Finding Bounded-Length Cycles in Decentralised Networks under Privacy Constraints

Consumer-Friendly Transaction Monitoring

CSE3000: Thesis Project

Juno Jense

Finding Bounded-Length Cycles in Decentralised Networks under Privacy Constraints

Consumer-Friendly Transaction Monitoring

by

Juno Jense

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday July 11th, 2024 at 01:30 PM.

Student number:	4565541	
Project duration:	November, 2023 – July, 2024	
Thesis committee:	Dr. Z. Erkin,	TU Delft, supervisor
	F. W. Dekker	TU Delft, daily supervisor
	Dr. J. G. H. Cockx	TU Delft

TU Delft

Preface

My curiosity manifested itself while I was doing my bachelor's degree, during a group project on multi-party computation. The supervisor for the project ended up becoming the supervisor for this thesis. During the master's degree I started following more and more courses on security. I was most captivated by cryptography, and security of programming languages. After following a course on privacy-enhancing technologies, I realised I wanted to do a thesis on cryptography and privacy.

First, I would like to thank my supervisor, Dr. Zeki Erkin, for exposing me to cryptography all those years ago, and their continuous support during the past nine months. Thanks to their guidance and feedback I developed a passion for cryptography. Their expertise and enthusiasm encouraged me to learn as much as I did, far exceeding my expectations. I am especially thankful for the constant encouragement, positivity, and setting a high standard without imposing much stress. This helped me tremendously in setting goals and boundaries for myself, and gave me the confidence to overcome my perfectionism.

Next, I want to express my appreciation and gratitude towards my daily supervisor, Florine W. Dekker. There has not been a single time where they were not ready to discuss and challenge my ideas. Each discussion left with new insights and gave me motivation to continue my research. My sincere thanks also goes out to the PhD students Jelle Vos, Jorrit van Assen, and Tianyu Li, as well as the master students Chelsea Guan, Davis Sterns, Vojta Crha and Prakhar Jain, who contributed a significant amount to my thesis experience.

I am thankful towards my friends who shaped my life as a student in Delft, and towards my family. Special thanks go out to my partner, Lukas, and to my parents. Their unconditional love and support is what got me this far.

Juno Jense
Delft, July 2024

Summary

Financial crime has seen a surge in complexity over the past decades as a result of digitisation. This required better tools for detecting financial crime, creating a cat-and-mouse game between law enforcement and criminals. Anti-money laundering (AML) is the collective term describing these tools, as well as financial regulations designed to combat money laundering. The detection of fraud is done through data analysis. Typically, the financial data is a large set of transactions between accounts. Transactions have a sender, receiver, and an amount that is transferred from sender to receiver. This can be represented by a directed graph where nodes represent accounts and edges transactions between accounts. Methods for finding fraudulent transactions in transactional graphs are widely studied in both literature and industry. This usually assumes a centralised entity with full access to the data. At an international scale, the centralised approach is not suitable. Besides its massive scale posing a challenge, it is hard to imagine a global agreement on what capabilities such an entity should have. At the same time, organisations sharing financial data for analysis raises privacy concerns. We limit our scope to transactional cycles. These typically occur when criminals send money through a series of international bank accounts back to the original account. This lowers the chance of the money being traced back to its illicit source.

We propose a privacy-preserving decentralised protocol which takes as input a parameter ℓ from any node and outputs all cycles of at most length ℓ containing the node. The protocol relies on the decisional Diffie-Hellman assumption, and we show the protocol is secure against a polynomially bounded adversary within our security model. The communication complexity of an instance is $O(n^\ell)$, multiple instances can be ran in parallel. We implement the protocol and measure its performance on scale free graphs in terms of communication and computational costs. Lastly, we discuss our results and suggestions for future work.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
ABB	Arithmetic Black Box
AML	Anti-Money Laundering
AMLA	Authority for Anti-Money Laundering and Countering the Financing of Terrorism
DDH	Decisional Diffie-Hellman
DFS	Depth-First Search
DLP	Discrete Log Problem
EU	European Union
FATF	Financial Action Task Force
FIU	Financial Intelligence Unit
GDP	Gross Domestic Product
MPC	Multi-Party Computation
ORAM	Oblivious Random Access Memory
PPT	Probabilistic in Polynomial Time
TTL	Time-To-Live
UB	Unbounded

Symbols

Symbol	Definition
ℓ	Hop Distance, TTL, Search Depth
G	Graph $G = (V, E)$
V	Vertex Set
E	Edge Set
v	Vertex v, Node v
(v_i, v_j)	Edge v_i to v_j
n	Total Nodes in G, Size of V
e	Total Edges in G, Size of E
π	Path $(v_1, \ldots, v_k)$ of Length k
σ	Cycle $(v_1, \ldots, v_{k-1}, v_1)$ of Length k
$N(v)$	Neighbours of v
$N^+(v)$	Outgoing Neighbours of v
$N^-(v)$	Incoming Neighbours of v
d, d_i	Degree, Size of $N(v)$, $N(v_i)$
d_{in}	Indegree, Size of $N^+(v)$
d_{out}	Outdegree, $N^-(v)$
p	Prime (1024-3072 bits), Modulus
q	Prime (160-256 bits), Order
r	Integer (864-2816 bits)
h	Integer $(0 < h < p)$
g	Generator
$\kappa_p, \kappa_q, \kappa_r$	Security Parameters for Public Keys, Nonces
$\mathbb{Z}_p^*$	Multiplicative Group of Integers Modulo p
$\mathbb{G}_q$	Cyclic Subgroup of $\mathbb{Z}_p^*$ with order q
g^x	Public Key
$x, x_i, x_{i,j}, y$	Secret Key
$k_{i,j}$	Partial Key
$r_i, r_{i,j}$	Nonce
m, m', m''	Message
`keys`	Established Keys
`pending`	Partial Keys Awaiting Response
`routes`	Tuple with Key and Two Pairs (Nonce, Node)
`retrace`	Tuple with (Nonce, Key, Key) and (Nonce, Node)
$\mathcal{A}, v_A$	Adversary, Adversarial Node
M	All Messages over Multiple Instances
M_A	Messages in M Received by v_A
M_f, M_e, M_t	Message Set (Forward, Echo, Trace)
m_0, m_1	Parameters Scale-Free Graphs Generation
c	Total Cycle Edges $\Sigma\|\sigma_i\|$
t_{avg}	Average Execution Time
σ_{all}	All Cycles in the Graph

1 Introduction

Criminals obfuscate funds obtained through illicit means by running them through a complex series of financial transactions. By integrating their illegal proceeds into the legitimate banking system, they can disguise the true nature of their activities and make their money appear to be from legal sources. The Russian invasion of Ukraine and its associated funding are a prominent example showcasing the scale and global impact of money laundering. Russia has an estimated $1 trillion hidden abroad, of which a quarter is controlled by Putin and his associates [8]. The origin of these funds often involve the extortion of private businesses, or are straight up stolen from state budget [40]. Once the money is laundered and moved abroad, it can be used for espionage, propaganda, bribery, and other nefarious purposes [11], while keeping the true origin hidden.

Anti-money laundering (AML) is an umbrella term for the laws, regulations and procedures which are designed to combat the generation, concealment, and integration of illicit funds. Financial institutions are obliged to proactively monitor and report suspicious activity to the authorities. Detection of money laundering requires a sophisticated approach, as financial data is under heavy regulation. At the same time, organisations may want to share data, such as suspicious transactions, for a joint analysis. Due to the sensitive nature of the data, the exchange should reveal no more private information than strictly necessary. For this reason coordination between financial organisations is not straightforward, and remains unconventional within the industry [46]. Criminals exploit this by setting up accounts at different banks across multiple countries to avoid getting caught by law enforcement agencies. Since countries and their governments are largely independent in the creation of their legislative policies that regulate the exchange of such data, expecting them to consistently report to a central authority is unrealistic. Performing international data analysis, without sacrificing confidentiality nor the autonomy of countries, requires a privacy-preserving and decentralised approach.

1.1. Emergence of Money Laundering

The use of the phrase *money laundering* for describing the process in which criminals obscure their earnings has its roots in the 1970s, when a major political scandal in the United States resulted in the resignation of former president Nixon [60]. In the scandal, a large amount of cash was deposited in Mexican banks, which was later transferred back to the United States, thus hiding the origin of the money. At the same time, criminals were setting up feign businesses which seem legitimate, which were used to make their winnings appear to be from legal activities. Back in 1989, the financial action task force (FATF) was formed as a response to the increase in money laundering. The task force consisted of 16 members at time of founding and was given responsibility to examine and combat money laundering techniques [28]. The standards they set out became more relevant as a result of the terrorist attacks which took place in 2001; their mandate was expanded to combat terrorist funding [29]. Since then, an increasing number of countries made efforts to collaboratively tackle money laundering, and the FATF now consists of 39 members. At the same time, threats evolved further and traditional systems started falling behind [52], which led to regulations becoming more strict.

The regulations of the European Union require financial institutions to report suspicious activity to the financial intelligence unit (FIU) of their respective country [25][26]. These units operate independently from organisations which are subject to AML reporting, and are typically integrated into law enforcement agencies or governmental administrative bodies. Based on these reports, information might be relayed to the authorities if necessary. This suffices at a national level, however, fraudulent chains of transactions often span multiple countries [41]. For this reason, the European Commission has made significant efforts to improve cooperation and coordination between FIUs. The FIU.net project [25] is one of such efforts. Initially funded by the European Commission since 2002, the project facilitates secure international exchange of financial intelligence between EU member states. Its underlying technology relies on ma^{3}tch [9][39], an anonymous type of analysis. Ma^{3}tch makes use of an anonymisation algorithm which takes as input a set of sensitive records, and outputs an extremely minimised filter which captures its characteristics without leaking any sensitive information. To achieve this, ma^{3}tch leverages space efficient probabilistic data structures, hashing, fuzzy logic and approximation techniques. This procedure produces a uniform anonymised filter, and is also referred to as *hashing the hash*. While this seems promising, the ongoing efforts to standardise cross-border dissemination of suspicious transaction reports via FIU.net have not yet led to a wide adaptation across EU member states [46]. This is a worrying statistic, especially since the past decade has seen a surge in money laundering and terrorist financing [52].

