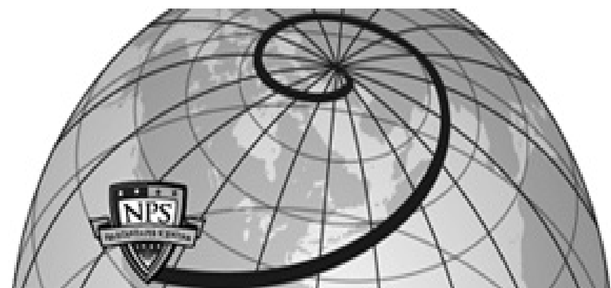
Calhoun
Institutional Archive of the Naval Postgraduate School



Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

2016-03

Granular security in a graph database

Crawford, Brian

Monterey, California: Naval Postgraduate School

https://hdl.handle.net/10945/48509

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

http://www.nps.edu/library

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

GRANULAR SECURITY IN A GRAPH DATABASE

by

Brian Crawford

March 2016

Thesis Advisor: Thomas Otani
Second Reader: Arijit Das

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	03-27-2016	Master's Thesis 01-06-2014 to 03-27-2016

4. TITLE AND SUBTITLE

GRANULAR SECURITY IN A GRAPH DATABASE

5. FUNDING NUMBERS

6. AUTHOR(S)

Brian Crawford

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

With the growing use of data in all aspects of government and commerce, the need for that data to be both accessible and secure is also growing. One solution to this dual need is provided by Accumulo, a database that allows multiple users of various security levels to access one platform but receive authorization to view only portions of the database. Various databases, however, organize information differently. This thesis examines the possibility of implementing a granular security on a graph database. Using Neo4j as a reference implementation, graph theory concepts are used to find a method of allowing data access while retaining security in a data environment that emphasizes connectivity. Using adjacency matrix multiplication on bipartite graph slices of the network of security layers, a mathematical justification exists for locating two step connections that exit from and return to a security layer. These connections can be revealed to a user without granting access outside of the assigned security layer.

14. SUBJECT TERMS

database security, graph database, granular security, bipartite graph, directed graph, multi-slice graph

15. NUMBER OF PAGES 69

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

GRANULAR SECURITY IN A GRAPH DATABASE

Brian Crawford
Lieutenant, United States Navy
B.A., King College, 2004
M.A., Lipscomb University, 2009

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2016

Author: Brian Crawford

Approved by: Thomas Otani
Thesis Advisor

Arijit Das
Second Reader

Geoffrey Xie
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

With the growing use of data in all aspects of government and commerce, the need for that data to be both accessible and secure is also growing. One solution to this dual need is provided by Accumulo, a database that allows multiple users of various security levels to access one platform but receive authorization to view only portions of the database. Various databases, however, organize information differently. This thesis examines the possibility of implementing a granular security on a graph database. Using Neo4j as a reference implementation, graph theory concepts are used to find a method of allowing data access while retaining security in a data environment that emphasizes connectivity. Using adjacency matrix multiplication on bipartite graph slices of the network of security layers, a mathematical justification exists for locating two step connections that exit from and return to a security layer. These connections can be revealed to a user without granting access outside of the assigned security layer.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Introduction **1**

2 Security Models **5**
2.1 General Models 5
2.2 Data Management 6
2.3 Granular Access. 8

3 Graph Databases **9**
3.1 Graphs and Digraphs 9
3.2 Nodes and Relationships 10
3.3 Neo4j and Cypher 12

4 Granular Security on Graph Nodes **15**
4.1 Assigning Security. 16
4.2 Relationship Types 17
4.3 Possible Relationship Security Policy 20

5 Security Exploration on Graph Edges **21**
5.1 Hypergraph 21
5.2 Weighted Graph. 22
5.3 Temporal Network. 24
5.4 Coloring. 25
5.5 Multilayer Graphs 28
5.6 Bipartite Graphs. 31

6 Granular Security Implementation **37**
6.1 Granular Security through Properties 37
6.2 Proof of Concept 37
6.3 Plugin. 38

6.4 Cypher GUI 39
6.5 Synthetic Graph Analysis 40
6.6 Real World Graph Analysis 43

7 Conclusion and Future Work **47**
7.1 Conclusions 47
7.2 Future Work 48

List of References **51**

Initial Distribution List **53**

List of Figures

Figure 2.1 Security levels diagram. 7

Figure 3.1 Basic graph example. 9

Figure 3.2 Directed graph examples. 10

Figure 3.3 Simple animal database. 11

Figure 4.1 Simple manipulation graph. 15

Figure 4.2 Query with node security. 16

Figure 4.3 Connection not visible with node security. 16

Figure 4.4 Trivial relationship. 18

Figure 4.5 Restricted relationship. 19

Figure 4.6 Induced relationship. 19

Figure 5.1 Simple manipulation graph with hyperedges. 22

Figure 5.2 Simple manipulation graph with edge weights. 23

Figure 5.3 Simple manipulation graph with node coloring. 26

Figure 5.4 Simple manipulation graph with edge coloring. 27

Figure 5.5 Simple manipulation graph with node and edge coloring. 28

Figure 5.6 Simple manipulation graph as multilayer graph. 29

Figure 5.7 Simple manipulation graph in multislice format. 31

Figure 6.1 Results from Neo4j query. 40

Figure 6.2 Unclassified layer as rendered in Neo4j. 43

Figure 7.1 Longer induced relationship examples. 49

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

DOD	Department of Defense
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
NSA	National Security Agency
OPSEC	Operations Security
SQL	Structured Query Language
URL	Uniform Resource Locator

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

Thesis writing is a long and arduous journey; however, it is not a trek one accomplishes alone. Although a few short paragraphs cannot fully express my appreciation to those who propelled me ever onward, I will, nevertheless, make the attempt. I thank my advisor, Dr. Thomas Otani, for his support and guidance through both the successes and failures of researching and writing, as well as for his exemplary classroom instruction; never before have I learned so much so quickly.

For setting me up for success, I thank Arijit Das, my second reader, who helped me acquire all the resources I would need for this endeavor. I thank Dr. Ralucca Gera for quite literally teaching me everything I know about graph theory, both in understanding and appreciating the complexity.

To my three little munchkins, Jocelyn, Zoe, and Lucy, I know you were not fans of the long hours I worked, but getting home to see you was my constant motivation to work efficiently and expeditiously. Thank you also for reminding me to always see daunting challenges as exciting new adventures.

Lastly, to my wife, Denise, without your steadfast love and unwavering patience I would not have made it. You give me the courage and fortitude to conquer all obstacles because I know that while you are not always physically by my side we always confront life's hurdles together.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Network science and graph theory are relative newcomers to the worlds of mathematics and computer science, but their emergence has highlighted that "most events and phenomena are connected, caused by, and interacting with a huge number of other pieces of a complex universal puzzle" [1]. The analysis of these connections reveals information about the world that could not have been discovered if those same pieces of information were examined in isolation.

Databases have long served as a tool where data can be both stored and linked. Starting in the mid-1980s, database designers were able to decouple the manipulation of information from the computer's storage processes [2]. This split allowed database use to become more accessible to non-expert users who did not have extensive knowledge of programming code or computer architecture. These early computer database models, called relational databases, employed Structured Query Language (SQL) to allow for more human-readable and human-writable queries.

SQL databases, however, encountered competition with the rise of spreadsheets, which included a more straightforward way to manage and manipulate small data sets, a task previously possible only through the use of a relational database [2]. The prevalence of modern "big data" has seen a resurgence of the popularity of databases. These newer databases, collectively referred to as No-SQL databases, depart from the traditional relational model.

Although they are lumped into one category, the data storage models of these No-SQL databases are highly varied. These various models allow data to be stored, searched, and analyzed in different ways. Frequently, these models also provide some advantage over a traditional relational model either in the speed with which the data can be searched or the complexity of query and analysis options.

The differences between these models can be compared to finding a specific room number at two different hotels. The searcher's approach to finding the room would be different for a high-rise in the city than it would be for a beach bungalow. The same is true with

data storage models; how the data is stored and searched varies according to the goals and designs of the data model. In particular, one type of new database model is constructed around the ideas of graph theory, which permits data to be organized in a way that leverages its inter-connectivity.

With database security, it is typically assumed that any authorized user for the database should have access to the entire data set. With the size of databases steadily increasing, this blanket access may no longer be appropriate. Securing data within a database to allow users access only to a portion of the entire data set allows large databases to grow while imposing restrictions to prevent every user from having access to the entire database. This partitioned data access, called granular security, is not widely utilized.

One database, Accumulo, successfully implements granular security control on a more common No-SQL data storage model. Based on Google's BigTable database, Accumulo adds a security control feature to the underlying design. Users of different access levels can all be permitted to access the same Accumulo database, and the database system will return information to a given user only if it matches her assigned access level.

The security restrictions of Accumulo provide a good model, but the organization of the data still relies on a design where data points, although they can be linked, are still largely isolated. Newer database designs based on a graph model, called graph databases, store information with more emphasis on how the individual data points are linked to each other. Implementing a granular security feature on a graph database would allow for finer access controls; doing so places restrictions on what data is accessible to a user and may, thus, limit the benefits that a graph database has for leveraging data connections.

The goal of this thesis is to explore the feasibility of implementing granular security controls onto a graph database without sacrificing the graph's ability to connect data. Neo4j, a current graph database software application, is used as the base template for implementing granular security on a graph data model in the same way that BigTable served as the base for the security controls of Accumulo.

In Chapter 2, the concept of granular database security is further developed. Graph structures and graph databases, specifically Neo4j, is discussed in Chapter 3. The basic elements

of implementing granular security in a graph database and a mathematical exploration of the challenges of that implementation is addressed in Chapters 4 and 5, respectively. Chapter 6 explores the security implementation on a generated graph, and Chapter 7 concludes this thesis and presents opportunities for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Security Models

Implementing granular security on a database would not happen in a vacuum. An organization or corporation desiring to exert finer controls on access within a database likely already has some sort of data security model in place. In a corporate context, these partitions may exist to keep sales, marketing, and manufacturing information separate for most employees, but managers might have access to multiple areas. In a government context, information is typically segregated into Unclassified, Secret, and Top Secret information. This chapter addresses a few generic security models and outlines the current government data-management model. It then describes the implementation of granular security on the government classification system.

2.1 General Models

In order for a granular security implementation to be successful, it must be built on a general access security policy that addresses not only what information can be read but also what information can be written. Information separation architectures can be hierarchical, compartmentalized, or some combination of the two. Hierarchical denotes security structures where one security level can be considered higher or lower than another, and someone who has access to a given level also has access to any lower levels. For example, in a government agency, a Secret-level user would, by default, have access to Unclassified information because Unclassified is a lower security level than Secret.

Compartmentalized, on the other hand, denotes security levels that reside on an equal plane; access to one compartment does not automatically grant access to another one. In a business setting, for instance, accounting and marketing departments operate independently, and access to one area does not, by default, grant access to another because the two levels exist in parallel with each other.

In a hierarchical security environment, the Bell-LaPadula model is a set of common security properties developed for the Department of Defense (DOD). The model is commonly

summarized down to two simple rules: no read up and no write down [3]. These principles allow a user to read any information at his own security level or lower and to write any information at his own security level or higher.

Other security models, such as the Biba model for access control, also exist. The Biba model, which emphasizes data integrity vice data security, is the reverse of the Bell-LaPadula model; its rules are frequently summarized: no read down and no write up [4]. Not only do these two models have differing objectives, they are also only applicable to a hierarchical security environment. The concepts of up and down do not exist in a compartmentalized security structure where security levels are divided in parallel.

