, we conjecture that the time complexity of obtaining M from G_3 without knowledge of K, that is, the complexity of *inverting* the graph encryption function, is *always* exponential in the size of G_3. We also note that, by the time M is found in this way, the value to the enemy of knowing it in a timely manner would have been lost.

Formally, let f_G be the function that maps the graph G_1 to the graph G_3, under the control of the key K, as detailed in Sections 3.1–3.3; thus,

$$f_G(G_1, K) = G_3.$$

Given G_1 and K, it is computationally easy for the sender A to obtain G_3 from $f_G(G_1, K)$; as pointed out above, this computation requires polynomial time in the size of G_1. By construction, f_G is invertible. Given a graph G_3, inverting f_G means finding a graph G_1 such that:

$$G_1 = f_G^{-1}(G_3, K).$$

The receiver B has no difficulty, given G_3 and K, to obtain G_1 from $f_G^{-1}(G_3, K)$, a computation which is also easy, requiring polynomial time in the size of G_3, that is, polynomial time in the size of G_1. We claim that without knowledge of K, f_G is a *one-way function*, that is, evaluating $f_G^{-1}(G_3, ?)$ is computationally infeasible for large values of n_3 (the question mark symbol indicating absence of knowledge of K). Specifically, we conjecture that computing $f_G^{-1}(G_3, ?)$ *always* requires *exponential time* in the size of G_3. If this claim is true, it would follow that f_G is a *trapdoor one-way function*, the trapdoor here being K.

5 Applications

The encryption algorithm described in Section 3 would be useful in the encryption of the following graphs:

1. Geographic maps,
2. Communications networks,
3. Transportation infrastructures,
4. Industrial designs (e.g. integrated circuits),
5. Architectural plans,
6. Geometric constructs (e.g. Voronoi diagrams),

7. Organizational charts,
8. Information systems,
9. Processes in scientific domains (biology, chemistry, physics),
10. Text messages,

and so on, in any application where a graph is used to model an object, a concept, a real-life situation, or a relation among various entities.

For most of the applications listed here, the input (plaintext) graph G_1 may be quite large. As shown in Section 6.1, the size of G_1 somewhat grows even further when encrypted as G_3. When several ciphertext graphs are to be transmitted, the heavy traffic coupled with the data overhead may cause the communication network to become congested. This issue needs to be taken into consideration, and the parameters of the algorithm in Section 3 must therefore be selected with care.

In the following section we discuss the case in which the encrypted graph G_3 need not be transmitted, thus mitigating the problems associated with graph size and network traffic.

5.1 Graphs in databases

It is also interesting to note that the encryption algorithm of Section 3 could be used, if so needed, in the context of the database application mentioned in Section 1. This application would, of course, violate the one-time key property of the algorithm in Section 3, since, in this case, the encryption/decryption key remains valid for long periods of time, and is used repeatedly for decryption. Furthermore, the data in such an application would not possess the time-sensitive nature, a crucial characteristic in Section 3 of the data carried by the graph to be encrypted. All the same, we include this option here, as detailed in the next few paragraphs, in the interest of completeness. This will serve, as well, to illustrate the versatility of the basic idea.

Suppose then that graph G_1 is stored in the cloud, encrypted as G_3. In this case, some queries can be performed by legitimate users on the encrypted data without decrypting them. Only when the reply to the query is received, does the legitimate user who knows the encryption/decryption key K obtain the plaintext. Examples of such queries include straightforward ones, such as “What is the weight of the edge (v_i, v_j)?”, as well as more complex ones such as “Find the weight of a simple path between v_i and v_j that goes through a given set of vertices”, and “Find the weight of a spanning tree (or that of an Euler tour, or a Hamilton cycle) over a given set of vertices”. Whether the encrypted weight of one edge is returned (as in the first query), or a sequence of edges and their encrypted weights are returned (as in the second and third queries), the true weights are obtained using K. Similarly, queries that involve finding neighborhoods or connected entities, as in social networks, can easily be handled in the same way.