Cost of Compliance

To combat this the EU introduced numerous new anti money-laundering directives between 2015 and 2022 [25]. Alongside these rapid developments, a survey by Reuters done in 2017 showed that the cost of maintaining compliance has been growing as well [36]. Their results show that organisations struggle to keep up, and the degree to which compliance is maintained varies across organisations. Naturally, individuals partaking in illegal activities aim to minimise their chances of being caught, and set up accounts at banks which provide them with the highest chances of staying undetected. For example, investigations into which countries are most involved in money laundering schemes have shown that there is a clear skew towards banks established in eastern European countries [41]. Detecting money laundering is a complex task, especially due to its massive scale. The United Nations Office on Drugs and Crime estimated that between 2 and 5% of the global GDP is laundered each year [22]. Not only are a lot of funds involved, the transactions are also distributed across many parties; in Europe alone there are well over five thousand banks registered [57]. Currently, transactional data is only shared between two FIUs if the transactions are flagged as suspicious [46]. In an ideal world, we are able to include all cross-border transactions, from more than two parties, in one single analysis. This analysis would bring cases of fraud to light with high accuracy, without loss of privacy for individuals not involved.

1.2. Transaction Graphs

The amount of financial data which is generated each day has been steadily increasing. As a result, big data technologies and artificial intelligence contribute to better accuracy in detecting fraud compared to traditional approaches, and became a central part in modern AML systems [16]. Even though every system is different, they do share a common goal: distinguishing fraudulent transactional patterns from legitimate ones. Money laundering has a consistent definition, and consists of three stages: placement, layering, and integration. In the placement stage, illicitly obtained assets are obtained from or placed into a set of source accounts. As an example, banking credentials may be stolen, or criminals may run a business and forge financial documentation to make illicit proceeds seem like legitimate profit earned through the business. The next phase, layering, consists of transactions that have no purpose besides hiding the true source of funds. The last stage integrates the assets into the legal economy, such that other investments and purchases can be made [56]. The focus of this thesis is on the central component of money laundering, the layering stage.

Transactional data can be modelled as a graph, such that nodes represent accounts and the edges the flow of funds from one party to another. The layering stage can be defined using the same model. Funds originate from a set of source nodes, are moved through one or more middle accounts, before eventually being deposited into a set of target accounts. In practice, source nodes are mules or compromised victim accounts, middle accounts are mules or feign businesses, and targets have high spendings and may invest in legitimate assets. To give a concrete example, Starnini et al. [56] use this representation to identify two subtypes of money laundering known as smurfing. This is a money laundering technique in which large sums of money are broken up into multiple small transactions. The first smurf-like motif they identified has multiple nodes in its source and target sets, which are all connected through the same singular middle node. The second pattern has only a single source and single target node, both which are connected through multiple middle accounts.

Cyclic Cash Flow

Laundering activities often produce special subgraphs in the transactional graph representation. Some elementary fraudulent patterns are cliques, stars, and cycles. While during the past few decades schemes have taken on more sophisticated forms, transactional cycles in particular remained as a strong indication of potential money laundering [35]. For a more extensive overview of fraud patterns, we refer to the overview provided by Dumitrescu et al. [23] in their work on anomaly detection. In this thesis, we study the problem of privacy-preserving cycle detection in distributed networks. Cycles represent a flow of cash in which one account, belonging to the criminals, is both the source and target node. Middle nodes are legitimate accounts used to hide the illicit origin of funds. Nearly all modern laundering schemes involve criminals recruiting money mules; non-involved individuals who give access to their account for a monetary reward. Another popular strategy that is often employed is the use of feign businesses to make intermediate accounts seem legit.

European and Commercial Instruments

Tighter regulations have led to financial crime detection becoming a high priority among organisations subject to AML audits. GraphS [50], a tool developed by the e-commerce platform Alibaba, is one such large-scale industry solution based on cycle detection. The system monitors the graph of all transactions and identifies bottlenecks for its optimisation. Even though the system is not suitable for sensitive financial data, they did make an important observation on cycle properties. They found that cycles rarely exceed a length of six, and have a near-constant value across their transactions. Hajdu et al. [35] developed a cycle-based fraud detection system for a Hungarian bank, and observed the same property while performing experiments on real financial data. Additionally, they found short cycles to be extremely uncommon in legitimate transfers and are thus strong indicators of fraud. Nearly all money laundering schemes involve more than one organisation, and almost always span across multiple countries [41, 26]. Such schemes are hard to detect because the transactional data does not belong to a single central entity. The recent collaboration between five Dutch banks addresses this issue at a national level by anonymising and aggregating transactions to a single database. Not only did this raise controversy and privacy concerns [48], its multi-year development is also associated with a high cost [47]. This type of approach becomes an even bigger challenge at an international scale. To mitigate risks of cyberattacks, privacy invasion, and potential abuse of power, international monitoring has to be decentralised across multiple parties.

Current Status

In Europe there have been ongoing efforts to standardise cross-border cooperation through introducing a decentralised network, which FIUs of member states are legally obliged to use. Each FIU manages their own database containing suspicious transactions, and share information proactively or by request. Yet, as this imposes a high workload on the FIUs and requires them to take a proactive stance, many reports are never forwarded to other member states. As a result, the lack of cooperation persists [46]. Since January 2016, Europol has been made responsible for maintaining the FIU.net project in an attempt to improve the situation. They integrated FIU.net into their existing AML and terrorist funding monitoring landscape [1]. While the intent was to create more synergy between financial intelligence, a 2019 report [27] from the European Data Protection Supervisor concluded that Europol failed to comply with data protection regulations. At the same time, in an effort to further enhance collaboration between member states, the European Commission passed a proposal to establish AMLA, an authority for anti-money laundering and countering the financing of terrorism [26, 17]. This authority is decentralised across European member states, and is purposed to enhance coordination between the respective national authorities of members participating. One of the responsibilities of this authority will be hosting the FIU.net infrastructure, while respecting data protection laws. Its activities are expected to start mid-2025 [18], and reach full staffing in 2027.

1.3. Research Objective

The current system in Europe does not make use of graph data and only compares two transactions at a time. To find more sophisticated patterns such as cycles, especially in the upcoming decentralised setting, a different approach is needed. Expressing the European transactions in the graph model produces a rather large transactional graph. Nodes and transactions are partitioned among the participants, or in graph terminology, participants have partial knowledge of the topology. The full topology is not known though, nor do participants know which external accounts may have been flagged for suspicious activity. By definition decentralised networks have autonomous nodes which exchange information with only their neighbours. For a network of financial intelligence organisations to be considered decentralised each party must be honest, does not perform centralised computation, and independently controls each node as unique entity. We give an example of a decentralised network in Figure 1.3 which is partitioned between two banks. In practice, any decentralised protocol can be made suitable for a network in which any number of overarching entities may own multiple nodes. The required computational power can be significantly reduced if initiating is only done over edges that have their respective nodes in different partitions. For a set of nodes belonging to only one organisations standard tools can be used to find the transactional cycles. Besides, criminals typically employ cross-organisation strategies for their schemes. From the decentralised perspective, nodes only have knowledge of their direct neighbours. Privacy implies that topology unrelated to the node has to stay hidden, unless a cycle is found and shared among the network. The ability to infer existence of transactions between other accounts leads to leakage of sensitive information.

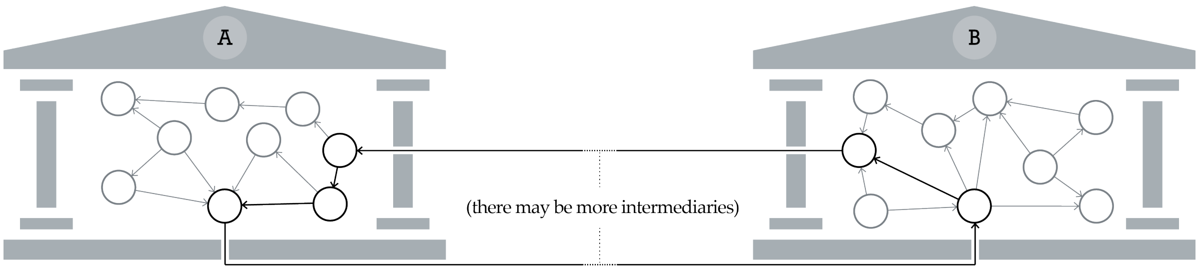

Figure 1.1: Decentralised view of a transaction cycle between accounts in different banks.

We consider a generalised setting in which an arbitrary real-life application is modelled as a decentralised directed graph. Besides finance, a few examples of where this model can be applied are transport, logistics, and social networks. Work on privacy-preserving cycle detection in the decentralised setting is limited. To the best of our knowledge, only the work of Porsius [49] achieves similar goals without scaling quadratically on the number of vertices in the graph. However, their proposed protocol is vulnerable to attacks as it re-uses a one-time pad. We show how an attack on this protocol can be constructed in chapter 3. This leads us to our main research question:

"How do privacy and efficiency scale when finding bounded length cycles in decentralised networks?"

Current literature on financial fraud detection can be classified into subgraph detection, graph queries, statistical functions and learning approaches. To briefly elaborate, subgraph detection targets special topologies such as stars, cycles, cliques and various dense subgraphs. For graph queries two subtypes are considered, the point-to-point reachability and shortest path queries. Statistical functions use the global distribution of one or more features to compute a normality measure over neighbourhoods. This measure is used to assess whether a particular neighbourhood shows anomalous behaviour. Lastly, approaches based on learning optimise a global objective function that characterises deviation from the norm. Learning algorithms build a generalised model on historic data which can predict or classify unseen transactions. This abstraction removes the need to manually define rules and makes learning the most dissimilar to our approach. Our work can be considered as a decentralised approach to subgraph detection.

1.4. Contribution

Our motivation is in part due to a discovered vulnerability in the current state-of-the-art. Therefore, our first contribution is an adversarial construction on an existing protocol. This steered us to our primary contribution; a novel privacy-preserving decentralised protocol which detects cycles in the local neighbourhood of nodes. The remainder of our contribution consist of a security definition and subsequent proof, followed by a performance evaluation of our implementation of the proposed protocol.

1. We highlight a security vulnerability in the current state-of-the art. We first construct a scenario in which an honest-but-curious adversary can deduce which messages belong to the same instance. Next, we show how this information can be used to imply the existence of non-neighbouring edges. Lastly, we show how a significant part of the graph topology can be obtained when the adversary controls multiple nodes.
2. We propose a protocol based on the Diffie-Hellman key exchange. In this exchange, two parties establish a shared secret over a public network, which can be used to encrypt communication after the initial exchange. We extend this concept to more than two parties by involving the local neighbourhood of nodes. As nodes are autonomous, the protocol is ran at node-level and can be ran in parallel. Nodes interested in learning which cycles they are in perform a key exchange with each node that it can reach within a specified hop distance ℓ. The intuition behind our protocol is that an initiating node exchanges a key with itself iff it is in a cycle of at most length ℓ. The output for a particular instance is the set of all cycles of at most length ℓ the initiating node of the instance is part of.
3. After stating our cryptographic assumptions, we define two objectives that form the basis of our security definition. First, we prove that an adversary cannot derive whether any pair of public keys belong to the same instance. Secondly, we consider an adversary controlling multiple nodes, and prove that the topology unknown to the adversary remains hidden using a formulation based on node contraction. Finally, we show these properties hold in the context of our protocol such that our privacy requirements are fulfilled.
4. We provide theoretical worst-case upper bounds on communication, computational, and storage complexity. Following this, we implement the protocol and run experiments to evaluate its performance. We use the resulting empirical data to verify the efficiency of the protocol for low values of ℓ, and discuss the execution time and total exchanged messages in relation to the average degree and the number of cycle edges.