2.2 Data Management

DOD information is classified into one of the three security levels from least secure, Unclassified, to most secure, Top Secret. (Confidential has been intentionally omitted both for ease of discussion with only three access levels and because no current broadly reaching information system operates strictly at that level.) Each of the levels also contains compartments that partition in parallel rather than from top to bottom, making the system both hierarchical with levels and compartmentalized within those levels. The diagram in Figure 2.1 illustrates this concept.

A security clearance could allow user access to information at all levels (Unclassified, Secret, and Top Secret), but not all the information at those levels is necessarily accessible to that user. The information at each level that the user can access depends on the compartments to which she has access. Access to the Secret level with compartments 1 and 3 will not grant a user access to compartment 2, even though it is also at the Secret level (compartment names used here are arbitrary).

Of note, in the hierarchical structure, access to any level implies access to the lower levels. A user with Secret access, by default has Unclassified access, but that user may not necessarily have Top Secret access. All users have access at the Unclassified level. Compartments do not function with the same implications; a user with access to compartment 3 does not necessarily have access to any other compartments. In addition, a user may not have access to any compartments at a given level. For clarity, the term levels will be used

when referring to a hierarchical classification system, and compartments will refer to data that is partitioned in parallel.

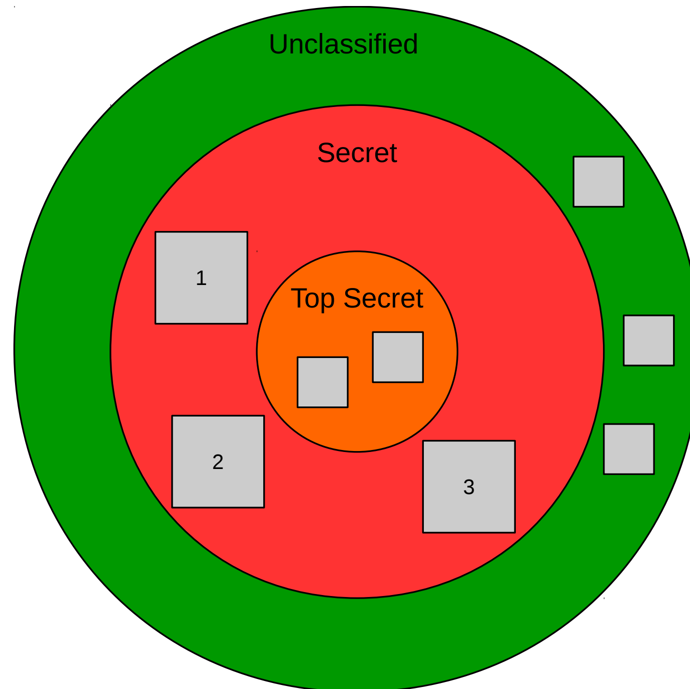


Figure 2.1: Security levels diagram. The concentric circles represent hierarchical security levels; the gray boxes represent compartments within the security levels. A Secret user with access to compartments 1 and 3 would have access to everything red and green, but only the gray boxes labeled 1 and 3 (all other gray boxes would not be accessible, even if they are in the red and green circles).

The previous description outlines the regulations for access to information according to clearance level, but the actual usage of this information may involve accessing physically separate systems. A Top Secret user likely will need to access a separate Secret and/or Unclassified system to collect all needed data, and even then, connections between the data at different levels must be inferred by the user. Having one system with a granular access control would enable Top Secret and Unclassified users to access the same information but with limitations on what information would be available to them. This composite system would also be able to link data from different levels rather than relying on the user to make those connections.

2.3 Granular Access

The government model of information systems works well for keeping information from leaking from more secure to less secure levels of access; however, the limitation of these separate systems prevents data at one level from being easily linked to data at a different level. Under the current architecture, the only way to draw connections between information at the Secret and Top Secret levels is to replicate that information onto the Top Secret system. Extending the process of connecting information on different levels for a single data point would require replicating the entire Secret system on the physical Top Secret system, for instance, as well as updating regularly since the information is not static.

If a user needs access to two separate compartments, replicating one system onto another is likely not the best solution since those with access to one compartment do not necessarily have access to both compartments. In this case, an additional physical system would need to be created to house the information from both compartments. These systems would also need to account for regular updates.

If all the information were instead stored on a single physical system and each item of data was assigned an access level (and compartment if necessary), then all users or applications could have access to the same system. The database, when queried, would return information only for the levels and compartments to which the user has access. This granular access is not only less cumbersome than physically separate systems, it also allows users, with the proper access levels, to connect information points where a connection was previously not possible due to the physically separate systems.

In order to explore granular security in a way that is more broadly applicable, the security model utilized in the present work will be simplified. In the illustrations, users will be assumed to have authorization to read and write only at their own access level. Generic, compartmentalized (rather than hierarchical) security levels of X, Y, and Z will be used. This more restrictive model removes ambiguity and confusion that can be caused by more complicated models. In addition, this simplified model can apply to either hierarchical or compartmentalized security environments that emphasize either data security or integrity.

CHAPTER 3:
Graph Databases

Relational databases, the most widely used database architecture and historically the only widely available database architecture option, are constructed as collections of cross-referenced tables. These databases are searched, typically using SQL, for information usually collating data from multiple tables into the query output. This storage structure has functioned well for several decades; however, with the prevalence of increasingly larger and more complex data sets, these relational databases have not been able to scale up. The time required for some types of SQL queries to traverse a large, modern database does not keep pace with the speed of other applications which depend on the results of these queries, causing a choke point in the processing time for modern applications.

This SQL choke point has led developers to explore alternate means of storing and searching large data sets. Due to the historical prevalence of SQL in relational databases, these new architectures are collectively referred to as non-SQL or NoSQL databases. Classifying these new, disparate types of databases into categories is not simple or consistent. Some common categories include tabular, key-value store, column, document store, and graph databases.

3.1 Graphs and Digraphs

Graph databases find their roots in the concepts of graph theory; a graph is a group of edges and vertices where a vertex, or node, can exist independently, but an edge must connect two vertices. In the specific graph database implementation of Neo4j, the type of graph structure used is a directed graph, or digraph, where the edge between two vertices is one-way directed from one vertex to the other. A basic graph is illustrated in Figure 3.1 and digraphs are depicted in Figure 3.2.

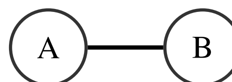
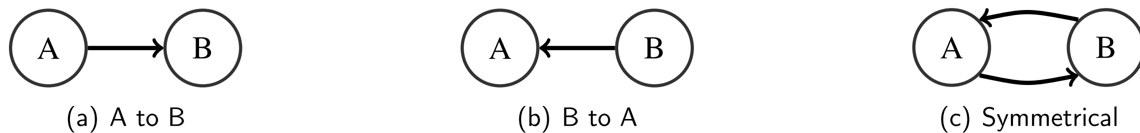


Figure 3.1: Basic graph example.

Mathematically, a digraph is the set of vertices, V, where V is non-empty, and the set of edges, E, consisting of ordered pairs of vertices. In a digraph, those ordered pairs convey directionality of an edge in addition to its existence. For two arbitrary vertices u and v, the ordered pairs (u,v) and (v,u) connect the same two vertices, but are oriented in opposite directions [5]. Figure 3.2(a) shows a directed edge from vertex A to vertex B indicating directionality of the edge that points from A to B, but not from B to A. Figure 3.2(b) is a separate and distinct graph where the edge points from B to A, but not the reverse. In order to enable pointing both from A to B and B to A, two directed edges between A and B, one in each direction as shown in Figure 3.2(c), are required.



(a) A to B (b) B to A (c) Symmetrical

Figure 3.2: Directed graph examples.

Graphs are used for database models when "information about data interconnectivity or topology is more important, or as important, as the data itself" [6]. Data related to an entity can be stored in the node and related information can be found in how the edges connect the various nodes. Specific queries can be run against a graph database such as finding cycles or shortest paths [6]. In Neo4j, data is stored in both the nodes and edges, or relationships, in the form of labels and properties; this specific data storage structure can vary slightly across various graph database models, but the basic elements of nodes and edges are a constant.

3.2 Nodes and Relationships

In graph database models, nodes and relationships are essential elements where the nodes store data about an object and the edges convey data about relationships between objects [6]. Neo4j employs four essential data constructs in its database architecture: nodes, relationships, properties, and labels. Nodes and relationships are the standard nodes and edges found in basic graphs, and in the case of Neo4j, the relationships are directed edges.

Labels and properties are assigned to nodes and relationships for classification and data storage, respectively [7].

For nodes, labels enable the grouping of nodes into categories of similar types; for example, a large database may have nodes classified as animal, vegetable, or mineral. It is also possible for nodes to carry more than one label. The details of each node are stored as properties. For a node for fox, which carries the label of animal, details would be items such as color (red), number of legs (4), and habitat (forest).

Queries can be run against the database specifying not only nodes, but also any of the node properties. A query could be run against the graph database that broadly returns all nodes with the label *animal* or narrowed to a search that returns all nodes with the label *animal* and a habitat of *forest*, which would return all animal nodes in the database with the habitat property of *forest*. Conversely, a query could be run to return all animals that have a habitat property that is not forest. The label can be helpful for grouping nodes but is not a required part of a query; the database could be queried for anything with the color property of red, which would return the animal node: fox and, for instance, a vegetable node: tomato.

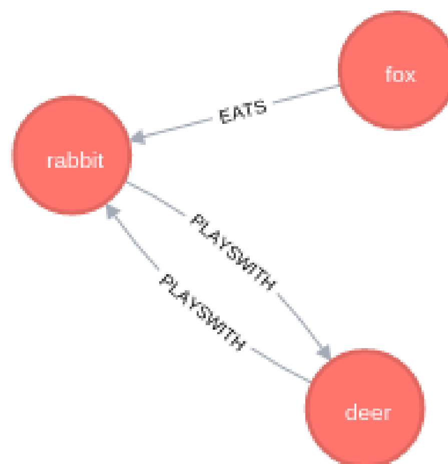


Figure 3.3: Simple animal database.

The relationships of a graph database also contain labels and properties. They are also identified by the start and end nodes they connect, making the database a directed graph. A relationship with the label *eats* could connect the fox node to a rabbit node with the property *nutritional value: protein*.

A relationship can connect any two nodes regardless of their labels; a rabbit node could be connected to a vegetable node *carrot* via an *eats* relationship. These relationships are unidirectional, so any reciprocal connections are best identified by two relationships, one in each direction. A *playswith* relationship from a rabbit node to a deer node would need to be accompanied by an additional *playswith* relationship that starts at deer and ends at rabbit. This reciprocal relationship represented in a graph database would resemble the symmetric graph in Figure 3.2(c). The graph of the database with fox, rabbit, and deer nodes can be found in Figure 3.3.

The directionality of relationships in Neo4j is semantic in nature but does not prohibit a search of the graph database from traversing against the direction of the edges. Using the example of Figure 3.3, the *eats* relationship points from fox to rabbit because foxes eat rabbits, not the reverse. If I wanted to search this database to find what animals eat rabbits, I would be able to construct a query that will follow all of the *eats* relationships that terminate at rabbit backward to find the originating nodes. Although the graph can be searched counter to the direction of the relationships, the directionality of the relationships are firm (i.e., foxes eat rabbits, not vice versa). The directionality of the relationships, not the direction of traversal, is used in this thesis to better understand security in a graph database.

3.3 Neo4j and Cypher

In Neo4j queries of the graph database are executed using Cypher, a custom, "declarative, pattern-matching query language" [7]. This section is meant to be a brief introduction to Cypher with a focus on the basic language structure and commands used to implement and illustrate granular security; for a more comprehensive discussion of Cypher see references [7], [8], [9].