Certain database queries cannot be handled by the system as described. These include optimization queries, such as “What is the *shortest* path between v_i and v_j that goes through a given set of vertices”, or “Find the *minimum* spanning tree over a given set of vertices”. If answers to such queries are to be sought, then care must be given at the outset, during the encryption stage, to the selection of $w^E_{i,j}$ and $o_{i,j}$. Thus, for example, we can deliberately set $o_{i,j} = 1$ (that is, multiplication) for all $1 \leq i \leq n_2$ and $1 \leq j \leq n_2$, and ensure that $W^1_{i,j}$ has the same value for all $1 \leq i \leq n_2$ and $1 \leq j \leq n_2$. This allows the sum of two encrypted edge weights to be equal to the encryption of the sum of the two original weights; thus,

$$(W^1_{i,j} \times w_{i,j}) + (W^1_{j,k} \times w_{j,k}) = W^1_{i,j} \times (w_{i,j} + w_{j,k}).$$

Of course, an enemy would also know that calculating the shortest path in the encrypted graph reveals a shortest path in the plaintext graph. However, the enemy will not know the true total weight of the shortest path, nor the true identity of the (unencrypted) vertices on such path. In some circumstances, a typical time-storage tradeoff can be achieved by computing and storing in the untrusted database a *distance matrix* for the graph G_1, in which entry (v_i, v_j) holds, in encrypted form, the total weight of the shortest path between v_i and v_j (and, if necessary, the intermediate vertices along this path, if any, also in encrypted form).

Finally, we note that typical database operations, such as insert, delete, and update, can be executed without forcing a complete re-encryption of the database.

6 Discussion

It is said that cryptography is the process of applying *confusion* and *diffusion* to a plaintext in order to obtain a corresponding ciphertext. In a classical encryption scheme, confusion is implemented by *substitution*, that is, by using an encryption key to replace basic constituents of the plaintext (such as letters, symbols, bits, and so on) by other objects of the same or another type, and then diffusion is implemented by *permutation*, that is, by shuffling these objects, also under the control of the encryption key.

In the algorithm proposed in Section 3 to encrypt a graph, diffusion is achieved by adding new vertices and weighted edges to the original graph G_1, thus obtaining the graph G_2. Confusion is achieved by replacing all the edges connecting two vertices in G_2 with one edge whose weight combines the weights of the edges it replaces, thus obtaining the graph G_3. Confusion is also achieved by renaming the vertices of G_3 before sending it to B.

In the remainder of this section we discuss a possible implementation of the graph G_3 and examine alternative algorithms for encrypting a graph.

6.1 Implementation

The encryption algorithm of Section 3 does not specify in what form the graph G_3 is transmitted to the receiver. We can assume that A sends G_3 to B as a *data structure*. For example, G_3 can be organized as a two-dimensional array whose rows and columns are labeled with the vertices in V_3. Because the edges in E_3 are undirected, only a triangular array is needed, having $n_3 - 1$ rows labeled

$$v_2, v_3, \ldots, v_{n_1}, v_{n_1+1}, \ldots, v_{n_1+n},$$

and $n_3 - 1$ columns labeled

$$v_1, v_2, \ldots, v_{n_1}, v_{n_1+1}, \ldots, v_{n_1+n-1}.$$

The entry in position (v_i, v_j), $i > j$, of the triangular array, is the weight $W_{i,j}$ of the edge (v_i, v_j). The important point here is that, while G_3 is sent in structured form, it does not reveal anything about the structure or contents of G_1 to anyone who does not know the key K.

Since G_1 is a *complete* graph (besides being simple, undirected, and weighted), no other data structure for its implementation is more efficient than the triangular array just described in the previous paragraph. We use this data structure in the remainder of this section as we explore alternative algorithms for encrypting the input graph G_1.

Finally, we note that, while the encrypted graph G_3 is larger than its original version G_1, the sizes of both graphs differ by a (relatively small) multiplicative constant. To wit, G_1 and G_3 have n_1 and $O(n_1)$ vertices, respectively. Similarly, both G_1 and G_3 have $O(n_1^2)$ edges.