1.5. Outline

The remainder of this thesis is divided as follows. In chapter 2 we formally introduce the preliminaries and notation. Chapter 3 provides an overview of existing techniques for cycle detection and adjacent related work. We propose our solution in chapter 4 and describe its various components. Next, we provide a formal security argument and share the results of our experiments in chapter 5. Lastly, we discuss our findings in chapter 6 and provide some concluding remarks.

2 Preliminaries

This chapter we introduce the notation, definitions and concepts used throughout the remainder of this thesis. First, the graph model and its components are introduced. Next, we discuss graph problems related to finding cycles and a closely related problem, finding shortest paths in graphs. We then turn to properties of decentralised systems, and discuss the aforementioned graph problems in the decentralised context. We continue by introducing multi-party computation, our mathematical foundation, and cryptographic primitives.

2.1. Graph Theory

We denote a graph G as the pair $G = (V, E)$, where V contains elements, and E contains edges which connect elements. We refer to the elements of V as vertices or nodes. The solution is a general algorithm not specific to the financial setting, nodes and edges can be thought of as accounts and transactions in spirit. Within our framework G is a simple directed graph. By this definition, given two vertices $v_i, v_j \in V$, the edge $(v_i, v_j) \in E$ connects a source v_i to target v_j. The term *simple* restricts G from containing self-loops (i.e. $i = j$ for $(v_i, v_j) \in E$) and ensures there are no two edges with the same source and target nodes. The set of vertices which a node $v_i \in V$ shares edges with are called neighbours, and is given by $N(v_i) \subset V$. The size of this set is equivalent to the degree d_i of v_i. We denote $N^+(v_i)$ as the neighbours to which v_i has an outgoing edge, and use $N^-(v_i)$ for neighbours v_i has incoming edges with. The respective sizes of these sets are the outdegree and indegree of the node. The path π is a set of k connected vertices, where $\pi = (v_1, v_2, \ldots, v_k)$ for distinct vertices $v_1, v_2, v_k \in V$. Cycles of length k are a type of path denoted by $\sigma = (v_1, \ldots, v_{k-1}, v_k)$ such that $v_1, v_2, v_{k-1} \in V$ are distinct vertices and $v_1 = v_k$ holds. We use $\pi(i)$ (or $\sigma(i)$) to refer to the edge $(v_i, v_{i+1}) \in E$.

2.2. Cycle Detection Algorithms

Functions that find cycles play an important role in the domain of graph theory. The most primitive implementation takes as input a graph $G = (V, E)$ and outputs true if a cycle exists within the graph, and false otherwise. Extended variants of the algorithm may be given an upper bound on the cycle length, or search for cycles with certain characteristics. For example, typical search algorithms can be adapted to detect cycles containing a particular node. Depth-first search (DFS) is a well-known algorithm which takes as input a root node and target node. Starting from the root node, it explores as far as possible along each branch before backtracking. This process is repeated until the target node is found or no more paths can be traversed. Cycles can be detected by starting the search with the same root and target node. Graph algorithms which can only detect the presence of a cycle are useful when the cycle is not a relevant output. Finding the actual vertices and edges that make up the cycle requires a more advanced class of algorithms which generally have a higher run-time.

Methods that find strongly connected components implicitly find all cycles in the graph. Directed graphs are said to be strongly connected if each vertex can be reached from every other vertex, which is a property that holds for all cycles. The only strongly connected components that are not cycles are nodes in isolation, with a degree of zero. These similarities are reflected in the overlap between the approaches used for cycles and strongly connected components. For example, Tarjan's algorithm [58] partitions the graph into its strongly connected components using an approach based on depth-first search. Nodes are assigned a unique incremental index and are placed on a stack in the same order in which they were visited. Additionally, nodes each initialise a *lowlink* variable equal to their index indicating their smallest reachable index. Each time a new node is visited its non-visited descendants are recursed on. After all neighbours of a node have been visited, its lowlink value is compared to its index. If these values are identical, the node is part of a strongly connected component and becomes a root. The connected component is then popped off the stack and the call is returned. The algorithm terminates if all reachable vertices have been explored.

2.3. Shortest Path Algorithms

The shortest path problem in graph theory involves determining the path between a pair of vertices that minimises the total weight of its edges. For paths that do not exist in the graph the length is typically set to infinite. The presence of a cycle can be derived from the output by checking whether a shortest path exists with the same source and endpoint. The shortest path in unweighted graphs is the path containing the least amount of edges, which is equivalent to a weighted graph with weight equal to 1 for each edge. For scenarios where the objective is to find more than one shortest path, one could naively run the same algorithm for each node pair. However, there are specialised algorithms that use significantly more efficient methods. Many solutions have been proposed in literature, however, we are only interested in algorithms for which privacy-preserving counterparts exist. These are Dijkstra, radius stepping, and Floyd-Warshall. Dijkstra's algorithm is a well-known algorithm for finding the shortest path from a given node $v \in V$ to all other nodes in the graph. The algorithm follows an iterative bottom-up approach [21]. The first iteration of Dijkstra marks the shortest path to each node as infinity, and marks the starting node as visited with distance zero. Then, at each subsequent iteration, the nearest unvisited node is marked visited and distances are updated accordingly. One way to detect cycles using this algorithm, is to check during each iteration whether any new reachable nodes are the starting node. The fastest known single-source shortest path algorithm for arbitrary directed graphs is an implementation of Dijkstra which runs in $\Theta(|E| + |V| \log |V|)$ and is based on Fibonacci heaps [32]. The downside of Dijkstra is that it is sequential, which may yield long execution times. The radius stepping algorithm [12] is a parallel alternative for the single-source shortest path problem. Where Dijkstra's algorithm considers one node at a time, the radius stepping algorithm visits all nodes within a certain radius simultaneously. At each iteration the radius is incremented based on predetermined rules. The process is repeated until all nodes are visited.

All-Pairs Shortest Paths

The Floyd-Warshall algorithm [30] is one of the most popular algorithms for solving the all pairs variant of the shortest path problem. The algorithm is implemented using three nested for loops and a two-dimensional array representing vertex pairs, which we refer to as the routing table. Values in this array represent the known shortest paths between vertex pairs and are only initialised if there is an edge connecting the vertices. The outer loop iterates over all vertices, and considers each vertex as pivot point. The middle and inner loops iterate over all pairs of vertices, and update values in the array if the path through the pivot point is shorter than the current known shortest path between the two vertices. With slight modifications, such as keeping track of visited nodes, the algorithm can be used to detect cycles.

2.4. Decentralised Networks

Decentralised networks have no single point of control or authority. Nodes are modelled to be autonomous entities with unique identifiers, which compute independently and share no common storage nor computational power. Nodes exchange information through a message-passing framework. To compare performance we make use of complexity measures. These state how various types of resource consumption scale with the size of the input. Bachmann-Landau notation [38] is used to express the asymptotic bounds. Distributed algorithms consume resources in several ways, namely (i) bandwidth or total message count, (ii) the number of bits exchanged, (iii) execution time, and (iv) memory usage. We primarily focus on the message count and time complexity. The time complexity is tied to the bit-size of underlying cryptographic primitives and is explained in section 2.7. We revisit complexity in section 5.2.

To make the exchange we use in our protocol possible a node v needs the ability to respond to a message they receive. To enable this, each directional edge has a corresponding bidirectional communication channel. This is possible because edges in G represent context-based relations such as social infrastructure or financial transactions and have no relation to the communication architecture. Communication methods in the distributed setting are often aimed at reaching the whole network. The two primary examples we discuss are broadcasts and echoes. Broadcasts allow a node to announce information to the rest of the network. Decentralised broadcasting produces a wave of messages propagating through the network until all reachable nodes received the message at least once [31]. As the graph may be cyclic, there is a chance messages are indefinitely propagated. For scenarios with only a single broadcast this can be prevented by not letting nodes transmit if they already received the message. To allow multiple broadcasts in parallel while still guaranteeing termination of the protocol, messages can be given a unique session key such that nodes can discard messages containing a session key they have already seen before. Alternatively, the initiating node may opt to only broadcast to nodes within a certain distance. To achieve this, it picks a value ℓ which is included in the message as time-to-live (TTL) parameter. The TTL is decremented each time a message is forwarded until a node decrements it to zero, at which it halts the propagation. The time-to-live restricts the overhead and may be needed to achieve reasonable performance in large networks. a cycle

Mutual decision making is an essential part of distributed computing and is facilitated by traversal algorithms. Similar to broadcasts, a message is sent to each node in the network. Additionally, each node submits a response to the initiator. The initiator sends a message, or token, to its neighbours. The token proceeds to traverse the whole graph and incites each node to reply to the initiator. The echo algorithm [31] is such an algorithm and forms a stepping stone to a more complex class of distributed algorithms and shares similarities with our approach. After an initiator sends a message to all its neighbours, each other node that receives a message for the first time, marks the sender as its parent. The receiving node then forwards the message to all its other neighbours. After each of its neighbours returned a response, the node sends a message to its parent containing an aggregate of the messages it received from its other neighbours. Finally, the algorithm terminates when the initiator has received messages from all its neighbours.

Decentralised Cycle Detection

The most straightforward way to detect cycles in this setting, without concern for privacy, is to perform a token-based depth-first search. This is nearly identical to broadcasting messages to the complete network. The only required change is that initiators store the session key. The message is propagated in the same fashion as a standard broadcast. Given that there may be concurrent executions of the protocol, the initiating node that receives a message checks whether the stored session key it initiated with matches the session key of the message and concludes it is in a cycle if the comparison is true. The bounded variant of this protocol initiates with ℓ as the TTL in messages and can detect cycles of at most length ℓ.

Decentralised Shortest Path Algorithms

The algorithm proposed by Toueg [59] is a distributed generalisation of Floyd-Warshall. The distributed setting poses two main challenges. First, the order in which pivots are selected must be consistent across all participants. Secondly, pivots have to broadcast their routing table as any subsequent pivots require this information to compute their distances over accumulated paths. The resulting worst-case time complexity, given n nodes and e edges, is $O(n^2)$ and the worst-case message complexity is $O(ne)$. Kanchi et al. proposed an alternative requiring less messages using spanning trees [37]. Their algorithm assumes nodes to have unique identifiers which are unknown to other nodes in the graph. The first step finds a spanning tree of the graph. Secondly, nodes determine the identifiers of their neighbours. The main functionality is performed in the third step. Each node keeps track of a local distance matrix, which at first only contains distances to direct neighbours. Then, leaf nodes send their distance matrix along the tree edges to their parent node. The computation is continued by non-leaf nodes that received information from all but one neighbour. This process repeats until one or two nodes contain the full distance matrix. In the last step the matrix is propagated throughout the network. The time complexity is $O(n)$ and the message complexity is $O(e + n \log n)$, which are both proven to be optimal.