The primary commands in interacting with Neo4j are CREATE, MATCH, and RETURN. Other common commands with high utility that will not be discussed in-depth are WHERE, DELETE, SET, FOREACH, and MERGE; these commands illustrate the breadth of Cypher, but are not necessary for understanding granular security.

The RETURN command is typically used in conjunction with MATCH to instruct the

database to return the results of the query, as opposed to deleting or merging them instead; without the RETURN command, Cypher will match the requested query, but then not actually display the results to the user. The MATCH and CREATE commands, although they are used for different purposes, follow the same syntax. MATCH is used to locate nodes and relationships in the database that correspond to the query criteria. CREATE generates a new node(s) and/or relationship(s) according to the structure of the command. A CREATE UNIQUE command can also be used to prevent duplication of data.

The MATCH and CREATE commands can be as simple or complex as desired. A simple, empty node query would match and return the entire contents of a database:

```
MATCH (a)
RETURN a
```

would match and return the entire contents of a database by finding all nodes and their associated edges. A similar query could also be constructed that queries edges instead of nodes:

```
MATCH ()-[r]->()
RETURN r
```

This query would also return the entire contents of a database by returning all of the edges and their associated nodes. Cypher commands can also be accompanied by relationships, labels, and properties. A more complex instruction would take the form:

```
CREATE (a:Label1 {Property1: "data"})
    -[:RELATIONSHIP {Property2: "quality"}]->
    (b:Label2 {Property3: "info"})
```

This command would create two nodes with a relationship between them. The nodes would be of type Label1 and Label2, each with different property elements. The connection between them is of the type RELATIONSHIP with its own property. Each component of the instruction can contain multiple or no labels or properties; the degree of specificity depends on what the user wishes to accomplish. The "a" and "b" in the command are local variables used within the context of the command to reference already specified nodes as in the MATCH and RETURN command above.

The fox and rabbit nodes from Section 3.2 can be used in a create command:

```
CREATE (a:Animal {name: "fox", color: "red", legs: "4", habitat: "forest"})
    -[:EATS {nutritionalValue: "protein"}]->
    (b:Animal {name: "rabbit", color: "gray", legs: "4", habitat: "forest"})
```

This command will create the fox and rabbit nodes, including their labels and properties, with an eats relationship from fox to rabbit with its property. Once that data is entered into the database, a query could be run to find out what animals are prey for other animals:

```
MATCH (a:Animal) -[:EATS]-> (b:Animal)
RETURN b
```

This query will return the rabbit node (along with any other animal nodes that are eaten by other animals. Using the node labels here, prevents the database from returning other things that are eaten, such as plants.

Queries can also be run against a Neo4j database with much more complexity; Cypher can find and match any pattern that exists within the database. For instance:

```
MATCH (f:Animal) -[:PLAYSWITH]-> (a:Animal) -[:EATS]-> (b:Animal)
    -[:EATS]-> (c:Animal) -[:EATS]-> (d:Animal)
RETURN f,a
```

would return the any animals that are both high up in the food chain and have an animal that they play with; the query would also return the animal that is the playmate. A full graph database of all animals would likely return a *human* node (a in the query) because humans are at the top of the food chain and they play with other animals; a *dog* node (f in the query) would also be returned because humans play with dogs.

Cypher has much more depth and complexity to offer beyond these simple illustrations, but the use of the CREATE, MATCH, and RETURN commands is sufficient for understanding the implementation of granular security. The implementation of that security relies on the use of the property elements for nodes and relationships.

CHAPTER 4:
Granular Security on Graph Nodes

Granular security in a graph database must address the two primary facets of graphs: nodes and relationships. Node security will be addressed in this chapter, and the more complicated edge security will be introduced in this chapter and then be examined in depth in Chapter 5.

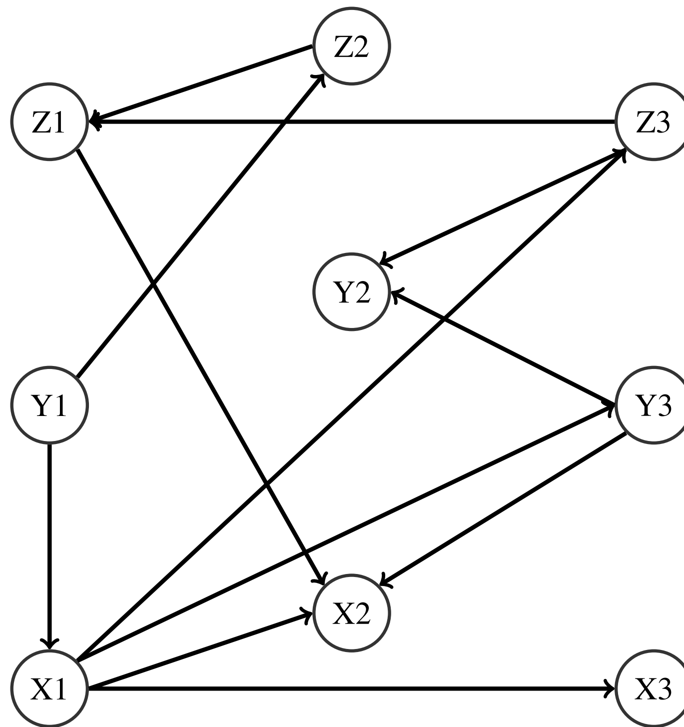


Figure 4.1: Simple manipulation graph.

To examine the various graph structures, the simple graph displayed in Figure 4.1 will be used for illustrative purposes. This graph contains nine nodes and 12 relationships; three nodes will be classified at each security level (X, Y, and Z as discussed in Chapter 2). The graph does not represent any actual data set but is examined as though every node and every relationship contains data.

4.1 Assigning Security

The nodes, being the principal elements of a graph database, must be directly assigned a security property according to the sensitivity of the data represented. These classifications determine which parts of the database a user can access. With no granular security on a graph database of Figure 4.1, a query for all the nodes would return exactly what is displayed.

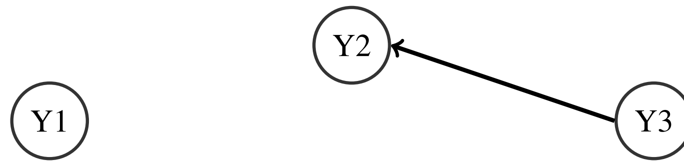


Figure 4.2: Query with node security.

Because nodes are both necessary and sufficient elements of a graph (i.e., a graph can contain nodes and no edges, but a graph cannot contain only edges because they, by definition connect nodes), they must receive a security classification at the time they are created, and they must always retain some security designator. While the security classification can be changed, it cannot be deleted from the node.

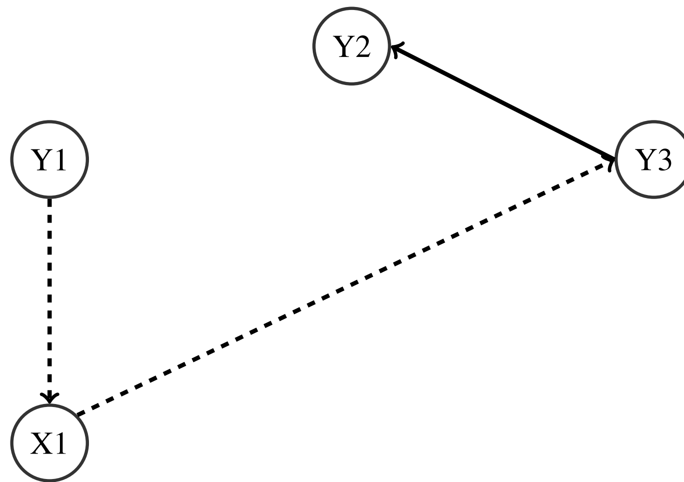


Figure 4.3: Connection not visible with node security.

The same database with granular security implemented on the nodes would return a subset of the total graph according to the user's assigned security level. A user with access to only

Y-level information could query the database for all nodes, but the results of that search would be restricted. As shown in Figure 4.2, it would return all Y-level nodes rather than all nodes.

What Figure 4.2 does not show to the Y-level user are indirect connections that exist when a relationship extends to a non-Y-level node and then back to a Y-level node. The two-edge connection between the Y1 and Y3 nodes via X1 seen in Figure 4.3 is a link hidden from A Y-level user. While the data at the X1 node lies outside of the user's access level, the connection between the two Y-level nodes may be a valuable link that is not visible to the user using only node security.

4.2 Relationship Types

The main goal of any database is to store data in such a way that a user can access that data in a way that is useful. A graph database has the added objective of showing connections between data points the user might not be able discern in another data storage structure. In the case of Neo4j, those relationships can also store data that expands on the nature of the relationship between two nodes.

The limitation of granular security on a graph database is that it shows only relationships in the subgraphs of a node set at a given security level; connections between data points in different security compartments would not be visible to users operating one of those compartments, but not both. In a hierarchical security model, the connections occurring with lower security levels would ideally be visible, but valuable connections that exist to a higher level would not be seen by a user at the lower level. Without these cross security level connections accessible in some way, using a graph database of combined security levels provides no data analysis advantage over keeping the compartments on physically separate systems.

The most straightforward way to handle the issue of edges would be to individually classify all of them in the same way that the nodes are classified. While this method is likely the most secure, because each edge must be individually assigned a security level, it is also the most cumbersome and the most likely to lead to the least optimal data organization for observing connections between data points. Individually classifying relationships is a good

option to have for handling specific pieces of information, but it is not the best route for a general data disclosure policy for the relationships between nodes of the database.

Three main types of relationships between nodes of different levels exist: trivial, restricted, and induced. Same and different are relative terms that designate whether the two nodes at the ends of a relationship are of the same or different security levels. These relationship types will be introduced here and a graphical analysis of the most optimal policy for handling these relationships will be discussed in Chapter 5. The nodes for illustrating these relationship types will be labeled "X" and "Y" where "X" is the security of the current user and "Y" is some other security level.

4.2.1 Trivial Relationships

A relationship from a given node to another node of the same classification level, as illustrated in Figure 4.4, is the trivial case for this exercise. The two nodes would both be visible to a user at level X, and thus the relationship between them would be returned for any query that returns those two nodes.

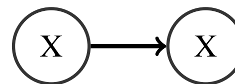


Figure 4.4: Trivial relationship.

4.2.2 Restricted Relationships

A relationship from a node at one security level to a node at a different security level, as illustrated in Figure 4.5, would not be seen by a user operating at either of the security levels. A user at level X would only see the X node, and a user at level Y would only see the Y node; neither one would see the opposite node nor the presence of a connecting relationship. Because Neo4j is a directed graph, the restricted relationships can be either an outbound relationship, as illustrated in Figure 4.5(a), or an inbound relationship, as illustrated in Figure 4.5(b). These two cases, although they are directionally opposite, will be addressed as the same type.

With this relationship, it is unclear what the classification is of either the existence of the relationship or of the content of the relationship. Whether or not the X- or Y-level user should

be able to have access to the relationship could also be dependent on if the relationship is an inbound relationship to the user's security level or outbound from the user's security level. Even though it restricts the visibility of the graph's connectivity, the recommended policy for this type of relationship is to only allow it to be visible to a user if that user has access to both the X and Y security levels. The graph of Figure 4.1 contains examples of each relationship type. For a user at level X, only the X node should be visible with granular security. The Y-level node should only be visible to a Y-level user and the relationships between the X- and Y-level nodes should only be visible to a user who as permission to access both levels.

Figure 4.5: Restricted relationship.