6.2 Alternative graph encryption algorithms

How does the algorithm of Section 3 differ from other possible approaches for encrypting a graph G_1? In what follows we consider several such alternatives in the context of the application studied in this paper, namely, that the graph travels from A to B, encrypted using a one-time key. The latter is generated separately by A and B, when needed, by means of a random-number generator, as described in Section 3.2.1. It is to be known exclusively by A and B and (by definition) is never to be used again to encrypt another message.

6.2.1 Encrypting the edges of each vertex separately

In the first approach, the one-time secret encryption/decryption key shared by the sender and the receiver is a set of n_1 coefficient matrices $\{X_1, X_2, \ldots, X_{n_1}\}$, each with $n_1 - 1$ rows and $n_1 - 1$ columns, each of which is associated with a distinct vertex of G_1. Further, each of these matrices is non-singular and its entries are all positive numbers.

Encryption proceeds as follows. For every vertex v_i of G_1, the weights of the $n_1 - 1$ edges connecting v_i to its $n_1 - 1$ neighbors, and represented by the vector

$$Y_i =< w_{i,1}, w_{i,2}, \ldots, w_{i,i-1}, w_{i,i+1}, \ldots, w_{i,n_1} >,$$

are encrypted using the $(n_1 - 1) \times (n_1 - 1)$ coefficient matrix X_i. Thus, A computes the vector

$$X_i \times Y_i^T = Z_i^T,$$

and sends Z_i^T to B.

Upon receipt of Z_i^T, B who knows X_i, obtains Y_i from

$$X_i^{-1} \times Z_i^T = Y_i^T,$$

or, equivalently, by solving $n_1 - 1$ equations in the $n_1 - 1$ unknowns $w_{i,1}, w_{i,2}, \ldots, w_{i,i-1}, w_{i,i+1}, \ldots, w_{i,n_1}$,

$$X_i \times Y_i^T = Z_i^T.$$

The difficulty with this approach is that it reveals too much about the structure of G_1 to an eavesdropper. Also, each edge weight is encrypted twice and decrypted twice. Nonetheless, by using n_1 distinct coefficient matrices, each to encrypt the weights associated with one of the n_1 vertices, this approach has a better chance to withstand cryptanalysis than the following simple variant.

6.2.2 Using a single coefficient matrix

Let R_1 denote the *weight matrix* of graph G_1, that is, the matrix whose entry at row v_i and column v_j is the weight $w_{i,j}$ of the edge (v_i, v_j) connecting the two vertices v_i and v_j. In the encryption algorithm we consider in this section, a *single* $n_1 \times n_1$ coefficient matrix Q_1 is used to encrypt the entire weight matrix R_1 of the graph G_1, by computing

$$Q_1 \times R_1 = S_1.$$

Here, Q_1 is non-singular and all of its entries are positive numbers. Now S_1 is transmitted to the receiver. The latter, who knows the one-time key Q_1, recovers the weight matrix R_1 from the obvious equation

$$R_1 = Q_1^{-1} \times S_1,$$

or equivalently by solving $n_1(n_1 - 1)/2$ equations in $n_1(n_1 - 1)/2$ unknowns $w_{i,j}$, $i = 2, 3, \ldots, n_1$ and $j = 1, 2, \ldots, n_1 - 1$,

$$Q_1 \times R_1 = S_1,$$

where the unknown weight matrix R_1 is symmetric and $w_{i,i} = 0$, for $i = 1, 2, \ldots, n_1$.

Note that if R_1 is represented as a triangular array, as described in Section 6.1, then so are Q_1, S_1, and Q_1^{-1}.

This algorithm is less secure than the one in Section 6.2.1, as the same coefficient in Q_1 is used to encrypt several edge weights in R_1.

6.2.3 Matrix confusion and diffusion

The algorithm we examine in this section for encrypting the graph G_1 uses a one-time key L, which is a triangular array with $n_1 - 1$ rows labeled $2, 3, \ldots, n_1$, and $n_1 - 1$ columns labeled $1, 2, \ldots, n_1 - 1$. The entry in position (i, j), $i > j$, of L holds the following values:

1. The first value $e_{i,j}$ is either a 0 or a 1,
2. The second and third are two positive integers $a_{i,j}$ and $b_{i,j}$, respectively.