2.5. Security Model

The security model of a system is a specification of potential threats, assumptions, requirements, and the mechanisms put in place to maintain the confidentiality, integrity, and availability of private data. Entities that pose a threat may have an interest in uncovering private information or disrupting the functioning of a system. Such threats are often represented using the adversarial model. The adversary is an entity that attempts to compromise the security and integrity of a protocol or system. The adversary may take a passive role, and merely collect information without disrupting the protocol. This type of adversary is also called semi-honest or honest-but-curious. Malicious or active adversaries may tamper with secret data or interfere with the functioning of a system. Adversaries are classified based on their available computational power. Unbounded adversaries (UB) have an unlimited amount of computational power but do not exist outside the theoretical setting. Typically, we consider probabilistic polynomial-time (PPT) adversaries, for which the probability of the adversary compromising the system in polynomial time must be negligible for the system to be considered secure. There are many more facets to modelling security, see [34] for a comprehensive introduction.

2.6. Multi-Party Computation

Multi-party computation (MPC) is a privacy-enhancing technology in which entities with separate private inputs jointly compute a function while keeping those inputs private. Perhaps the most well-known problem in MPC is the Millionaire's problem [63] introduced by Yao in 1982. This problem considers two millionaires, Alice and Bob, who want to learn which of the two is richer without revealing their actual wealth. This involves a construction which takes as input two numbers and outputs a boolean based on which of the numbers is higher. This was later extended by Goldreich et. al to a setting with an arbitrary amount of parties [33]. One of the building blocks of MPC is Shamir's secret sharing [54], which we call a (t, n)-threshold scheme. The scheme considers as input a secret, which is split up into n shares and distributed evenly among n parties in a way nobody can derive any information about the secret. To reconstruct the secret, at least t players are needed. Some constructions consider a dealer and players, the dealer being responsible for generating shares for each player. Within the scope of this thesis, other MPC building blocks are secure variants of arithmetic operations and comparisons, such as secure multiplication and secure equality testing. We refer to [61] for an in-depth introduction to these concepts.

2.7. Number Theory

Modular arithmetic and group theory are fundamental to cryptography [55]. Hence, we provide a brief introduction to relevant topics within these domains. The defining feature of modular arithmetic is a fixed integer $N \geq 1$ called the modulus. Given integers a, b, the equation $a = b \pmod{N}$ holds if N divides $b - a$. Furthermore, we say the integers a and b are congruent modulo N if the equation holds. Or, in simpler words, computations that produce a value surpassing the modulus wraps around and continues from zero. Within our cryptographic context, the set $\mathbb{Z}/n\mathbb{Z} = (0, \ldots, N-1)$ is called the set of remainders modulo N. Groups are sets which have an associated operation (e.g. addition) and satisfy the following four conditions: closure, associativity, presence of an identity element, and the existence of inverse elements. Groups may be finite or infinite, and are called *abelian* groups if they are commutative. Certain finite abelian groups are cyclic and contain an element g called a generator. This element has the special property that each other element of the group can be obtained by repeatedly applying the group operation to g. Given a multiplicative cyclic group $\mathbb{G}$ with generator g, each element h of $\mathbb{G}$ can be expressed by $h = g^x \bmod p$. The integer x is called the discrete logarithm of h to the base g. Integers in $\mathbb{Z}/n\mathbb{Z}$ that are coprime to N form a group called the multiplicative group of integers modulo N. For a prime p, the group $\mathbb{Z}_p^*$ is cyclic. Moreover, it has $p - 1$ elements, which is referred to as the order of the group. All arithmetic in this thesis is modular, the modulo is omitted for clarity. Now, let us finally address the cryptographic relevance of the definitions that have been introduced so far. First, note that for finding the discrete log no efficient algorithm exists. This makes cyclic groups of large order suitable for cryptography because finding the discrete log is computationally infeasible for a selection of these groups. The group $\mathbb{Z}_p^*$ is not sufficient, because a function referred to as the Legendre symbol can be used to determine whether g^a is odd or even for arbitrary a. The Legendre symbol has different outputs based on whether an integer is a quadratic residue modulo a large prime. See [44] for a formal definition. To avoid this vulnerability, we use a subset of $\mathbb{Z}_p^*$ to construct a multiplicative subgroup $\mathbb{G}$.

Modular Arithmetic Complexity

The hardness of solving problems such as discrete log and integer factorisation relies on the bit-length of the underlying prime numbers. The 2019 cryptographic guideline published by the National Institute of Standards and Technology [10], recommends picking a key size in the range of 128 to 3072 bits, and considers key lengths of under 112 bits to be insecure. Performing computations on numbers of this size makes simple arithmetic costly in terms of bit operations. For $\mathbb{Z}_n$, the bit complexity of integer operations scales with n. Exponentiation is the most expensive operation [44], see table 2.1 for a complete overview.

Table 2.1: Bit complexity of modular operations under modulo n.

Modular Operation	Bit Complexity	Notation
Addition	Logarithmic	$O((\log n))$
Subtraction	Logarithmic	$O((\log n))$
Multiplication	Polylogarithmic	$O((\log n)^2)$
Inversion	Polylogarithmic	$O((\log n)^2)$
Exponentiation	Polylogarithmic	$O((\log n)^3)$

2.8. Cryptographic Primitives

The Diffie-Hellman key exchange [20] is an asymmetric algorithm for communicating an encryption key between two parties over an insecure channel without putting sensitive information at risk. The exchange is shown in Figure 2.1. After the exchange, two parties share a symmetric key which is only known to them. The likelihood of an adversary compromising the keys is dependent on the security level of its underlying primitive, the discrete logarithm problem (DLP). The problem is intractable, or rather, there exists no classical algorithm which can efficiently compute discrete logarithms. To guarantee no real-life adversary can compute the discrete log, the key size (in bits) must be sufficiently large. The decisional Diffie-Hellman (DDH) assumption is a computational hardness assumption related to the discrete log problem [13], and is said to hold for certain cyclic groups [44].

Definition 2.8.1 (Decisional Diffie-Hellman Assumption)**.** Consider the multiplicative cyclic group $\mathbb{G}$ of order q, with generator g. The decisional Diffie-Hellman assumption states that, given uniformly and independently picked $x, y, z \in \mathbb{Z}_q^*$, the following two probability distributions are computationally indistinguishable, (i) the DDH tuple (g^x, g^y, g^{xy}), and (ii) a random tuple (g^x, g^y, g^z).

Note that it is not proven that the assumption holds for the multiplicative group $\mathbb{Z}_p^*$ with prime p of finite field $\mathbb{Z}/p\mathbb{Z}$. Recall that the Legendre symbol can be used to determine whether g^a is odd or even for arbitrary a. Fortunately, there are various subgroups for which the DDH assumption is believed to hold. We opted to work with Schnorr groups [44] because of their simplicity.

Definition 2.8.2 (Group Construction)**.** To generate such a group $\mathbb{G}$, we pick p, q, r such that p, q are large primes and $p = qr + 1$. Next, we select any $h \in \{1, \dots, p\}$ such that $h^r \neq 1 (\text{mod } p)$. The value $g = h^r (\text{mod } p)$ is a generator of the subgroup $\mathbb{G}$ (of order q) and is publicly known.

Prime p should be large enough (say 3072 bits [14]) to resist attacks such as index calculus [53], and prime q should be sufficiently large (say 384 bits [42]) to resist the birthday attack [44] on discrete log problems.

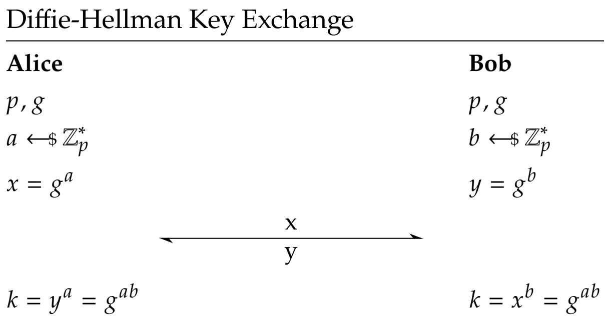
Diffie-Hellman Key Exchange
Alice
Bob
p, g
p, g
$a \hookleftarrow\$ \mathbb{Z}_p^*$
$b \hookleftarrow\$ \mathbb{Z}_p^*$
$x = g^a$
$y = g^b$
x
y
$k = y^a = g^{ab}$
$k = x^b = g^{ab}$

Figure 2.1: Diffie-Hellman key exchange for two parties.

Random Number Arithmetic

Our protocol relies on sampling random numbers for its security. The benefit of working in finite structures is that the uniform distribution is the maximum entropy distribution. For a group $\mathbb{G}$ of order q this means that each element has the same probability q^{-1} of being sampled from the distribution. The order q should be large enough to make the chance of collision negligible. Note that in Schnorr groups each element is a generator, making it harder for an attacker to find weak points. Now, consider a pair of random independent variables $x, y \in_R \mathbb{G}$ and arbitrary $z \in \mathbb{G}$. The amount of (x, y) pairs for which $z = xy$ holds is invariant of z. As z and thus xy have the same uniform distribution, multiplication with a newly sampled independent variable produces another uniformly distributed variable. We abuse this property by repeatedly applying the product operation. This achieves our desired level of security.

3

Related Work

In this chapter we survey related work that meets two or more of the following criteria: (i) subgraph detection is focused on cycles, (ii) the network is distributed and preferably decentralised, and (iii) the setting considers (real) transactional graph data. As financial datasets are scarce, we also briefly touch upon contributions with experimental outcomes based on real data. To the best of our knowledge, the message passing protocol proposed by Porsius [49] is the only contribution that matches all criteria. We discuss the protocol and its vulnerabilities in section 3.1. The rest of this chapter focuses on significant contributions matching two of the three criteria. We provide a full overview of the works we discuss in Table 3.1. In section 3.2 we discuss works on cycle detection and other types of pattern matching. Contributions based on multi-party computation are covered in section 3.3. An adjacent problem to cycle detection is finding shortest paths, we discuss this relation in section 3.4 alongside relevant literature. section 3.5 highlights some state-of-the art contributions that can identify suspicious financial activity in a real-world setting. Lastly, we discuss a contribution in section 3.6 which proposes topology-hiding variants of common algorithms for general graphs.