4.2.3 Induced Relationships

The induced relationship is a set of two relationships. The first relationship is a simple restricted relationship that begins in the user's security level. The second relationship is another restricted relationship that begins at the terminal node of the first relationship and points back to a node in the user's security level creating a two-edge path that starts and ends in the same security level.

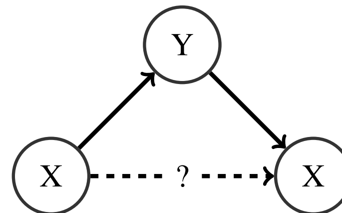


Figure 4.6: Induced relationship.

This relationship type is illustrated in Figure 4.6. In this relationship a connection exists between two nodes of the same classification level that a user at that classification level would not be able see. Chapter 5 will explore this relationship type using graph theory methods to find a mathematical justification for showing the induced relationship (marked with a '?') that exists between the two X-level nodes.

More complicated relationship types exist that expand upon the basic induced relationship type. These types will be briefly addressed in Chapter 7, but they remain largely outside the scope of the present work.

4.3 Possible Relationship Security Policy

A potential security policy for keeping data from being disclosed to unauthorized users while leveraging the connectedness of information within a graph database requires a thoughtful balancing act. In all cases of a security policy implemented directly on the nodes of a graph, a user at a given security level would be able to access all the nodes at the corresponding level (i.e., a Y-level user would be able to access all Y-level nodes).

In the case of trivial relationships, a security policy should allow a user at a given security level to access the relationships that both originate and terminate at the user's access level. Restricted relationships, regardless of whether they originate or terminate in the user's security level, should only be accessible to a user if she has access to both security compartments within the database. Because the relationships cannot exist without the nodes, having a relationship that points to nowhere does not fall within the typical parameters of a graph database, and knowing that a relationships exists without knowing the other end of the connection is not likely to provide much additional information.

For induced relationships that cross to another security level and then back to the user's security level the relational elements that graph databases provide can be leveraged by giving the user access to the induced relationship that exists between the two nodes. This induced relationship allows the user to see the presence of the data connection without also having access to the information in a security level to which she does not have access.

CHAPTER 5:
Security Exploration on Graph Edges

Understanding the nature of security on the edges in a graph database requires exploration of the possibilities of graph representation. Organizing the same graph using different network models can reveal characteristics of the connections that become visible in the different models.

The goal of the present chapter is to use the construction models of graph theory to find a mathematical justification for generating the edges of induced relationships within a large graph database. On a small graph, finding those relationships is easy to accomplish visually; however, on a large data set, finding the induced relationships would need to be accomplished via a sound mathematical algorithm.

Using a simple manipulation graph from Figure 4.1, the different ways in which that same graph can be represented will be illustrated and evaluated for its applicability to producing the induced relationships as well as the overall impact to the graph database. The discussion will begin with ruling out hypergraphs as well as weighted and temporal graphs. Colored graphs, although germane to the task, will next be shown to provide no additional insight into the problem at hand. The chapter will conclude with a look at the bipartite feature multislice graphs to produce a mathematical process for finding induced relationships.

5.1 Hypergraph

A hypergraph is comprised of nodes and edges; however, the edges are not limited to connecting a maximum of two nodes. Edges in hypergraphs, called hyperedges, can pass through any number of nodes between the two end nodes [10]. For a database in Neo4j, hyperedges would replace the directed relationships and their content. Instead of a relationship that contains information, each edge would become a directed hyperedge that would have the same beginning and ending nodes as the regular directed edge, but would also pass through a fabricated relationship-node. These hyperedges are implemented on the simple manipulation graph in Figure 5.1 with each of the new nodes labeled *he*.

This construction with hyperedges leaves a graph database that has many more nodes, but with relationships that do not store any content. Classifying the content of the relationships becomes a moot issue since that data is now stored in nodes, which must receive a security designation.

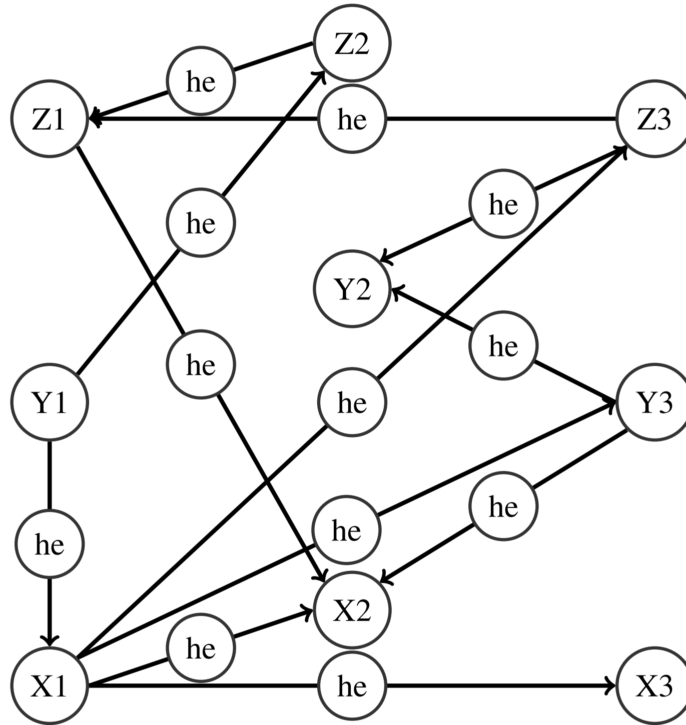


Figure 5.1: Simple manipulation graph with hyperedges.

Creating these additional nodes does not address the issue of how the database should address the restricted and induced relationships that cross between security levels. In addition, creating more nodes with independent classification levels could increase the prevalence of these ambiguous relationships. While a hypergraph construct allows for an easy way to directly classify the content of the edges, it does not aid in addressing the issue of how to handle the more complicated, cross-compartment security relationships.

5.2 Weighted Graph

A graph with weights assigned to each edge is called a weighted graph. In these graphs traversing from one node to another along an edge is associated with a cost; the distance

between cities is an example of a weight that might be placed on an edge. For example, the distance from New York to Los Angles is about 2,500 miles while the distance from New York to Philadelphia is only 100 miles. Thus, the weight of an edge connecting the New York and Los Angles nodes would be 2,500, and the weight of an edge connecting the New York and Philadelphia nodes would be 100. The edges could be weighted via any metric appropriate to purpose of the graph; travel time, fuel consumption, or number of rest stops are examples of other weight options.

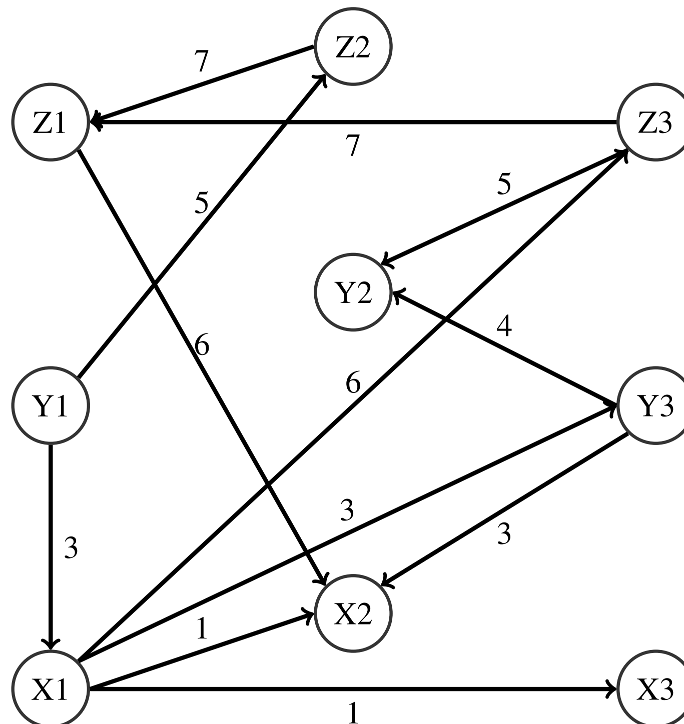


Figure 5.2: Simple manipulation graph with edge weights. Relationships are given weights according to the following–X to X: 1, Y to Y: 4, Z to Z: 7, X to Y: 3, Y to Z: 5, X to Z: 6.

In a graph database with cell level security, the weights could serve as a way to regulate the edges a user can access. If a user is assigned a path traversal budget that is applied independently, not cumulatively, to each path (either in addition to or instead of a security clearance level), then he would be able to access any paths that are less than or equal to that maximum budget. The simple manipulation graph with weights applied is found in Figure 5.2.

In a weighted graph database, the user's security level along with his traversal budget would determine what information was accessible. A user at the Y-level with a traversal budget of 4 would be able to see all the trivial relationships at the Y- level, because they each weigh 4, as well as be able to gain information about relationships that extend down to the X-level, because they each weigh 1, but not relationships that extend to the Z-level, because they each weigh 5. He would also not be able to see any induced relationships to the X-level because each hop on the path to the unclassified level and back weighs 3 for a total induced cost of 6, more than the user's traversal budget.

If that same user were given a travel budget of 6, he would be able to see the relationships up to the Z-level as well as the induced relationships to the X-level. If the traversal budget is used with the security level, the nodes outside of the user's security level would still not be accessible.

Displaying the relationships that are anything other than the trivial type would entail somehow conveying a relationship without all the end nodes. The most straightforward way to display relationships in this way would be to make them hyperedges, while doing so does solve the problem of displaying one-sided relationships, it does not allow the user to accurately see induced connections. This solution ultimately leaves the user with the same limitations that exist with using a hypergraph.

Using a weighted graph model without also classifying the nodes is another option implementing granular security. In this construct a user would still be given a traversal budget that would, in this case, not be restricted to the visibility of the nodes. Only assigning a weighted traversal security to the nodes, however, would only be applicable to hierarchical security environments; a traversal budget with a large value would be able to see all edges of lesser value. In addition, devising a scheme to appropriately assigned edge weights and traversal budgets to ensure proper disclosure to all users would require a difficult algorithm with high potential for giving improper access to data.

5.3 Temporal Network

A temporal network is similar to a weighted graph where the weights instead operate similar to timestamps. In this graph type, the presence of an edge depends on when along

a timeline the graph is examined [11]. Normally this type of graph is employed to show how the connections in a graph change over time. But, if windows of time are instead seen as security levels, the concept could potentially translate to allow access according to the values of the weighted edges. For instance, supposing that the weights of Figure 5.2 are timestamps, the graph at timestamp 4 would show only the trivial relationships at the X-level.

Timestamps can also be assigned for periods of time. If a user were given access to all timestamps 4 and less, then the relationships at the X and Y levels would be visible. Similarly a timestamp access from 3 to 5 would give a user access to all relationships connecting to and/or from the X-level. Although the timestamps appear to be relegated to a hierarchical system, they could be implemented in a way that makes compartmentalization possible. User timestamp access could also be disjoint; a user with access to timestamps 1, 6, and 7 would be able to see the Z and X-levels as well as the connections between them.

The temporal framework provides an excellent model for examining edge security, but the timestamps only act as a proxy for assigning edge security directly. In a security environment where this direct assignment is desirable the temporal model would provide insight into how to best conduct that arrangement. The model, however, does not provide additional means for leveraging the restricted and induced relationships.

5.4 Coloring

Although coloring graphs has its foundations in cartography, it is easily applicable to nodes and edges [5]. Applying colors to a graph can help to bring out patterns and connections that are not readily apparent in a graph without those classifications. Graph coloring is typically employed for situations where two or more graphs may exist mostly independently, but those quasi-independent graphs have connections tying them together [12]. The segregation of a graph database into separate security lends itself to viewing those separate compartments and independent graphs with connections linking the data across security levels.