Key L encrypts G_1's weight matrix R_1, stored as a triangular array. The result is an $n_1 \times n_1$ matrix D with entries $d_{i,j}$, $1 \leq i, j \leq n_1$. Let $c_{i,j}$, $1 \leq i, j \leq n_1$, be a random positive integer, of the same magnitude as $a_{i,j} w_{i,j} + b_{i,j}$, $1 \leq j < i \leq n_1$. Encryption proceeds as follows, for $i > j$:

1. If $e_{i,j} = 0$ then $d_{i,j} = a_{i,j} w_{i,j} + b_{i,j}$ and $d_{j,i} = c_{j,i}$,
2. If $e_{i,j} = 1$ then $d_{i,j} = c_{i,j}$ and $d_{j,i} = a_{i,j} w_{i,j} + b_{i,j}$.

Finally, for $i = j$, $d_{i,i} = c_{i,i}$.

In other words, the $n_1(n_1 - 1)/2$ elements of R_1 are stored in D, encrypted using $a_{i,j}$ and $b_{i,j}$, and scattered using $e_{i,j}$, with $c_{i,j}$ creating further confusion. Having computed D, the sender expedites it to the receiver. The latter, who knows L, ignores all the $c_{i,j}$ entries, and decrypts the relevant entries of D.

The algorithm in this section aims to bring together the advantages of the algorithms presented in Section 6.2.1 and Section 6.2.2, by offering a combination of simplicity and security. Like other algorithms in this discussion, however, it provides the eavesdropper with a glimpse into the structure of G_1.

6.2.4 Encrypting at the binary level

In its most basic digital form, the graph G_1 (that is, its vertices, its edges, and its edge weights), is seen as a string M_1 consisting only of 0s and 1s, whose length is denoted by N_1. The string M_1 is obtained by concatenating the rows of the weight matrix R_1 of G_1 into a one-dimensional array, and expressing its entries in binary notation. For this representation of G_1, encryption will employ a one-time key K_1, also of length N_1 bits. The graph G_1 is encrypted by computing the bit-wise Exclusive-OR of M_1 and K_1,

$$C_1 = M_1 \oplus K_1,$$

which is sent to B. The latter recovers M_1 from

$$M_1 = C_1 \oplus K_1.$$

In other words, B has no difficulty in recovering G_1 (in binary notation!). The problem facing B is that the string M_1 presents the graph G_1 in an entirely *unstructured* form. The receiver needs to make sense of a string M_1 of N_1 bits, and derive from it the graph structure of G_1. This necessarily means that A must somehow communicate some information to B, relating to the number of vertices, number of edges, and nature of the edge weights of the graph represented by M_1. Whether this information is sent in encrypted form separately from C_1, or it is included in M_1 and is sent as part of the ciphertext C_1, the trick is to avoid introducing a weakness that a malicious eavesdropper might be able to exploit profitably over time and over a succession of distinct graphs G_1 sent from A to B.

We also note that this algorithm is not adaptable to the application in Section 5.1, in particular when certain optimization operations, such as computing a minimum spanning tree or a shortest path, are to be performed in the cloud. These computations will not be possible because the sum of two encrypted edge weights, is not necessarily equal to the encryption of the sum of the two original weights:

$$(w_{i,j} \oplus K_1) + (w_{j,k} \oplus K_1) \neq (w_{i,j} + w_{j,k}) \oplus K_1.$$

6.2.5 The spider web

In this final variant of the algorithm of Section 3, we return to some of the ideas used in that algorithm, and apply them with a twist. The sender A obtains a new graph G' from the input graph G_1 with the help of an encryption/decryption key K', and sends it to the receiver B. The steps for creating G' are detailed in what follows.