Table 3.1: Comparison of contributions covered in this chapter. Adversarial capabilities may be probabilistic in polynomial time (PPT), unbounded (UB), or may be assumed secure for arithmetic using the arithmetic black box (ABB) model from [19].

Work	**Pattern**	**Ad.**	**Se.**	**Dy.**	**Re.**	**De.**	**Co.**	**Co. Bound**	**Comp. Bound**	**Technique**
[49]	ℓ-cycles	PPT	○	✗	✗	✓	async	$O(d^\ell)$	$O(d^\ell)$	token passing
[56]	smurf-motifs	-	-	✗	✓	✗	-	-	-	database filter-join
[35]	ℓ-cycles	-	-	✗	✓	✗	-	-	$O(d^\ell)$	depth-first search
[50]	all cycles	-	-	✓	✓	✗	-	-	-	hot-point indexing
[61]	ℓ-cycles	ABB	✓	✗	✗	✓	sync	$O(\log\log n)$	$O(n^\ell(\ell + \log n))$	vertex removal
[62]	even cycles	PPT	✓	✗	✗	✓	sync	$O(n^6)$	$O(n^6)$	hungarian algorithm
[4]	maximum flow	UB	✓	✗	✗	✓	async	$O(n^5)$	$O(n^3)$	push-relabel
[6][5]	shortest-path	ABB	✓	✗	✗	✓	sync	$O(n^2)$	$O(n^2)$	dijkstra
[61]	shortest-path	ABB	✓	✗	✗	✓	sync	-	$O(n \log^2 n)$	dijkstra
[7]	shortest-path	ABB	○	✗	✗	✓	sync	-	$O(\log n + \log m)$	radius-stepping
[45]	communities	-	-	✓	✓	✗	-	-	-	feature extraction
[24]	communities	-	-	✗	✗	✗	-	-	-	spectral analysis
[23]	egonets	-	-	✗	✓	✗	-	-	-	feature extraction
[2]	any	PPT	✓	✗	✗	✓	sync	$O(\kappa n^5)$	$O(\kappa n^3)$	random walks

adversary (ad.), security (se.), dynamic (dy.), real data (re.), decentralised (de.), communication (co.), computational (comp.)

3.1. Private Cycle Detection in Financial Transactions

Existing research on privacy-preserving cycle detection in the decentralised setting is limited. The work by Porsius [49] is the closest to our solution, and to the best of our knowledge no other privacy-preserving solutions have been proposed for the decentralised setting. Their proposed protocol is based on the graph model and is tailored towards financial data. Accounts are represented by nodes, transactions by edges, and a set of banks by a partitioning of the graph. Banks are semi-honest, they try and learn as much information as possible without deviating from the protocol. The protocol is deemed secure iff there is no bank which can learn about the existence of any nodes and edges it does not already know, after the protocol has been executed an arbitrary amount of times.

Vulnerabilities

The algorithm proposed by Porsius [49] makes use of a one-time pad each time a message is forwarded to hide the existence of transactions to entities not involved in the transaction. Their protocol relies on a message following the same path twice, such that the one-time pad is applied twice and cancels out. Re-using the one-time pad in such a way opens up the possibility for an attack.

Let Alice, Bob and Eve be three nodes within the network which form the cycle $\sigma = (A, E, B, A)$. Eve is a honest-but-curious adversary. After the protocol has finished executing (assume $\ell = 3$ for simplicity), Eve has sent or received the messages $((6, R), (5, RS_E), (3, RS_ES_BS_A), (2, RS_BS_A))$. Eve can deduce that the messages $(6, R)$ and $(3, RS_ES_BS_A)$ are part of the same instance by computing $RS_ES_BS_A \oplus RS_BS_A \oplus RS_E = R$. As the difference in ℓ for the messages is exactly 3, Eve learns that there exists an edge between Alice and Bob, which may not be desirable.

Similar attacks can be devised for more complex topologies and larger values of ℓ. In particular, when messages are propagated to Eve through a cycle which the initiating node is not apart of, they may gain knowledge about edges that are not in a cycle. This becomes especially problematic in settings where nodes collude or are owned by the same adversary. Additionally, the more instances are ran which Eve is a part of, the more they can potentially learn about the network by observing throughput and estimating their centrality.

3.2. Pattern Matching

Compared to graph analytics for fraud detection, traditional rule-based methods fail to encompass features of the full transaction graph. As such, finding structural patterns in large transactional graphs has gained traction as a complement to existing monitoring systems. The significance of this trend is demonstrated by the massive study performed by Starnini et al. [56] on a real dataset containing 180 million transactions and 31 million bank accounts. Using a filter-join approach they were able to identify four new motifs rule-based methods failed to identify. The problem of finding specific structures in graphs is also referred to as subgraph matching. Solutions to this problem generally enumerate all subgraphs that match the specified pattern. Cycle detection can be interpreted as a type of subgraph matching. For transactional graphs, there are only few of publications that have results based on non-synthetic data.

The proposed method from [35] uses a DFS-based approach to find cycles in a static dataset. The crux of DFS-based approaches is that the search space explodes when a vertex with high outdegree is encountered. To keep the runtime reasonable the authors introduce a parameter α, which limits the maximum difference in weight (i.e. transaction amount) between the edges that make up the cycle. The algorithm takes a few hours to run and is able to detect 2 to 4 new fraudulent transactional patterns a month. The approach described in [50] can detect cycles in dynamic graphs, and is deployed on the e-commerce platform Alibaba. Their system identifies vertices with high outdegree, referred to as hot-points, and keeps an index of length constraints between each pair of hot-points such that these do not need to be traversed during the search. The index is kept relatively small such that it can be updated quickly in a dynamic setting. While these approaches are not decentralised nor privacy-preserving, their relevance stems from their results. Both contributions show that nearly all cycles are of length 6 or lower. This has significant implications for the complexity of our proposed solution.

3.3. Graph Algorithms with MPC

In this section we discuss MPC-based contributions. We first discuss a cycle detection algorithm that uses secret sharing to hide the adjacency matrix representing the graph. The second contribution is tailored to goods bartering, and abstracts a problem within this domain to cycle detection in bipartite graphs. The last work addresses maximum flow problems, and is unique in the sense that parties may control one or multiple nodes. This is of interest as banks within the financial domain control multiple nodes (accounts) as well.

Cycle Detection through Vertex Removal

The algorithm proposed by Vorstermans [61] uses the incoming edges $N-(v)$ to find cycles. On each iteration, all vertices that have no incoming edges are removed. This process is continued until no vertex is removed in the current iteration. The remaining structure contains only cycles, and is empty if no cycles exist. The MPC implementation makes use of a secret-shared adjacency matrix and uses an auxiliary list of decision bits to indicate which bits have been removed. The output is a boolean denoting whether the graph contains a cycle. For enumerating cycles of a fixed size α they propose a round-based algorithm which iterates over all possible paths of length α. To determine the product of edge existences a log-depth multiplication tree is employed. Paths that form a cycle are stored. While this keeps the round complexity reasonable, the computational complexity is exponential for α. This approach requires a set of nodes to collectively have knowledge about the relevant part of the adjacency matrix for a particular node. On top of that, nodes are expected to be fully connected in their communication graph.

Cycle Detection for Bipartite Graphs

Wüller et al. proposed a method for finding trade cycles between parties bartering for goods [62]. They construct a weighted bipartite graph in such a way that finding the maximum weight matching encodes an optimal bartering opportunity. To find such matchings in a conventional setting they adapt a function called the Hungarian algorithm. The proposed privacy-preserving protocol consists of three steps. First, parties compute an encrypted bi-adjacency matrix such that no party learns anything about the inputs of other parties. The second step utilises a modified variant of a function called the Hungarian protocol to compute encrypted maximum (perfect) matchings. In the third and last step the optimal bartering opportunities for each party are extracted in a cooperative manner. This is achieved in a way parties are able to learn only their own trade partners. The (τ, ι)-threshold Paillier cryptosystem is used to achieve security. This variant of the Paillier cryptosystem splits and distributes the private key among ι parties. Ciphertexts can only be decrypted if τ parties cooperate in the decryption. The protocol relies on the additive homomorphic property of Paillier. This property states that given a public key and two ciphertexts, one can compute the sum of the plaintexts by taking the product of their ciphertexts. Matchings can be used to find even-length cycles in bipartite graphs by searching for self-intersecting paths that alternate between matched and unmatched edges. The main limitation with respect to our objective is that an upper bound on the maximum trade cycle length cannot be specified.

Network Flow Problems

The maximum flow problem takes as input a directed weighted graph, a source node, and a sink node. Weights represent capacity, and each edge can receive a non-negative flow { that is at most equal to its capacity. The objective is to maximise the flow of each edge, such that the flow between the source and sink is maximised. Besides the capacity, the incoming and outgoing flow of each node must be equal. The maximum flow is equal to both the total flow of outgoing edges of the source and the total flow of incoming edges of the sink. The approach by Aly in [4] describes a privacy-preserving push-relabel algorithm, which considers a group of players that each only know a part of the network. Their goal is to compute the maximum flow while keeping their inputs private. The intermediate steps of the algorithm push flow through edges that still have residual capacity, even if that violates the in-out flow balance of a node. Nodes for which $\{_{in} > \{_{out}$ are said to be active. Generally speaking, the algorithm consists of the following steps:

1. Identify active nodes that are not source nor sink.
2. Select some active node v.

3. Push and relabel flow in node v.
4. If any active nodes remain, repeat from step 2.

The privacy-preserving adaptation has some changes that do not affect the logic of the protocol. The first difference is that the operations in the push-relabel step are made identical for active and non-active nodes. Secondly, iteration is performed over all vertices or all edges, for which the order has been agreed upon in advance. The time complexity of the algorithm is $O(n^4)$ for general graphs. Besides maximum flow, Aly describes various other graph-related algorithms with equal or worse complexity in their dissertation [4].

3.4. Secure Shortest Path Algorithms

There is a line of research on solving the shortest path problem in MPC settings. Securely finding shortest paths may yield information about cycles in the graph depending on the implementation. The naive approach to use MPC for this purpose is to encrypt or secret share the input before passing it to conventional algorithms. However, this approach still leaks information. This flaw arises from the fact that conventional algorithms branch conditionally and thus leak structural information in their outputs. Aly et al. proposed a handful of secure implementations of popular shortest path algorithms [6]. The lowest worst-case complexity was achieved on their optimised secure implementation of Dijkstra's algorithm. In [6] they made two significant changes to the basic Dijkstra algorithm: (i) iteration is performed over all vertices rather than the neighbours of the current vertex when considering the next vertex, and (ii) to update a vector in a shared list all elements have to be iterated over even though only one value is updated. This changes the computational complexity from quadratic to cubic. They optimised the initial version of their secure Dijkstra [5] to a run-time of $O(n^2)$ by applying an oblivious permutation to the input matrix. This removes the link between rows and vertices during the execution of the protocol and lets the algorithm directly explore the most suitable vertex rather than having to iterate over the entire set of vertices.