5.4.1 Node Coloring

Applying node coloring to a network requires assigning a single color to every node in the graph. Traditionally the coloring is applied in such a way that no two adjacent nodes share the same color; in the present work, however, the node coloring serves as a node label. A graph database with granular security applied is essentially already node colored. The security labels serve to separate the nodes into categories–the intention of coloring a network. Associating colors with the security levels, as can be seen on the simple graph in Figure 5.3 will give actual colors to the existing node labeled graph.

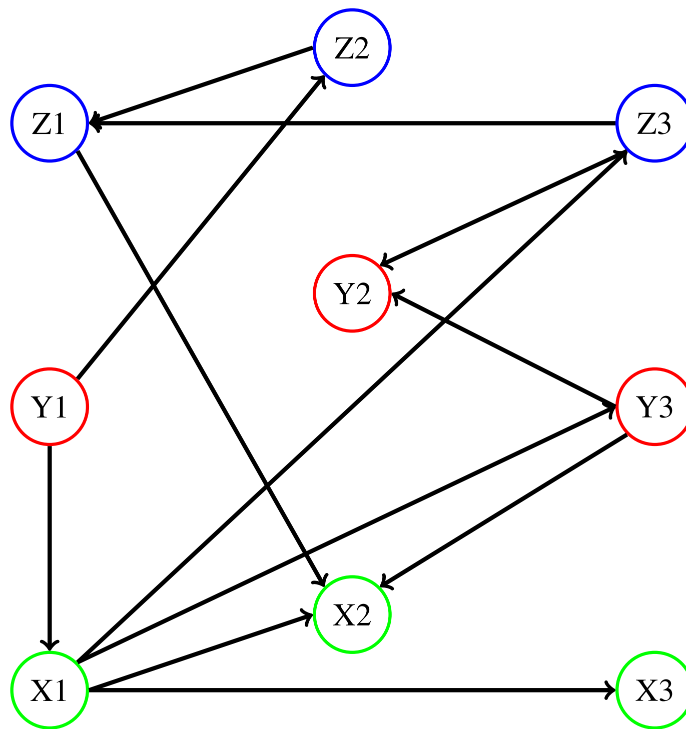


Figure 5.3: Simple manipulation graph with node coloring. Colors are assigned according to the following–X:green, Y:red, Z:blue

5.4.2 Edge Coloring

Similar to node coloring, edge coloring is applying a single color to every edge in a network; in the same way, the edge colors serve as labels and, like the node colors, are not applied in the typical non-adjacent fashion. The way edges are classified and colored can vary.

In a compartmentalized graph, they can be colored based on the security levels where they originate, where they terminate or both. They can be colored according to whether or not they cross a security level. In the graph database where relationships are given types, the edges could be colored based on those labels. Given that the sample graph does not have relationship labels, the simple graph in Figure 5.4 is edge-colored based on the classification of the node where the edge originates.

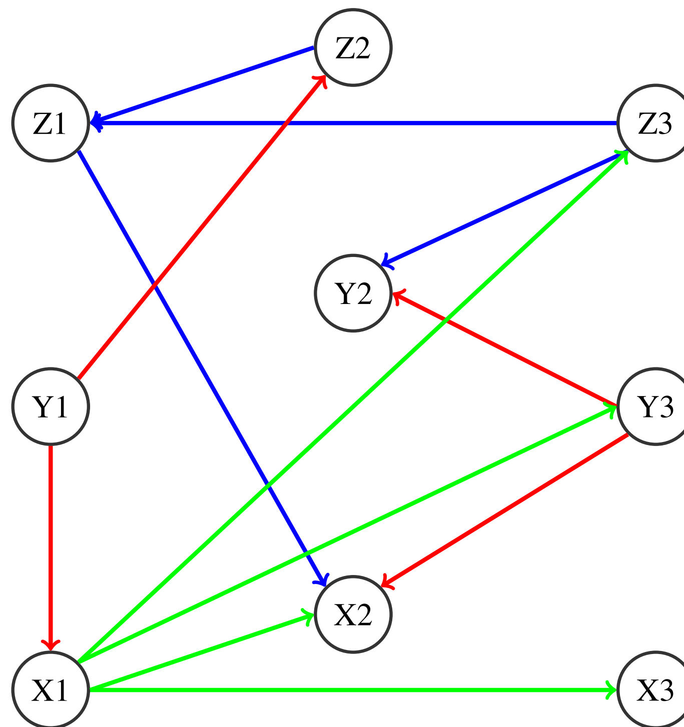


Figure 5.4: Simple manipulation graph with edge coloring. Colors are assigned based on the classification of the originating node according to the following–X:green, Y:red, Z:blue

5.4.3 Combined Coloring

Use of simultaneous edge and node coloring on the simple graph can be seen in Figure 5.5. These coloring schemes do not actually provide any new information. While the chromatic highlights bring visual attention to the features of the graph, their assignment is based solely on labels or attributes that were already present in the graph.

In Figure 5.5 the number of nodes at each classification level can be easily ascertained;

in addition, the connections that link nodes of different security levels are also apparent. The limitation is that the classification of the nodes and the attributes of the edges are not any different now than they were before the colors were applied. Any graph database large enough to be useful will be much too large to examine visually, making the color assignment an aesthetically pleasing feature, but one with no practical utility.

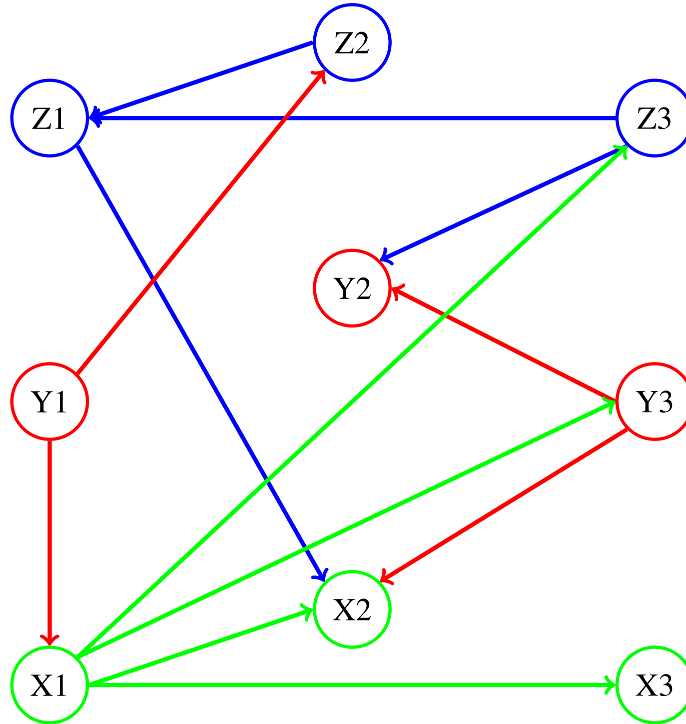


Figure 5.5: Simple manipulation graph with node and edge coloring.

5.5 Multilayer Graphs

The idea of a node colored graph, distinguishing nodes based on their labels, can be applied in an alternate way yielding a graph that can be analyzed differently. Instead of distinguishing nodes of the same security level by assigning a color, grouping the nodes of the same level all to a single layer reveals where relationships cross from one security level to another in a way that can be utilized for analysis. Figure 5.6 shows the simple graph rendered as a multilayer graph with each node on its appropriate classification layer.

5.5.1 Multiplex Graphs

Studies of multilayer networks frequently render graphs as multiplex networks, a subgroup of multilayer networks [13], [14]. Multiplex networks are a specific type of multilayer network where every node has a counterpart in every layer of the network [13].

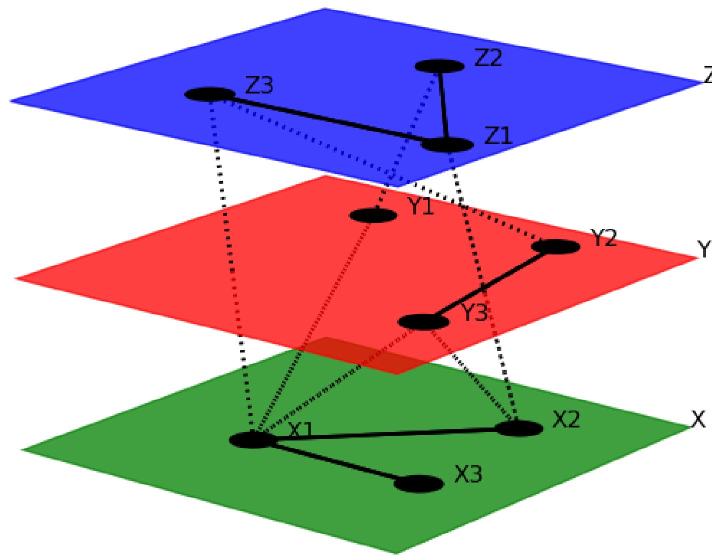


Figure 5.6: Simple manipulation graph as multilayer graph. Edges that remain originate and terminate on the same layer are solid lines; those that cross layers are dotted lines. The directionality of the network is omitted in this image due to the limitations of the visualization software.

This replication is sometimes done for simplicity, but it is usually used to represent constructs such as social networks where one node will have a presence in more than one layer. For example, a single social media user may have an account in Twitter, Facebook, and Instagram with different connections for each site. Those different services are then each represented as a single layer within a broader social network. A user with no Twitter account would just be an unconnected node in that layer.

Typically, in a multiplex network, connections between layers can only be made across the replicated nodes in each layer. The node for user A on the Facebook layer can only have intra-layer connections to user A on the Twitter or Instagram layers. This restriction reflects the reality that user A's Facebook account cannot connect directly to user B's Twitter account.

A security environment where the nodes are intentionally kept separate from each other does not lend itself to this common multiplex model. Copying a Z-level node into the X layer would defeat the purpose of having granular security for the database. While the copies of each node in multiple layers would allow for better access when relationships cross security levels, the security features would be rendered useless, and the result would ultimately be similar to producing multiple copies of one database. The more general idea of a multilayer graph is better suited to the needs of a secure database model.

5.5.2 Multislice Graphs

Applying slices to layers in a multilayer graph requires examining a separate layer or layers of the graph independently, allowing examination of subsets of edges based on those layers [15]. Multislice graphs are typically applied to multiplex graphs with all nodes replicated in every layer [12]; however, they can also be applied to generic multilayer graphs.