Step 1: For each edge (v_i, v_j) connecting the two vertices v_i and v_j, where $i > j$ and $1 \leq i, j \leq n_1$, in G_1, a set $V_{i,j}$ of ℓ new vertices,

$$V_{i,j} = \{v_{i,j}^1, v_{i,j}^2, \ldots, v_{i,j}^\ell\},$$

is inserted on (v_i, v_j) between v_i and v_j, thus splitting (v_i, v_j) arbitrarily into a set $E_{i,j}$ of $\ell+1$ segments, each of which is now a new edge,

$$E_{i,j} = \{(v_i, v_{i,j}^1), (v_{i,j}^1, v_{i,j}^2), \ldots, (v_{i,j}^\ell, v_j)\}.$$

These edges replace the original edge (v_i, v_j). Their respective weights,

$$w_{i,j}^{(1)}, w_{i,j}^{(2)}, \ldots, w_{i,j}^{(\ell+1)},$$

add up to $w_{i,j}$ the weight of the original edge (v_i, v_j). The edges created in this step form the set

$$E_1' = \bigcup_{i>j} E_{i,j}, 1 \leq i, j \leq n_1.$$

Given that G_1 has $n_1(n_1 - 1)/2$ edges, the total number of vertices added is $\ell n_1(n_1 - 1)/2$, forming the set,

$$V_1' = \bigcup_{i>j} V_{i,j}, 1 \leq i, j \leq n_1.$$

The graph G' therefore has $n' = n_1 + \ell n_1(n_1 - 1)/2$ vertices in the set $V' = V_1 \cup V_1'$.

Step 2: Each vertex in V' is now connected to all other vertices with which it does not already share an edge; let this new set of edges thus introduced be E''. This yields the set $E' = E_1' \cup E''$ of edges of G' of size $n'(n' - 1)/2$.

Step 3: The weight $w_{i,j}^{(p)}$, for $i > j$, $1 \leq i, j \leq n_1$, and $1 \leq p \leq \ell + 1$, of each edge added in Step 1, that is, the weight of each edge in E_1', is encrypted as ${w'}_{i,j}^{(p)}$ by means of two numbers $a_{i,j}^{(p)}$ and $b_{i,j}^{(p)}$; thus,

$${w'}_{i,j}^{(p)} = a_{i,j}^{(p)} w_{i,j}^{(p)} + b_{i,j}^{(p)}.$$

As well, all edges created in Step 2, that is, the edges in E'', are assigned random weights.

Step 4: The final step in encrypting G_1 is to use a mapping π' (similar to the mapping π of Section 3.3) in order to disguise the identities of the vertices in V'.

The encryption/decryption key K' consists of a sequence of pairs $(a_{i,j}^{(p)}, b_{i,j}^{(p)})$, where $i > j$, $1 \leq i, j \leq n_1$, and $1 \leq p \leq \ell + 1$, and the mapping π'. Note also that ℓ is an initially agreed-upon parameter of this algorithm. This allows B upon receiving G' to recover n_1 from $n' = n_1 + \ell n_1(n_1 - 1)/2$. The receiver now generates K' (as described in Section 3.2.1) and proceeds to identify the original vertices and the weights of the original edges. This algorithm is simpler than the algorithm of Section 3. However, unlike the algorithm of Section 3, its resistance to a potential cryptanalytic threat is generally more difficult to analyze.

7 Conclusion

The problem addressed in this paper is that of encrypting a graph to be transmitted privately from a sender A to a receiver B. The two communicating parties are assumed to share an encryption/decryption key to be used only once. The main algorithm described in Section 3 and its variants discussed in Section 6.2 are simple, efficient, and (one hopes) resistant to cryptanalysis. They also enjoy the property of being easily modified, if so required; the basic idea of each encryption algorithm can be readily extended for efficiency or security purposes.

The cryptosystems of Sections 3 and 6.2 protect A and B from a passive eavesdropper, that is, one who simply listens to their communications. They can also safeguard against an active eavesdropper, that is, one who injects spurious data into a message. An example of this attack is provided by the algorithm of Section 6.2.4. When encryption is at the binary level, an active eavesdropper can change the ciphertext C_1 to another ciphertext C_2, so that the plaintext message M_2 obtained by B is different from, but very closely related to, the message M_1 sent by A. Thus, with knowledge that

$$C_1 = M_1 \oplus K_1,$$

but without any knowledge of M_1, the eavesdropper can create an encryption of a message M_2, where

$$M_2 = M_1 \oplus P_1,$$

for any binary string P_1, by intercepting C_1, computing

$$C_2 = C_1 \oplus P_1 = (M_1 \oplus K_1) \oplus P_1 = (M_1 \oplus P_1) \oplus K_1 = M_2 \oplus K_1,$$

and sending C_2 to B. This property, known as *malleability*, is a common weakness plaguing almost all classical cryptosystems, with rare exceptions [7]. There are ways to detect such intrusion. Most cryptosystems, including all the ones in this paper, possess the ability to incorporate a digital signature [1], thereby allowing the sender of a message, as well as the message itself, to be authenticated.