Another approach was described in [61] and uses an oblivious random access memory (ORAM), this is a paradigm frequently used to obfuscate access patterns in the client-server setting such that no adversary can derive information from observing access behaviours. To make it work for the shortest path algorithm, ORAM is first adapted for the MPC setting and then used to construct oblivious data structures. To understand why this may be of use for implementing Dijkstra's algorithm, consider that the algorithm relies on selecting the lowest weight path on each iteration, updating it, and picking the new lowest weight path on the next iteration. Fast implementations of Dijkstra usually rely on an efficient data structure called a priority queue for keeping the list of paths dynamically sorted. Min-priority queues provide this exact functionality as they are dynamically sorted, such that the lowest element is at the front of the queue. The shortest path algorithm suggested in [61] is Dijkstra-inspired and makes use of an ORAM-based oblivious queue. The resulting complexity depends on both n and the number of edges m. The ORAM implementation outperforms state-of-the-art non-ORAM alternatives in sparse graphs with a complexity of $O(n\ (\log n)^2)$, but remains slower for general and dense graphs.

Secure Radius Stepping

The radius stepping algorithm has also been studied in the MPC setting. The secure adaptation in [7] uses the same basic structure as the standard algorithm, but encodes all variables as secret-shared vectors to acquire its privacy guarantees. The time complexity depends on the radius r. For $r = 0$ the complexity is $O(n + \log m)$ and for $r = \delta$, where δ is the current mapping, the complexity is $O(\log n + \log m)$. The algorithm stands out as it is resistant against semi-honest and malicious adversaries. There is a small amount of information leakage, though. The number of iterations the algorithm performs conveys information about the underlying graph structure and used radii.

3.5. Anomaly Detection

Both researchers and financial organisations have created tools for analysing transactional financial data in the form of a graph such that suspicious activities can be singled out. These kind of activities are also called anomalous, as they present itself as patterns that differ from normal transactional data. Anomaly detection is primarily researched in the financial setting. The discussed contributions are thorough or novel in how they use different properties to find anomalies. Two of the works were also given access to real financial data for their experiments. There has been a lack of research in this domain for the distributed and privacy-preserving setting. We discuss one method for detecting of anomalies that makes use of *egonets*, and two methods based on local community analysis.

Local Community Properties

Molloy et al. proposed a fraud detection method [45] which assumes that account holders perform trusted transactions within their local neighbourhood, and considers other transactions to be suspicious. They explore various ways to construct communities using purely graph properties, with the purpose of reducing false positives in existing systems. The analysis considers on transaction at a time and constructs a graph containing the transactions itself and the ones preceding it. The identified properties are the shortest distance between source and target, their page rank, and whether they share the same strongly connected component. Additionally, accounts are clustered through various means and an estimation is made on the likelihood of a transaction occurring between the source and target account. The evaluation is performed on a real-life dataset and shows a reduction upwards of 30% of false positives compared to existing methods.

Feature Extraction

The pipeline proposed in [24] combines various anomaly detection approaches which are based on spectral localisation, community properties and node connectivity. Spectral localisation is a mathematical abstraction of centrality which appertains to the idea that nodes are central in a network when they are connected to other central nodes. This concept is mathematically encoded in the eigenvectors of the adjacency matrix representing the graph. Returning to the context of fraud detection, high eigenvalues that are concentrated on a small subset of nodes are flagged as anomalous. Additionally, the pipeline integrates a network comparison methodology called *NetEMD*, which makes use of small subgraph degree distributions to assess similarity between (parts of) networks. The authors discuss two ways of combining outputs of all components. The first aggregated output is an unweighted sum of the features, and the second is based on a learning model and feature extraction.

Reduced Egonets

Dumitrescu et al. [23] proposed a set of features based on reduced egonets. For some node v, its egonet $ego(v)$ of radius 1 is defined as node v and all its direct neighbours. The reduced egonet $ego_{red}(v)$ is the egonet $E(v)$ from which all nodes that are only connected with v have been removed. The authors show that there are fraud patterns in which the features of the respective egonets and its reduced variant diverge from normal nodes. These are the in- and outdegrees, total and average transaction amounts sent and received, and the ratio between the number of nodes and edges. The implementation complements existing anomaly detection techniques, and the experimental results show competitive results on both synthetic and real data. The approach only makes use of the direct neighbourhood of a node, though, and the authors note that from the perspective of a single bank, extending the radius of egonets produces a more distorted view since the connections between clients of other banks are not available.

3.6. Topology Hiding Communication

Distributed protocols with partial communication graphs are said to be topology hiding if no information is revealed about the graph topology beyond what the output may reveal. The protocol in [2] shares many similarities with our protocol, but assumes a public-key infrastructure to be in place. Their main contributions are a proof that topology hiding computation is feasible for *every* network topology under the DDH assumption and a two-phase protocol based on random walks. The first phase propagates a message forward using a random walk. Homomorphic encryption is used to aggregate a secret bit for each node a message passes through. Upon forwarding the message, nodes add an encryption layer. The second phase traces back the walk, such that each node removes the layer of encryption added in the first phase. To obtain the complexities for general graphs, consider the input n, κ, where the input size is polynomial in the security parameter κ. The round and communication complexity are $O(\kappa n^3)$ and $O(\kappa n^5)$ respectively. This makes the random walk approach infeasible for graphs with large n.

4 Decentralised Cycle Detection

We propose a privacy-preserving protocol for detecting cycles in decentralised networks. Each node in the network owns a copy of the protocol, which consists of four algorithms that the node can execute, as well as three types of messages. The first algorithm is initiate, which may be executed at any time by any node. The propagate, echo, and trace routines are executed when a node receives a corresponding forward, echo or trace message. The basic idea is that messages are first propagated to each node within range ℓ. Each node receiving a forward message responds with an echo, and propagates forward messages to its other neighbours. After the initiating node becomes aware of a cycle, it may call the trace to determine the path that makes up the cycle. Once the nodes that make up the cycle have been found, the initiating node broadcasts the cycle and it becomes public information. In the remainder of the chapter we first by introduce our building blocks, container structures and parameters. Next, we present the algorithms as pseudocode and walk through them step-by-step. Lastly, we provide a visualisation of an instance of the protocol.

4.1. Protocol Architecture

Consider a directed graph $G = (V, E)$, and a Schnorr group $\mathbb{G}$ as subgroup of $\mathbb{Z}_p^*$. The construction of $\mathbb{G}$ is discussed in section 2.8. The protocol is based on message exchange between nodes. Messages consist of a set of values depending on the type, and have one source and one target. Each instance of the protocol has exactly one initiating node $v_i \in V$. The goal for a particular instance is to find all cycles of at most length ℓ containing v_i. As we do not allow self-loops, the parameter ℓ should be at least 2. To initiate the protocol, v_i sends out a forward message containing both a partial key and the parameter $\ell - 1$ to each of its neighbours. The receiving node propagates the forward message with $\ell - 1$ to each of its outgoing neighbours. Nodes do not propagate a message when they receive a message with $\ell = 0$, or when they have no available neighbours to forward the message to. For each forward message a node receives, it computes the full shared key and sends back an echo with their partial component of the key. Echoes are relayed back following the same path (but in reverse) back to the initiator. When the initiator receives an echo back, it computes the full shared key using the partial key from the echo and the secret value it initiated with. When there is a cycle, the node will have received a propagated message from itself, to which it responded with an echo. In other words, it (anonymously) performed key exchange with itself. Cycles can be detected by the initiator by checking whether a key has already been established upon receiving an echo. For an instance where the graph contains only a path $\pi = v_i, v_j, v_n$, the protocol terminates after the key exchange because no cycles were found. When v_n is replaced by v_i the key exchange is performed in the same way. Nodes receiving a forward message do not know the identity of the initiating node. For echoes received by the initiator, the unknown node in the key agreement is revealed to be the node itself only iff a cycle exists, and remains anonymous otherwise. When the initiator learns a cycle exists, it will start a trace by sending a message with a path containing only the initiator to the node it received the last partial key from. This node receives the message and adds its identifier to the path, after which it sends the message to the next node in the cycle. This continues until the trace, now containing σ, reaches the initiator again.

Before delving into the specifics of each algorithm, the following needs to be taken into account. Each computation is performed under modulo p, unless stated otherwise. For simplicity we assume $\mathbb{G}$ to be agreed upon before execution and known to all nodes in the network by the tuple $\mathbb{G} = (p, q, r, h, g)$. The nonce and key sizes are controlled by the security parameters κ_p, κ_q and κ_r. These correspond to the bit-lengths of the primes p, q in $\mathbb{G}$, and the bit-length of an arbitrary nonce r. The bit-length of private keys is κ_q, whereas public and shared keys have a bit-length of κ_p. There is a bidirectional communication channel between v_i and v_j for each $(v_i, v_j) \in E$. The channels are reliable and communication between nodes is asynchronous. Nodes have access to the function send, which takes a receiver as input as well as a tuple containing the message content.

4.2. Initiating Instances

Any node $v_i \in V$ may initiate an instance by executing initiate at any given time. The pseudocode for this routine is given in Algorithm 1. It takes ℓ as parameter and consists of a single loop, which iterates over the outgoing neighbours of v_i and sends each a forward message. For each $v_j \in N^+(v_i)$, a private key x_j and nonce r_j are uniquely picked. The private key is used to compute the (partial) public key g^{x_j}. The forward message is given by the tuple $(r_j, g^{x_j}, \ell - 1)$, which v_i sends to v_j. As multiple instances may be running at the same time, v_i uses a datastructure called `pending` to keep track of key-nonce pairs such that it can recognise responses to its own instance.

Algorithm 1 v_i initiating the protocol
`initiate(ℓ)`

```
1: for $v_j \in N^+(v_i)$ do
2:     $x_j \in_R \mathbb{Z}_q^*$
3:     $r_j \in_R \{0, 1\}_{\kappa_r}$
4:     append $(x_j, r_j)$ to pending
5:     send $m = (r_j, g_j^x, \ell - 1)$ to $v_j$                ▷ forward message
6: end for
```

4.3. Propagating Forward Messages

Upon receiving a forward message, nodes execute the propagate routine, which we specify in Algorithm 2. The resulting set of messages contains at least one echo message, and may contain forward messages to each outgoing neighbour. Given is a node v_j which receives a forward message $m = (r_{i,j}, g^x, \ell)$ from v_i. The routine takes the message m as its only parameter. As in initiate, a secret value y is picked from $\mathbb{Z}_{\shortparallel}^*$ which v_j uses to compute g^{xy}, and the public key g^y. The value g^{xy} is the secret key node v_j shares with the initiator. The value g^y is the partial key which will be echoed back to the initiating node, which can use this value to compute the same shared key. The datastructure `key` is used to keep track of shared secret keys with other nodes. Node v_j constructs the echo message $m' = (r_{i,j}, g^y)$ and sends it to v_i. For a graph containing the nodes v_i and v_j and only a single edge (v_i, v_j), the key generation is identical to the Diffie-Hellman key exchange [13].