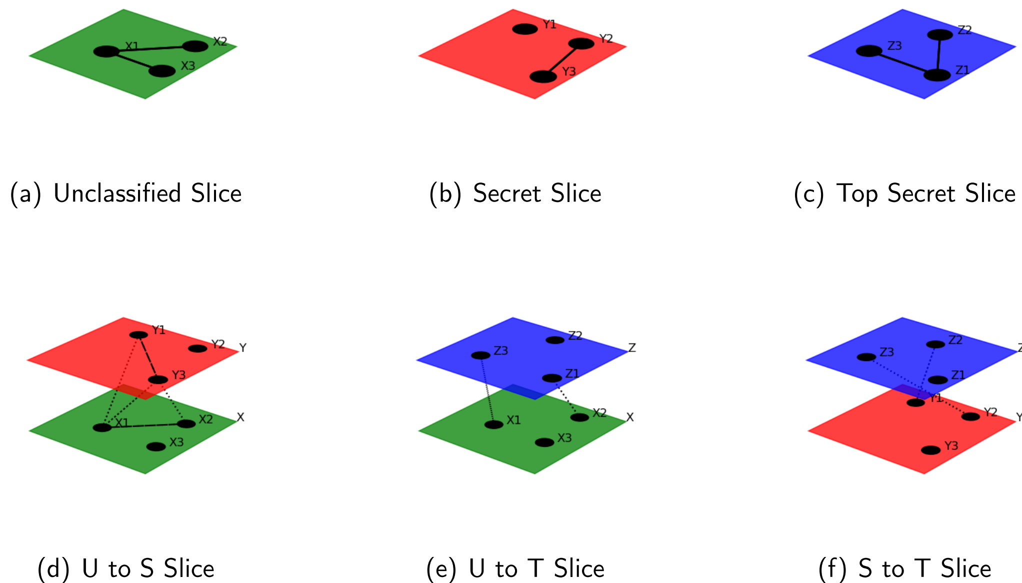
For a compartmentalized graph database, slicing isolates the individual layers as well as the connections between any two layers. In Figure 5.7, six slices of the simple graph are displayed. Each individual security level has its own slice and an additional slice is constructed for each pair of security levels. The number of slices will grow in proportion to the number of security levels in the database according to:

$$|S_n| = \frac{|L_n|(|L_n|+1)}{2} \tag{5.1}$$

where $|S_n|$ is the number of slices required and $|L_n|$ is the number of layers (i.e., the number of security levels in the database).

The individual slices pull out the trivial relationships, which does not add much to the ability to analyze the whole graph. The slices that contain pairs of security layers, on the

other hand, isolate the restricted relationships apart from the edges that do not cross security levels. In Figure 5.7(d), the induced relationships can be clearly seen. A connection exists from node X1 to node X2 and from node Y1 to node Y3. In the case of the former edge, a parallel relationship already exists in the graph, but the database can accommodate multiple relationships between the same pair of nodes. In the latter case, the relationship between the two nodes is not readily visible before examining the slice.



(a) Unclassified Slice (b) Secret Slice (c) Top Secret Slice

(d) U to S Slice (e) U to T Slice (f) S to T Slice

Figure 5.7: Simple manipulation graph in multislice format. (a), (b), and (c) each show a slice with one security level and all the trivial connections at that level. (d), (e), and (f) each show a slice with a pair of security levels. The dotted lines represent connections between levels, and the dashed lines in (d) represent induced connections across the two security levels. The directionality of the network is omitted in this image due to the limitations of the visualization software.

In a large graph, visually finding the induced relationships is not a feasible task. Mathematically finding those inferred relationships, however, is a possibility using the slices that contain the relationships between the security levels.

5.6 Bipartite Graphs

Each of the paired slices makes up a subgraph that is bipartite. Bipartite graphs are graphs where the nodes can be split into two groups (the security levels in this case), and edges

connect nodes from one group to another with no edges running between two nodes in the same group [5]. Because the trivial relationships are not contained in the paired slices the only relationships present connect nodes from one security level to another making the paired slices bipartite graphs.

The bipartite graphs that contain only the edges that connect two layers (Figures 5.7(d), 5.7(e), and 5.7(f)) can be used to mathematically locate the edges that exist behind any induced relationships. Using multiplication of the adjacency matrices of those bipartite graphs results in a new adjacency matrix for the induced edges.

5.6.1 Adjacency Matrices

An adjacency matrix is a matrix representation of a graph. The columns and row indices of the matrix represent the nodes of the graph, so a matrix with *n* nodes would have an *nxn* adjacency matrix. In an undirected graph, there will be two ones in the matrix for every edge and zeros for all other locations. For an undirected graph with an edge *(u,v)* the associated adjacency matrix will have a 1 at row *u*, column *v* and a 1 at row *v* column *u*.

If the simple graph were undirected, then the associated adjacency matrix would be represented as:

$$
\begin{array}{c}
\\ X1 \\ X2 \\ X3 \\ Y1 \\ Y2 \\ Y3 \\ Z1 \\ Z2 \\ Z3
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc} X1 & X2 & X3 & Y1 & Y2 & Y3 & Z1 & Z2 & Z3 \end{array} \\
\left(\begin{array}{ccccccccc}
0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\
\mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} \\
\mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0
\end{array}\right)
\end{array}
\tag{5.2}
$$

The 1 at the row for node X1 and the column for X2 represents the edge from X1 to X2 and has a corresponding 1 at the row for X2 and the column for X1. The simple graph,

however, is a directed graph and the edges do not flow in both directions.

5.6.2 Directed Adjacency Matrices

For a directed graph, two methods exist for representing the associated directed adjacency matrix. One way involves using ones to represent an edge that is directed from the row node to the column node and a negative one to represent the edge at the inverse row and columns of the matrix. In this method the *(X1, X2)* value would be one because the edge is directed from X1 to X2 and the *(X2, X1)* location would be a negative one.

The other method for representing directed adjacency matrix is to use ones for the direction of the edge and to leave the inverse location as a zero. Using this representation the directed adjacency matrix for the simple graph becomes:

$$\begin{array}{c} \\ X1 \\ X2 \\ X3 \\ Y1 \\ Y2 \\ Y3 \\ Z1 \\ Z2 \\ Z3 \end{array} \begin{array}{c} \begin{array}{ccccccccc} X1 & X2 & X3 & Y1 & Y2 & Y3 & Z1 & Z2 & Z3 \end{array} \\ \begin{pmatrix} 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \end{pmatrix} \end{array} \tag{5.3}$$

For an undirected bipartite graph, the partial adjacency matrix, where one group is the rows and the other group the columns, the matrix can be multiplied by its transposition, the opposite matrix where the groups switch rows and columns, to return a third matrix that shows the instances where a node is connected to another node in the same group with a node from the other group between them [16].

Using the undirected matrix for the Unclassified and Secret slices yields the matrix multiplication:

$$\begin{array}{c} \\ X1 \\ X2 \\ X3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} Y1 & Y2 & Y3 \end{array} \\ \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} \\ 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 \end{pmatrix} \end{array} x \begin{array}{c} \\ Y1 \\ Y2 \\ Y3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} X1 & X2 & X3 \end{array} \\ \begin{pmatrix} \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 \end{pmatrix} \end{array} = \begin{array}{c} \\ X1 \\ X2 \\ X3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} X1 & X2 & X3 \end{array} \\ \begin{pmatrix} \mathbf{2} & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array} \tag{5.4}$$

where the entries along the diagonal represent the degree of each node (X1 is degree, X2 is degree 1, and X3 is degree 0, which agrees with Figure 5.7(d) and the other ones represent the induced relationships from the bipartite graph on the Unclassified layer, namely the relationship from X1 to X2. Reversing the order of the multiplication yields:

$$\begin{array}{c} \\ Y1 \\ Y2 \\ Y3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} X1 & X2 & X3 \end{array} \\ \begin{pmatrix} \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & 0 \end{pmatrix} \end{array} x \begin{array}{c} \\ X1 \\ X2 \\ X3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} Y1 & Y2 & Y3 \end{array} \\ \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} \\ 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 \end{pmatrix} \end{array} = \begin{array}{c} \\ Y1 \\ Y2 \\ Y3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} Y1 & Y2 & Y3 \end{array} \\ \begin{pmatrix} \mathbf{1} & 0 & \mathbf{1} \\ 0 & 0 & 0 \\ \mathbf{1} & 0 & \mathbf{2} \end{pmatrix} \end{array} \tag{5.5}$$

which reveals the degree of the nodes on the Y-layer and the induced relationship between nodes Y1 and Y3.

Splitting the directed adjacency matrix according to the security level paired slices of Section 5.5.2 gives adjacency matrices for those directed bipartite subgraphs. The graph database and the simple graph, however, are both directed graphs. When performing the same multiplicative process on the directed adjacency matrices, the second matrix is not the transposition of the first, but rather the adjacency matrix for the edges pointing to the opposite group. The directed adjacency matrix multiplication for both directions yields:

$$\begin{array}{c} \\ X1 \\ X2 \\ X3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} Y1 & Y2 & Y3 \end{array} \\ \begin{pmatrix} 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array} x \begin{array}{c} \\ Y1 \\ Y2 \\ Y3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} X1 & X2 & X3 \end{array} \\ \begin{pmatrix} \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \end{pmatrix} \end{array} = \begin{array}{c} \\ X1 \\ X2 \\ X3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} X1 & X2 & X3 \end{array} \\ \begin{pmatrix} 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array} \tag{5.6}$$

$$
\begin{array}{c} \\ Y1 \\ Y2 \\ Y3 \end{array}
\begin{array}{c} \begin{array}{ccc} X1 & X2 & X3 \end{array} \\ \begin{pmatrix} \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \end{pmatrix} \end{array}
x
\begin{array}{c} \\ X1 \\ X2 \\ X3 \end{array}
\begin{array}{c} \begin{array}{ccc} Y1 & Y2 & Y3 \end{array} \\ \begin{pmatrix} 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}
=
\begin{array}{c} \\ Y1 \\ Y2 \\ Y3 \end{array}
\begin{array}{c} \begin{array}{ccc} Y1 & Y2 & Y3 \end{array} \\ \begin{pmatrix} 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}
\tag{5.7}
$$

Equations (5.6) and (5.7) show the directed edges that were visually added to Figure 5.7(d). This matrix multiplication process can be performed over a large graph where visual inspection cannot. Separating the larger graph into two-layer slices allows for the isolation of the edges that cross from one security level to another. The multiplication of the resulting bipartite adjacency matrices mathematically generates the edges that result from an induced relationship.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6: Granular Security Implementation

Implementing granular security in Neo4j would require adding a protocol to the database that would take a user's query and translate that input into a secure query. The database system would need to add elements to the user query that would only return results that align with the security authorizations of the user.

6.1 Granular Security through Properties

Implementation of granular security in Neo4j can be done through the use of properties within the nodes and relationships. Using a reserved word (e.g., *security*) as a property with the appropriately assigned values implements granular security on the database nodes.

A MATCH or CREATE command could perform a hidden operation, using the same Cypher format, to match or create nodes with a security property holding a value equivalent to the authorization found for the user in the *auth* file. A user's MATCH command could be appended with an additional security property after the user enters the command. The initial command that would return all nodes would have the property added to it before it was actually queried against the database:

```
MATCH (a {security: "Y"})
RETURN a
```

Instead of returning all nodes, this query would return only the nodes that matched the "Y" security property. The addition of a hidden security property, which the user cannot modify directly, allows for the implementation of granular security.

6.2 Proof of Concept

Neo4j, in addition to distributing the community version via free download, also hosts their source code on GitHub and invites users to contribute to the development of the software. This Neo4j source code file is designed to easily import the addition of user-written plugins

that can expand on the query capabilities of Cypher.

Writing and using plugins comes with several limitations. Because they are not embedded in the Cypher compiler, they cannot be run from the Graphical User Interface (GUI) as a normal query would. Plugins must be executed either through command line interface with Neo4j or though the GUI console in Hypertext Transfer Protocol (HTTP) mode. A plugin also does not have the necessary file permissions to access user data where information about a user's authorization tokens would be stored.

The plugins are simply additions to the Neo4j command library, thus, they cannot replace existing Cypher commands. A new MATCH command implemented via plugin that requires a security level to execute could simply be avoided by using the standard MATCH command of Cypher. The plugin to implement a match query serves as a proof of concept that the granular security could work as anticipated rather than serving as a secure, functioning access control tool.

6.3 Plugin

Plugins can be written to execute queries that simply perform specific Cypher queries through the HTTP console. The plugins are either written fully in Java or use Java to execute a Cypher command. The AltMatch plugin, written to implement granular security on Neo4j nodes, performs MATCH and RETURN operations on all nodes in the database with the security clearance level specified by the user.