This research began as an attempt to solve an open problem in the theory of computation: To find a function that is *provably* one-way. The exploration led to graph theory, and a search for a problem that forces any solution to necessarily require the enumeration of all subgraphs of a given graph. From there, it was only a small step to cryptography, and the potential discovery of a trapdoor one-way function, computing the inverse of which is hypothesized to have a complexity that is exponential in the size of the input.

Of course, this is not the first time that functions that are believed to be one-way are used in cryptography. In fact, the security of most, if not all, modern classical cryptosystems rests upon the unproven assumption that finding the inverse of the functions on which encryption is based, is computationally intractable [13].

Time will tell whether these conjectures are true. Time will also tell whether unconventional platforms may some day be used in the representation and secure transmission of graphs [18]. For example, molecules can be considered graphs. Specifically, proteins have a three-dimensional geometric structure, which is held together by connections between the atoms in the molecule. These connections are either electric or covalent and they vary in strength. Thus, the force that binds two atoms is the weight of the graph edge between these two atoms. In addition, protein folding is controlled by restrictions on the angles between edges in the graph, and these restrictions could express certain graph characteristics. Proteins also have the ability to add or delete edges. These observations suggest that hiding a secret message inside a protein is perhaps a future possibility. This way, we would have come full circle: Graphs, traditionally used as representations of physical entities, could ultimately be embodied by these physical entities themselves.

Acknowledgments

I am grateful to Pat Martin, Marius Nagy, Naya Nagy, and Kai Salomaa for their helpful comments.

References

[1] Akl, S.G., Digital signatures: A tutorial survey, *IEEE Computer*, Vol. 16, No. 2, 1983, pp. 15–24.

[2] Al Etaiwi, W.M., Encryption algorithm using graph theory, *Journal of Scientific Research & Reports*, Vol. 19, No. 3, 2014, pp. 2519–2527.

[3] Brickell, J. and Shmatikov, V., Privacy-preserving graph algorithms in the semi-honest model, in: *ASIACRYPT 2005*, Roy, B. (ed.), LNCS 3788, 2005, pp. 236–252.

[4] Cao, N., Yang, Z., Wang, C., Ren, K., and Lou, W., Privacy-preserving query over encrypted graph-structured data in cloud computing, *Thirty-First IEEE International Conference on Distributed Computing Systems*, 2011, pp. 393–402.

[5] Cao, Z. and Liu, L., On the weakness of fully homomorphic encryption, arXiv:1511.05341 [cs.CR]

[6] Charles, D.X., Lauter, K.E., and Goren, E.Z., Cryptographic hash functions from expander graphs, *Journal of Cryptology*, Vol. 22, No. 1, 2009, pp. 93–113.

[7] Dolev, D., Dwork, C., and Naor, M., Nonmalleable cryptography, *SIAM Journal of Computing*, Vol. 30, No. 2, 2000, pp. 391–437.

[8] El Makkaoui, K., Ezzati, A., and Beni Hassane, A., Challenges of using homomorphic encryption to secure cloud computing, *Proceedings of the International Conference on Cloud Technologies and Applications*, 2015, pp. 1–7.

[9] Frederick, R., Core concept: Homomorphic encryption, *Proceedings of the National Academy of Science*, Vol. 112, No. 28, 2015, pp. 8515–8516.

[10] Gentry, C., Computing arbitrary functions of encrypted data, *Communications of the ACM*, Vol. 53, No. 3, 2010, pp. 97–105

[11] Gerbracht, S., Possibilities to encrypt an RDF-graph, *IEEE Third International Conference on Information and Communication Technologies: From Theory to Applications*, 2008, pp. 1–6.