Node v_j continues by checking if $\ell == 0$ holds in received message m, and returns if this is true. If not, node v_j still has to propagate forward messages to its neighbours. The node could naively construct a message containing $r_{i,j}$ and g^x, but this makes it obvious that the messages belong to the same instance. Instead, node v_j picks two random values for each node in $N^+(v_j)$. These are a new unique independent nonce $r_{j,n} \in_R \{0, 1\}_{\kappa_r}$, and an intermediate secret key $k_{j,n} \in_R \mathbb{Z}_{\shortparallel}^*$. As v_j should still be able to pass echo messages back to the correct node, it utilises a datastructure `routes` to store a tuple for each $v_n \in N^+(v_j)$. This tuple contains v_i and its associated nonce, as well as v_n, the new nonce, and the intermediate key. The node v_j proceeds by constructing the message $(r_{j,n}, g^{xk_{j,n}}, \ell - 1)$ and sending it to v_n. Note that the mapping between values in the received message and the sent message is only known to v_j. We further explain this in section 5.1, where we show that an adversary cannot estimate the relatedness of these messages with significant accuracy.

Algorithm 2 v_j receives forwarded message from v_i
`propagate`$(m = (r_{i,j}, g^x, \ell))$

```
$y \in_R \mathbb{Z}^*_q$
append $g^{xy}$ to keys
send $m' = (r_{i,j}, g^y)$ to $v_i$                          ▷ echo message
if $\ell == 0$ return
for $v_n \in N^+(v_j)$ do
    $r_{j,n} \in_R \{0,1\}_{\kappa_r}$
    $k_{j,n} \in_R \mathbb{Z}^*_{||}$
    append $(v_i, r_{i,j}, v_n, r_{j,n}, k_{j,n})$ to routes
    send $m'' = (r_{j,n}, g^{xk_{j,n}}, \ell - 1)$ to $v_n$   ▷ forward message
end for
```

4.4. Echoing Responses

The echo pseudocode is specified in Algorithm 3, and is invoked when a node v_i receives an echo message. First, v_i checks `pending` to see whether it is the initiator of the instance the received message belongs to. Node v_i knows it is the initiator if `pending` contains a pair with a nonce matching $r_{i,j}$ in the received message. When v_i is the initiator, it retrieves the associated pair $x, r_{i,j}$ from `pending` and computes g^{xy} as the shared key. Node v_i proceeds by searching `keys` for this key. When the key is not present, v_i appends g^{xy} to `keys`. When the key is already present in `keys`, v_i knows it performed key agreement with itself, from which it concludes that a cycle exists. To determine which edges the cycle is composed of, v_i constructs a trace message $(r_{i,j}, g^y, [v_i])$ containing the same nonce and partial key as in the received message, together with an array representing the vertices currently known to be in the cycle. The only value in this array is v_i as this is the first trace message. The message is sent to node v_j, the same node v_i received the echo from.

When there is no pair in `pending` which has a nonce matching $r_{i,j}$, the echo has not yet reached the initiating node. Node v_i searches `routes` for records containing $r_{i,j}$. This returns the tuple $(v_n, r_n, v_j, r_{i,j}, k_{i,j})$, from which the node learns that v_n should receive the next echo message with nonce r_n. We use an additional structure `retrace` which is accessed upon tracing back cycles in Algorithm 4. Lastly, v_i constructs the echo message. During the propagation phase, v_i added $k_{i,j}$ to the exponent of the public key. This operation is commutative; if each node adds the same exponent in the echoing phase as it did in the propagation phase, irrespective of ordering, the resulting shared key will be identical upon reaching the initiator. Node v_i proceeds by sending $(r_n, g^{yk_{i,j}})$ to v_n.

Algorithm 3 v_i receives an echoed message from v_j
`echo`$(m = (r_{i,j}, g^y, \ell))$

```
if pending contains $(\_, r_{i,j})$ then
    fetch $(x, r_{i,j})$ from pending
    if keys contains $g^{xy}$ then
        send $m' = (r_{i,j}, g^y, [v_i])$ to $v_j$            ▷ trace message
    else
        append $g^{xy}$ to keys
    end if
else
    fetch $(v_n, r_n, v_j, r_{i,j}, k_{i,j})$ from routes
    append $(r_n, v_j, r_{i,j}, k_{i,j}, g^y)$ to retrace
    send $m'' = (r_n, g^{yk_{i,j}})$ to $v_n$                  ▷ echo message
end if
```

4.5. Tracing Cycles

The trace function in Algorithm 4 is fairly straightforward. The trace is started in echo with a message containing the nonce $r_{i,j}$, the public key g^x, and the currently known path that makes up the cycle.

The initiating node v_i includes only itself in this path, after which it sends the message to v_j. Upon receiving the trace message, v_j knows it is part of a cycle that has not been fully reconstructed yet. During the earlier phases of the protocol v_j stored routing information, from which it can derive which node is next in the cycle. The trace routine continues by iterating over the tuples in `retrace` that contain $r_{i,j}$ as its first element. The nonce occurs in multiple tuples because the forward message v_j initially received from v_i was propagated to all its neighbours. The resulting echoes are all sent back over the same edge with a nonce identical to the nonce in the initial forward message. For each matching tuple $(r_{i,j}, v_n, r_{j,n}, k_{j,n}, g^y$, the value $g^{yk_{j,n}}$ is computed. Because v_j generated a new intermediate secret key for each neighbour, there is exactly one neighbour v_n for which its tuple contains the same public key g^x as the received message.

Node v_j appends its identifier to the path, and sends a message containing the updated path, the nonce $r_{j,n}$ and the public key g^y from the tuple to v_n. This process is repeated until the initiator eventually receives a trace message which contains a path for which $\pi(1) = v_i$. The initiator appends v_i to the path and completes the cycle, after which it broadcasts the newly found cycle to the rest of the network. We do not provide an implementation for the broadcast function. Anonymous broadcasting has been widely studied in literature and is beyond the scope of our protocol. For an overview of anonymous communication techniques we refer to the survey performed by [51], which includes protocols that provide both sender and receiver anonymity.

Algorithm 4 v_j receives trace message from v_i
`trace(`$m = (r_{i,j}, g^x,$ `path[]`$)$`)`

```
if $v_j$ == path[0] then
    broadcast path++[$v_j$]
else
    for $(r_i, v_n, r_{j,n}, k_{j,n}, g^y) \in$ retrace do
        if $g^x == g^{yk_{j,n}}$ and $r_{i,j} == r_i$ then
            send $m'' = (r_{j,n}, g^y,$ path++$[v_j])$ to $v_n$        ▷ trace message
        end if
    end for
end if
```

Putting it all Together

We showcase an instance with messages indicated only by type in Figure 4.1. The instance is started by v_i and uses the parameters $n = 7$ and $\ell = 3$. The topology has some dead ends and contains three cycles each including v_i, where $3, 4, 6$ are their respective lengths. For parallel execution of multiple instances one can imagine a similar construction with more messages being exchanged at the same time.

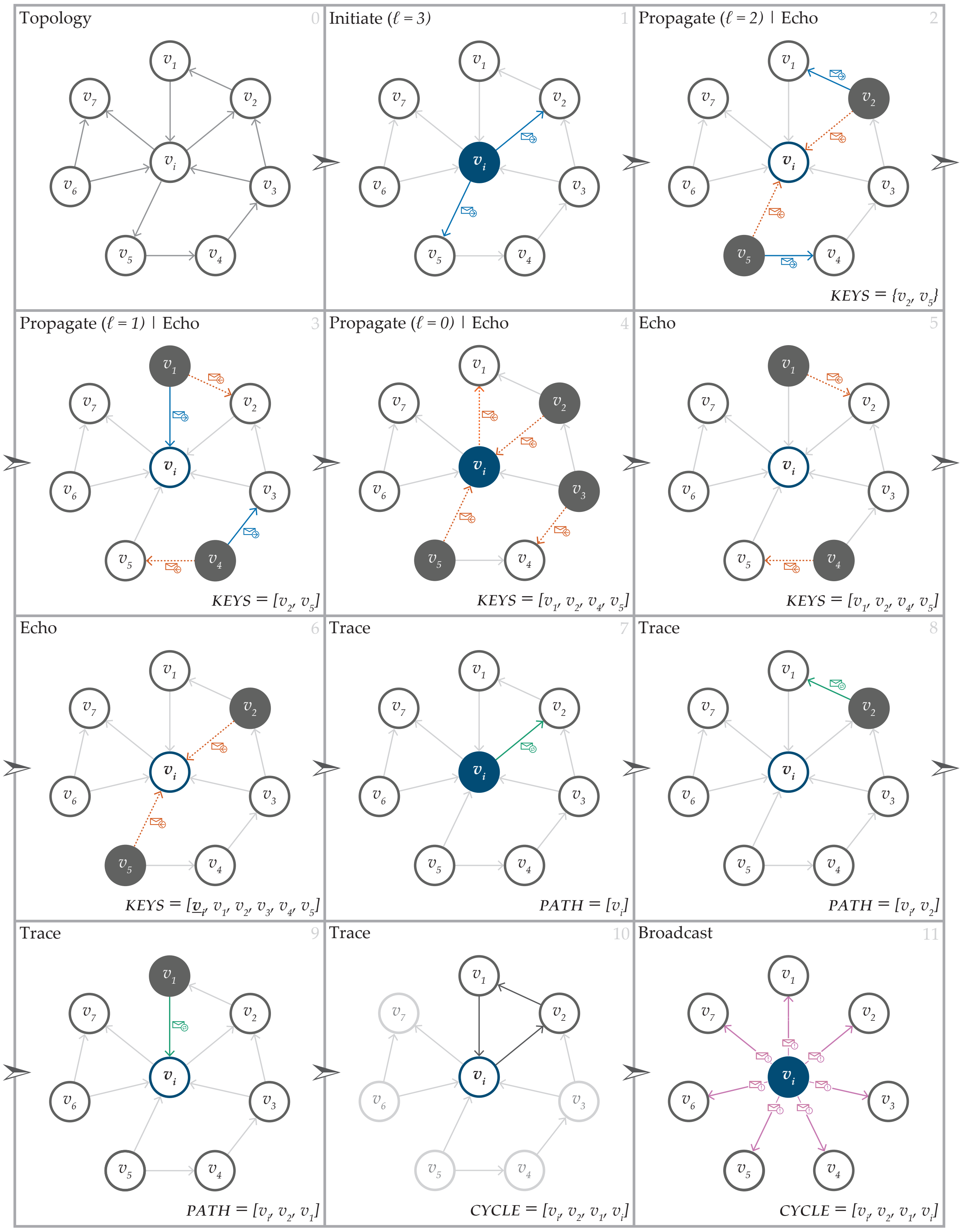
Topology
0
Initiate ($\ell = 3$)
1
Propagate ($\ell = 2$) | Echo
2
$KEYS = \{v_2, v_5\}$
Propagate ($\ell = 1$) | Echo
3
$KEYS = [v_2, v_5]$
Propagate ($\ell = 0$) | Echo
4
$KEYS = [v_1, v_2, v_4, v_5]$
Echo
5
$KEYS = [v_1, v_2, v_4, v_5]$
Echo
6
$KEYS = [\underline{v}_i, v_1, v_2, v_3, v_4, v_5]$
Trace
7
$PATH = [v_i]$
Trace
8
$PATH = [v_i, v_2]$
Trace
9
$PATH = [v_i, v_2, v_1]$
Trace
10
$CYCLE = [v_i, v_2, v_1, v_i]$
Broadcast
11
$CYCLE = [v_i, v_2, v_1, v_i]$
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_i