From the HTTP console the AltMatch command can be called with a GET request:

```
GET /altmatch/<clearance>
```

The structure of the command depends on how the plugin file for the plugin is packaged; in this case the "altmatch" portion of the GET request instructs the console to use the AltMatch plugin file. The "<clearance>" of the GET request allows the user to specify the security level of the nodes they query should return.

Using the artificial security labels for illustration, a query to search for all Y-level nodes would take the form:

```
GET /altmatch/Y
```

This command would return the data for each Y-level node in the database. Supposing that the fox and rabbit nodes from Chapter 3 have a security property of "Y", the output from the above query would be:

```
==>200 OK
==> {"matches":["fox","rabbit"]}
```

The deer node, which has a different classification, would remain hidden. Because the AltMatch plugin returns the results of a specific query, returning more information about the two nodes would require a separate plugin that would return the desired information. All separate queries would require separate plugins to run with the security level requirement making the plugin functionality not a realistic option for implementing granular security.

6.4 Cypher GUI

When called through the HTTP console, the AltMatch plugin runs the Cypher query:

```
MATCH (p {security: <clearance>}) RETURN p
```

Using the same animal node database, that same query run directly in the Cypher GUI returns the graph shown in Figure 6.1. The query for the plugin run in the database GUI returns not only the names of the nodes but also the full set of properties for each node, which can be viewed by selecting the node in the graph view or by switching to row view. Displaying this additional data about the nodes is consistent with the intentions of granular security, since all the data for a given node is classified at the same security level as the node itself.

In addition to the two nodes at the Y-level, the Neo4j GUI displays the induced subgraph for the two nodes. An induced subgraph contains the set of all edges (u,v) for every u and v that are a part of the subgraph [5]. In a graph database, for every pair of nodes displayed, a query also returns any relationships that connect those two nodes if such an edge exists in the database. With Neo4j these relationships contain labels and properties that are also returned with the query. Generating a sound security policy to return the appropriate rela-

tionships for the induced subgraph for the results of a query at a given security level is the complicated aspect of implementing granular security in a graph database.

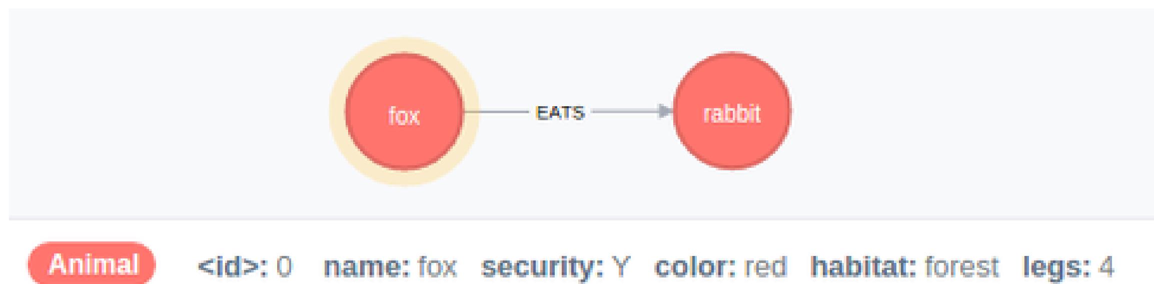


Figure 6.1: Results from Neo4j GUI query with granular security.

Neo4j can also return results to a query on only relationships, such as:

```
MATCH () -{r:EATS}-> () RETURN r
```

In this case, the graph returned would be a subgraph of the relationships returned as well as the nodes at either end of those relationships (i.e., the same graph returned in Figure 6.1). In addition, Neo4j would return an expanded induced subgraph of those nodes by including any additional relationships between those nodes even if they were not requested in the original query.

If another type of relationship existed between fox and rabbit, that relationship would also be returned. If the fox also had an *eats* relationship with the deer node then the deer node would be returned with the query along with the *playswith* relationships between the rabbit and deer nodes (even though that relationship was not a part of the original query). Proper direct security classification of the nodes would prevent the nodes from being returned to an unauthorized user, and if the node is not returned then a relationship to that node cannot be returned at all.

6.5 Synthetic Graph Analysis

Generating and verifying the connections from induced relationships on a graph of nine nodes is a simple task to perform visually. The same undertaking on a larger graph cannot reasonably be performed by hand, and instead must be calculated via computer. The

induced relationship creation was performed on a randomly generated graph both within Cypher as well as via matrix multiplication on the random graph's adjacency matrix.

This random graph was generated [17] using an Erdos-Renyi [18] model where the number of nodes was set to 300 and the probability that any given edge exists is 0.02, creating 896 total edges in this instance. To convert the generated graph into a directed graph, each edge was randomly assigned directionality with probability of 0.5 for either direction.

To simulate an actual security environment with this random graph, each node was assigned a security level according to the DOD classification system: Unclassified, Secret, and Top Secret. Since the graph was randomly generated, the security assignments were made via numbered groups (i.e., nodes 1-100: Unclassified, nodes 101-200: Secret, nodes 201-300: Top Secret).

In the hierarchical security environment of the DOD, a user at a higher level is assumed to have access to information at lower levels (with the exception of any additionally compartmentalized information as discussed in Chapter 2). The hierarchical structure does not fundamentally alter the application or the solution.

An authorized user has access to some subset of the entire database according to their access level. Induced relationships can be created between the security layers that the user can access and any other layers in the database. In the DOD hierarchical environment, a Secret level user, for instance, also has access to Unclassified information; induced relationships can then be created between the Secret and Top Secret layers as well as between the Unclassified and Top Secret layers. Induced relationships could also be created between the Unclassified and Secret layers, but since the user has access to both layers no new connections would be identified.

The directed graph with security properties that resulted from the random graph generation was loaded into Neo4j. The database was queried to find the induced relationships that existed between each pair of security levels using the following query where <user's security level> and <other security level> are replaced by "Unclassified", "Secret", or "Top Secret" (six queries total, one for each possible pair of security levels):

```
MATCH (n {security: <user's security level>}) -[]->
    (m {security: <other security level>}) -[]->
    (p {security: <users's security level>})
RETURN n, m, p;
```

In Neo4j the query found 107 relationships that traversed from an Unclassified node to a Top Secret node and back to an Unclassified node.

The Unclassified by Top Secret and Top Secret by Unclassified adjacency matrices for the graph were multiplied as in Equations (5.6) and (5.7). That multiplication returned a matrix of 107 induced edges between Top Secret nodes. This same step was performed for every layer pairing in both directions (6 parings total). The Cypher queries locating the induced relationships and the matrix multiplication of the bipartite graphs of each security layer pair agreed in every instance how many induced edges existed.

The induced edges for the Unclassified layer were added using the following Cypher command (plus the corollary for the induced edges with the Secret level):

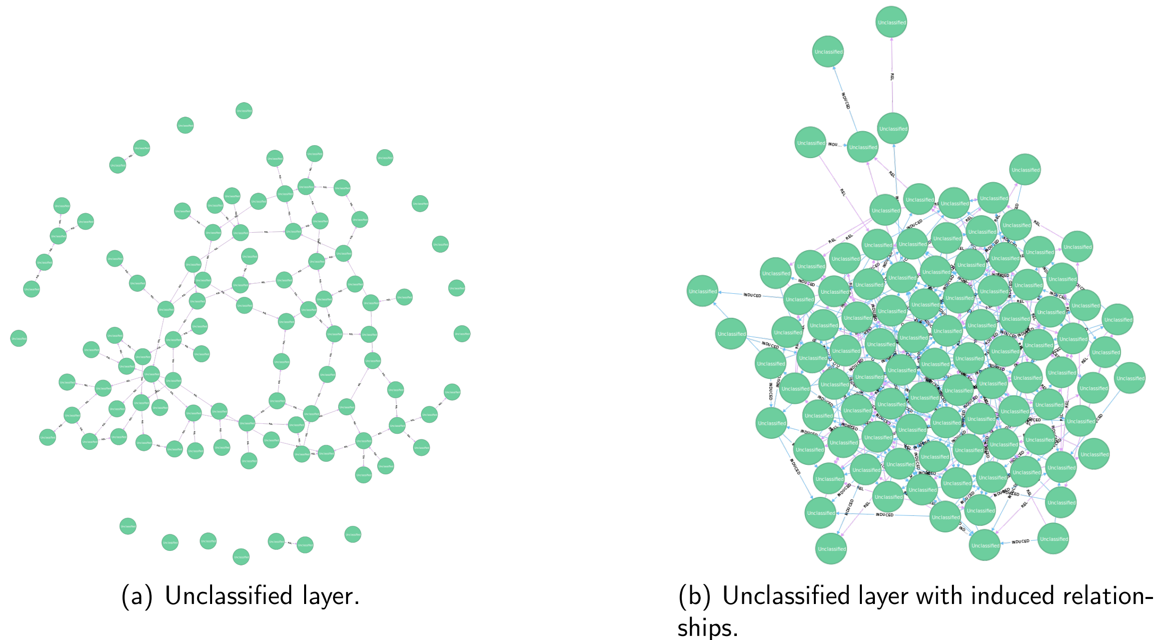
```
MATCH (n {security: "Unclassified"}) -[]->
    (m {security: "Top Secret"}) -[]->
    (p {security: "Unclassified"})
CREATE (n) -[r:INDUCED]-> (p);
```

These two commands added 201 relationships to the Unclassified layer (107 with the Top Secret layer and 94 with the Secret layer) increasing the number of relationships in that layer from 101 to 302. The graphs for the Unclassified layer can be seen in Figure 6.2 both with and without the induced edges.

The graph of the Unclassified layer in Figure 6.2(b) is much more connected than its counterpart without the induced relationships. Adding the induced relationships reduced the number of connected components of the graph from 17 down to one, meaning that all the nodes in the induced relationship graph are connected to each other vice some nodes being unreachable in the original graph.

In addition, the clustering coefficient, a measure ranging from zero to one of how connected

a graph is based on the number of triangles formed by the edges [10], of the original Top Secret layer is zero. The graph with induced relationships, on the other hand, has a clustering coefficient of 0.029, showing that the connections between the nodes is much higher if the user is able to look for those connections outside of a single layer.



(a) Unclassified layer.

(b) Unclassified layer with induced relationships.

Figure 6.2: Unclassified layer as rendered in Neo4j. (a) has only the original trivial relationships that exist in the Unclassified layer. (b) contains all the original relationships with induced relationships added to the graph.

When the induced edges are created across the entire graph, an additional 575 edges are produced (186 in the Top Secret layer and 188 in the Secret layer). The nodes in this random graph are much more connected than a user restricted to the relationships contained within a single level. Allowing the inclusion of induced relationships in the graph database allows the user better access to connections between data points while restricting him to his authorized security level.

6.6 Real World Graph Analysis

To better explore induced relationships, a real directed network was partitioned and examined. The network used was a modified version of links between Google's web pages [19]. The original data set contained 15,763 nodes and 171,206 relationships.

To make the data set more manageable, two large sections were removed: international and university. These sections were removed both because they were extremely large and easy to extract and because they functioned as separate mini-Google environments that were highly connected within themselves, but not very connected across the full set of web pages.

Additionally, a few comparatively high degree nodes were removed. Google's home page, for instance, had over 10,000 connections while most pages possessed 200 or less. These high degree nodes would cause a large number of induced relationships; they were removed to make the degree distribution more equitable across all nodes. The resulting data set contained 1,125 nodes and 16,941 relationships.