[12] Kasten, A., Scherp, A., Armknechet, F., and Krause, M., Towards search on encrypted graph data, *Proceedings of the International Conference on Society, Privacy and the Semantic Web*, 2013, pp. 46–57.

[13] Katz, J. and Lindell, Y., *Introduction to Modern Cryptography* (2nd edition), CRC Press, Boca Raton, 2015.

[14] Kinoshita, H., An image digital signature system with ZKIP for the graph isomorphism, *Proceedings of the IEEE International Conference on Image Processing*, 1996, pp. 247–250.

[15] Lauter, K. and Naehrig, M., Can homomorphic encryption be practical? *Proceedings of the Third ACM Cloud Computing Security Workshop*, New York, 2011, pp. 113–124.

[16] Maricq, A., Applications of Expander Graphs in Cryptography.
`https://www.whitman.edu/Documents/Academics/Mathematics/2014/maricqaj.pdf`

[17] Meng, X., Kamara, S., Nissim, K., and Kollios, G., GRECS: Graph encryption for approximate shortest distance queries, *Proceedings of the Twenty-Second ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 504–517.

[18] Nagy, N., Personal communication, May 2018.

[19] Nagy, N., Nagy, M., and Akl, S.G., Carving secret messages out of public information, *Journal of Computer Science*, Vol. 11, No. 1, 2015, pp. 64–70.

[20] Nguyrn, P.Q., Breaking fully-homomorphic-encryption challenges, in: *CANS 2011*, Lin, D. et al. (Eds.), LNCS 7092, 2011, pp. 13–14.

[21] Petit, C., On graph-based cryptographic hash functions, Thèse soutenue en vue de l'obtention du grade de Docteur en Sciences Appliquées, Université Catholique de Louvain, Louvain-la-Neuve, Belgique, May 2009.

[22] Polak, M., Romańczuk, U., Ustimenko, V., and Wróblewska, A., On the applications of extremal graph theory to coding theory and cryptography, *Electronic Notes in Discrete Mathematics*, Vol. 43, 2013, pp. 329–342.

[23] Priyadarsini, P.L.K., A survey on some applications of graph theory in cryptography, *Journal of Discrete Mathematical Sciences and Cryptography*, Vol. 18, No. 3, 2015, pp. 209–217.

[24] Samid, G., Denial cryptography based on graph theory, U.S. Patent 6823068 B1, 2004.

[25] Sharma, S., Powers, J., and Chen, K., Privacy-preserving spectral analysis of large graphs in public clouds, *Proceedings of the 11th ACM Asia Conference on Computer and Communications Security*, 2016, pp. 71–82.

[26] Szöllösi, L., Marosits, T., Fehér, G., and Recski, A., Fast digital signature algorithm based on subgraph isomorphism, in: *CANS 2007*, Bao, F. et al. (Eds.), LNCS 4856, 2007, pp. 34–46.

[27] Ustimenko, V.A., Graphs with special arcs and cryptography, *Acta Applicandae Mathematicae*, Vol. 74, 2002, pp. 117–153.

[28] Ustimenko, V.A., On graph-based cryptography and symbolic computations, *Serdica Journal of Computing*, Vol. 1, 2007, pp. 131–156.

[29] Ustimenko, V.A. and Khmelevsky, Y., Walks on graphs as symmetric or asymmetric tools to encrypt data, *South Pacific Journal of Natural Sciences*, Vol. 20, 2002, pp. 34–44.

[30] Wang, Q., Ren, K., Du, M., Li, Q., and Mohaisen, A., SecGDB: Graph encryption for exact shortest distance queries with efficient updates, *Twenty-First International Conference on Financial Cryptography*, 2017, pp. 79–97.

[31] Xie, P. and Xing, E., CryptGraph: Privacy preserving graph analytics on encrypted graph. arXiv:1409.5021 [cs.CR]

[32] Yuan M., Chen, L., Yu, P.S., and Mei, H., Privacy preserving graph publication in a distributed environment, *World Wide Web*, Vol. 18, No. 5, 2015, pp. 1481–1517.