The web pages were instead split into three groups that were arbitrarily assigned classification levels of Unclassified, Secret, and Top Secret. An attempt was made to use the *Louvain* method [20] to find natural groups within the data set, and to use those communities to simulate classification levels. The community detection algorithm, however, returned one large community and several much smaller ones that would not conform to the three-security-level model.

Divisions were, instead, made alphabetically on the basis of the Uniform Resource Locator (URL) under the assumption that pages who's URLs begin with "http://www.google.com/support" are more likely to be connected to, and as a corollary be classified in the same level as, web pages that have the same URL beginning. The groups sizes were 411 nodes in the Top Secret level, 343 nodes in the Secret level, and 353 nodes in the Unclassified level.

Searching for induced relationships on the Unclassified level in the Google web page database returns an additional 5,210 relationships (240 passing through the Top Secret level and 4,970 passing through the Secret level). Because the security levels were arbitrarily assigned, any one of them could be assigned the Unclassified security label. Thus, the induced relationships between all levels are equally relevant. The induced relationships for the other levels are: Top Secret-455 (141 with the Secret level and 455 with the Unclassified level) and Secret-1,824 (264 with the Top Secret level and 1,562 with the Unclassified level).

The density of a graph is a ratio of how many edges exist to the total number of possible edges for a given number of nodes. For a directed graph, this calculation includes the possibility for two distinct edges between every pair of nodes, one in each direction. The density of induced edges created can be computed according to:

$$D = \frac{|E|}{|V|(|V|-1)} \tag{6.1}$$

where $|E|$ is the number of induced edges that were created, and $|V|$ is the number of vertices in the security level. The density of the Top Secret layer is 0.0027. The density of the other two layers, with more induced edges is higher: Secret-0.016 and Unclassified-0.042.

In all cases, the density of the layers increased with the addition of the induced relationships. The Unclassified layer density increased from 0.012 to 0.054. The Secret layer density increased from 0.016 to 0.031. And, the Top Secret layer density increased from 0.058 to 0.061.

By way of comparison, a synthetic graph was generated [17] using an Erdos-Renyi [18] model where the number of nodes and the number of edges are both specified. These values were set to match the real world graph (1,125 nodes and 16,941 edges). The nodes were then split into three equal groups to simulate the Top Secret, Secret, and Unclassified security levels. This configuration created approximately 19,00 induced relationships for each level producing an induced edge density of approximately 0.15 for each security level, an approximate density increase of 0.040 to 0.18 for each level.

Although the density of induced relationships and the density increases are lower for the real network than for the synthetic graph, this outcome was expected due to the more *Poisson* degree distribution typical of synthetic graphs as opposed to the more power law degree distribution characteristic of real world networks. The higher number of moderately connected nodes in the synthetic graph would lead to more connections traversing security levels simply because a higher degree on a random graph increases the likelihood that a node at the other end of one of the relationships is in another security level.

The densities for the real network increased by almost 100% in one case and by 450%

in another. A graph database with a granular security implementation that gives users access to induced relationships will show the connections between data points more than a completely isolated graph database, which allows no interaction between different security levels.

CHAPTER 7:
Conclusion and Future Work

This thesis examined the feasibility of implementing granular security onto a graph database looking specifically at retaining the ability of the graph to connect data points. Neo4j was used as the actual graph database for both providing a concrete basis for implementation goals and as a testing ground for the functionality of the algorithms.

7.1 Conclusions

After introducing the current state of database models and the lack of granular security features on those models, the general security models and data management were discussed. The concept of granular security for a database was introduced.

A discussion of graphs and graph databases then followed to give the reader enough working knowledge of the topics to understand the subsequent research. The basics of granular security implementation on the database nodes was discussed next, followed by the introduction of the complication to implementation provided by relationships along with the three main types: trivial, restricted, and induced. A proof of concept for implementing granular security on Neo4j was also briefly demonstrated.

Details of the possible methods of mathematical justification were then described and analyzed appropriate. Graph theory concepts, which were determined as not helpful to the implementation of granular security include hypergraphs, weighted graphs, temporal graphs, colored graphs, and multiplex graphs. Multi-slice graphs and the adjacency matrices of the resulting bipartite graphs were used to demonstrate a mathematical justification for generating induced relationships.

The method for generating induced relationships as well as the correlated query in Neo4j were performed on a synthetic network and shown to produce equivalent results. The induced relationships generated in a single security layer of the synthetic graph were shown to better show the user the existing connections between nodes in the database.

The induced relationships were also generated on a real world network with artificial partitions. The density of each layer of a graph showed marked increase with the addition of induced edges indicating that including those relationships in a granular security implementation shows data connectivity better than simply partitioning security layers without including the induced relationships.

7.2 Future Work

The scope of this thesis is limited to provide an introduction to the proof of concept of implementing granular database security on a data set with security partitions. The full concept would need to be further explored in the following areas prior to drawing any complete conclusions about the implementation of granular security.

7.2.1 Aggregating Information

Related to Section 7.2.2 is the idea of critical information as it relates to Operations Security (OPSEC) where several pieces of information at one classification level can be combined to raise the classification level of the aggregated data. Determining a method for addressing when these combinations of critical information exist and induced relationships should not be drawn is crucial to implementing granular security.

7.2.2 Longer Induced Relationships

The induced relationships examined in the present work were simple two-hop paths that traversed to another security level and immediately back to the user's security level. Paths that hop to another security level, then hop to another node in that same security level before hopping back to the user's security level, as shown in Figure 7.1(a), are the most basic version of a longer induced relationship.

These relationships can go beyond this basic form to a path that crosses into multiple other security levels before returning to the user's security level, as shown in Figure 7.1(b). Ultimately, the induced relationship construct must address the N-Hop induced relationship, illustrated in Figure 7.1(c), where a path departs the user's security level and traverses an unknown path of unknown length before returning to the original security level.

The mathematical justification of finding these traversals is likely similar to the bipartite matrix multiplication as discussed in Chapter 5. However, the benefit from including all of these relationships versus the security of the database from including them must be further examined and understood.

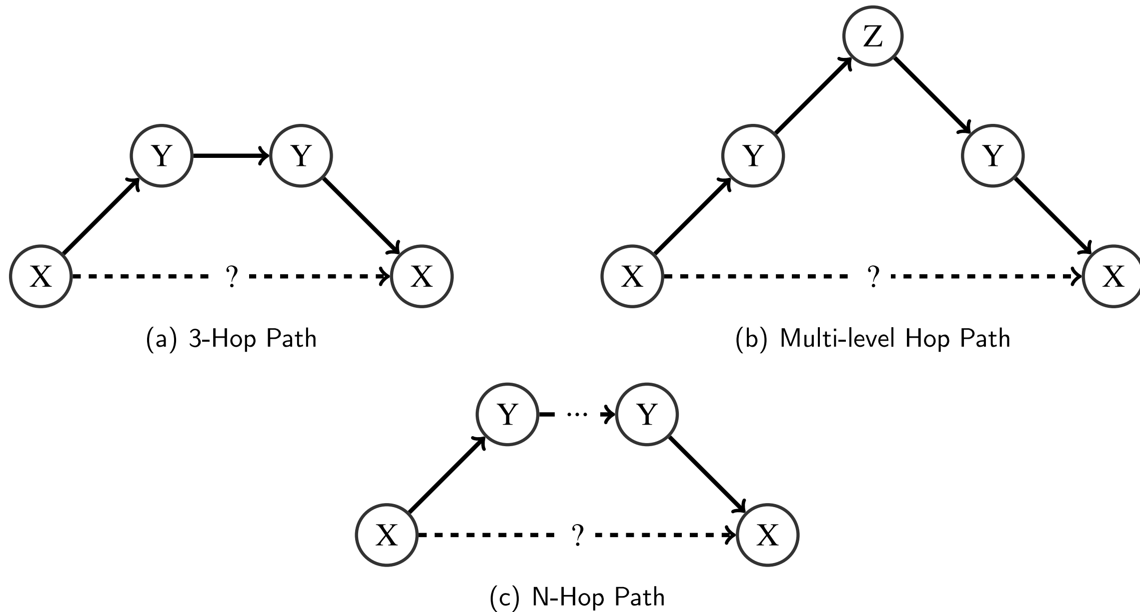


Figure 7.1: Longer induced relationship examples.

7.2.3 Hierarchy and Compartments

This thesis examined the detection and production of induced relationships in a compartmentalized structure and showed that the concept was trivially applicable to a hierarchical security environment. In the actual DOD security structure hierarchical and compartmentalized models are combined; compartmentalized partitions exist within the hierarchical levels. The implementation of granular security in a graph database would need to be able to adequately address the combination of the two security structures.

7.2.4 Testing on Secure Data Set

Lastly, this thesis examined the proof of concept by implementing granular security on both a synthetic graph as well as a real world data set. Even the real world network, however, had the security levels artificially added. Thorough research on granular security

would need to examine how an actual classified data set would respond to the production of induced relationships taking into account all the areas for further exploration discussed in Section 7.2.

List of References

[1] A.-L. Barabási and J. Frangos, *Linked: The New Science of Networks*. New York, NY: Basic Books, 2014.

[2] K. Driscoll, "From punched cards to 'big data': A social history of database populism," *Communication+ 1*, vol. 1, no. 1, p. 4, 2012.

[3] D. E. Bell and L. J. La Padula, "Secure computer system: Unified exposition and multics interpretation," DTIC Document, Tech. Rep., 1976.

[4] K. J. Biba, "Integrity considerations for secure computer systems," DTIC Document, Tech. Rep., 1977.

[5] G. Chartrand and P. Zhang, *A First Course in Graph Theory*. Mineola, NY: Courier Corporation, 2012.

[6] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Computing Surveys (CSUR)*, vol. 40, no. 1, p. 1, 2008.

[7] R. Van Bruggen, *Learning Neo4j*. Birmingham, UK: Packt Publishing Ltd, 2014.

[8] Neo4j-Team, *The Neo4j Manual v2.3.1*. San Mateo, CA: Neo Technology, 2015.

[9] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*. Sebastopol, CA: O'Reilly Media, Inc., 2013.

[10] M. E. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, no. 2, pp. 167–256, 2003.

[11] P. Holme and J. Saramäki, "Temporal networks," *Physics Reports*, vol. 519, no. 3, pp. 97–125, 2012.

[12] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter, "Multilayer networks," *Journal of Complex Networks*, vol. 2, no. 3, pp. 203–271, 2014.

[13] M. De Domenico, A. Solé-Ribalta, E. Cozzo, M. Kivelä, Y. Moreno, M. A. Porter, S. Gómez, and A. Arenas, "Mathematical formulation of multilayer networks," *Physical Review X*, vol. 3, no. 4, p. 041022, 2013.

[14] M. De Domenico, A. Solé-Ribalta, S. Gómez, and A. Arenas, "Navigability of interconnected networks under random failures," *Proceedings of the National Academy of Sciences*, vol. 111, no. 23, pp. 8351–8356, 2014.

[15] P. J. Mucha and M. A. Porter, "Communities in multislice voting networks," *Chaos*, vol. 20, no. 4, p. 041108, 2010.

[16] T. J. Fararo and P. Doreian, "Tripartite structural analysis: Generalizing the breiger-wilson formalism," *Social Networks*, vol. 6, no. 2, pp. 141–175, 1984.

[17] R. Gera, "Naval postgraduate school network discovery visualization project," http://faculty.nps.edu/dl/networkVisualization/, July 2015.

[18] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hungar. Acad. Sci*, vol. 5, pp. 17–61, 1960.

[19] G. Palla, I. J. Farkas, P. Pollner, I. Derenyi, and T. Vicsek, "Directed network modules," *New Journal of Physics*, vol. 9, no. 6, p. 186, 2007.

[20] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California