Figure 4.1: Visual representation of the flow of messages produced by the protocol for a small graph. The opaque nodes in each step are the nodes which send messages. Dotted lines indicate communication in the opposite direction (echoes) of the underlying edge. We discuss the protocol step-by-step, in practice however, the communication is asynchronous. Node v_i is the initiator in step 1. Propagation happens in step 0 – 3, and stops in 4 when v_3 receives a message with $\ell = 0$. All responses are echoed back to v_i in steps 2 – 6. In step 6, v_i receives a message containing a key it already knows. The resulting trace which finds the cycle happens in steps 7 – 10. Lastly, the newly found cycle can be broadcast across a another (anonymous) network.

5 Analysis

We begin this chapter with a security argument and show our protocol is secure under the given constraints. Next, we discuss the performance of the protocol in terms of time, communication and storage. For each algorithm, we identify the factors that influence its performance and state the theoretical asymptotic bounds for each of the metrics. We support this by implementing the protocol, and benchmarking its performance for varying graph properties.

5.1. Security Analysis

We introduce our notion of security and discuss the capabilities of the adversary $\mathcal{A}$. Given a set of constraints, we show that the protocol has provable security iff $\mathcal{A}$ is forced to solve the underlying cryptographic primitive to break the security. The capabilities cover communication channels, set-up, computational ability, and various restrictions on adversarial behaviour. As part of the set-up we assume group $\mathbb{G}$ to be publicly known. We use the general framework for security models in [34] to define the following constraints for $\mathcal{A}$,

- Communication channels are asynchronous and can be tapped.
- Network traffic is reliable and cannot be modified.
- Computational power is bounded such that DLP is intractable.
- Adversarial nodes are passive and execute the protocol honestly.
- Collusion is possible between any adversarial nodes.
- Honest nodes have hidden computations and memory.
- Broadcasting information can be done anonymously.
- Internal processes and memory of honest nodes are hidden.

Our security definition does not include attacks based on side-channels nor network traffic analysis.Some of these threats may be mitigated by using anonymous networks and parallel execution. When there are many instances running simultaneously timing attacks (e.g. by observing ℓ) become infeasible due to the high volume of messages being sent over the network. Now that the constraints are clear, we can construct our security definition. The goal of $\mathcal{A}$ consists of two objectives, namely (i) to determine whether any two messages belong to the same instance and (ii) guess the existence of an arbitrary edge with non-negligible accuracy. These two objectives pertain to linkability and edge existence respectively. This leads us to the following definitions.

Definition 5.1.1 (Linkability)**.** Any two messages (m_1, m_2) are linked iff they belong to the same instance.

From the perspective of the adversary, messages are unlinkable if there is no way to gain a significant advantage over randomly guessing whether the messages belong to the same instance. If $\mathcal{A}$ does have an advantage for a particular set of messages, we say the messages are linkable. We denote the set of nodes $\mathcal{A}$ controls by $V_A \subseteq V$. Consider the set M as the set of all exchanged messages across an arbitrary amount of instances. The objective is fulfilled if no pair of messages m_1, m_2 is linkable and non-implied. Given an adversarial node v_A, links between messages are implied in the following cases, (i) the set containing the input message to a function and the corresponding outgoing messages, (ii) any forwarded message with its corresponding echoes. We can simplify our definition if we only consider messages sent between honest nodes and adversarial nodes, as that is the only type of message which increases the knowledge of $\mathcal{A}$.

The set of messages available to $\mathcal{A}$ is given as $M_{\mathcal{A}} \subseteq M$ and contains all messages that were sent and received by nodes in V_A. Some messages have the same nonce. The re-use happens when nodes store routing information such that echoes can be returned to the correct sender after they forward a message. This is not a security flaw but rather a property inherent to the design of the protocol. Linkability is a requirement for these messages, as otherwise a node receiving an echo is not able to determine which neighbour it should propagate the echo to. In this scenario the messages have a *known* link. Links can also be known if $\mathcal{A}$ controls multiple nodes. For the unknown links we construct the definition as follows.

Definition 5.1.2 (Unlinkability)**.** Messages in M are unlinkable if for any pair of messages $(m_1, m_2) \in M \times M$ where $m_1 \neq m_2$, the existence of an unknown link cannot be guessed by $\mathcal{A}$ with significant advantage.

The second objective follows from our definition of unlinkability. Assuming the first objective holds and none of the messages are linkable, $\mathcal{A}$ cannot derive any information about edges that do not involve at least one node in V_A.

Definition 5.1.3 (Edge Existence)**.** Consider a game in which an adversary $\mathcal{A}$ is given all messages a node v_A received across an arbitrary amount of instances and is asked whether an unknown edge e exists within the topology. $\mathcal{A}$ wins if it guesses correctly, and is able to imply edge existence if it wins the game with non-negligible advantage.

Combining the edge existence property with link guessing, we obtain our security definition. Note that the definition requires messages from at least two instances, as otherwise the link between messages is trivial.

Definition 5.1.4 (Security)**.** The protocol is secure iff an adversary $\mathcal{A}$, given the set M of all messages $\mathcal{A}$ received across an arbitrary number of instances, the following holds:

1. Existence of an unknown link between $m_1, m_2 \in M$ cannot be guessed by $\mathcal{A}$ with significant advantage.
2. Edges unknown to $\mathcal{A}$ cannot be inferred by $\mathcal{A}$ with non-negligible accuracy.

Linking Messages

We start by proving that the first objective holds when the adversary is given M instead of $M_{\mathcal{A}}$. Naturally, if no messages are linkable in M then the same holds for $M_{\mathcal{A}}$. The adversary has access to the pair $(g^{x_{i,j}}, r_{i,j})$ for each message in M. When only the nonce $r_{i,j}$ is involved, unlinkability is implied as the nonce is picked at random for each new forwarded message. The argument for the remaining value, $g^{x_{i,j}}$, is based on similar reasoning. The main difference is that $x_{i,j}$ is a product of factors and is not always random. Factors of $x_{i,j}$ are picked randomly, but contrary to r_i, are repeated across messages. This leads us to the statement that we are trying to prove.

Lemma 5.1.1. *Consider an adversary $\mathcal{A}$ with access to the messages $M = (m_1, m_2, m_3)$, which respectively contain the elements (g^x, g^y, g^z). Given are independently picked $x, y, z \in_R \mathbb{Z}_p^*$ and $z' = g^{\alpha}$ such that α is a linear combination of x and y. The messages in M are unlinkable iff $\mathcal{A}$ cannot differentiate between the probability distributions (g^x, g^y, g^z) and $(g^x, g^y, g^{z'})$.*

Proof. If it were possible to efficiently compute discrete logarithms in $\mathbb{G}$, one could efficiently compute the discrete log of each element in the tuple (g^x, g^y, z) and derive whether $\log_g z$ is a linear combination of x and y. However, as we covered in section 2.8, computing discrete logs is hard in $\mathbb{G}$, thus leading to a contradiction. □

Lemma 5.1.2. *Given is an adversary $\mathcal{A}$ with access to n messages $M = (m_1, m_2 \ldots, m_n)$ and their corresponding elements $X = (g^{x_1}, g^{x_1 x_2}, \ldots, g^{x_1 x_2 .. x_n})$. The messages in M are unlinkable iff, given independently picked values $\{z_1, z_2, \ldots, z_n\} \in_R \mathbb{Z}_p^*$ such that $Z = (g^{z_1}, g^{z_2}, \ldots, g^{z_n})$, the adversary $\mathcal{A}$ cannot differentiate between the probability distributions X and Z.*

Proof. Solving the discrete log for elements in $\mathbb{G}$ is computationally hard. The exact construction was discussed in section 2.8. As $\mathbb{G}$ is cyclic by definition, enumerating the powers of g for $0..\phi(p)$ (where $\phi(p) = p - 1$ is Euler's totient function) gives all possible residues (modulo p) coprime to p exactly once. In other words, iff the exponents $x_1, x_1 x_2, \ldots, x_1 x_2 .. x_n$ of g in X are unique, then the values in X are unique. Given the premise of the birthday paradox, the odds of duplicates in Z the set $x_1, x_2, \ldots, x_n$ is negligible given that our group is large enough. Uniqueness for X is implied as group multiplication is a bijection. As the uniform distribution is identical for each variable $x_1, x_2, \ldots, x_n$, it follows that X has the same distribution as Z. □

Lastly, we need to show that this statement holds if and only if our protocol fulfills the first objective. This is equivalent to proving that $x_{i,j}$ is unique for each message. The first values are picked by the initiating node 0, which picks a random value $x_{0,j}$ for each neighbour j. When a node i receives $g^{x_{i,j}}$, it picks a random $k_{i,j}$ and sends $g^{xk_{i,j}}$ to each neighbour j. As on each hop the exponent is multiplied by a new random factor for each neighbour, we can obtain a set of public keys. The last node picks secret y and computes the shared key, the partial key is returned to the initiator over the same path. For return messages a similar set of public keys can be obtained.

Definition 5.1.5 (Public Key Sets)**.** The propagation of forward messages over a path $\pi = (v_0, \ldots, v_n)$ produces a set of messages each containing a public key. Upon each forward pass, node v_i adds its intermediate key $k_{i,j}$ to the exponent of the public key. This produces the set $X = (g^x, g^{xk_{0,1}}, \ldots, g^{x\Pi^n k_{i,j}})$. For the corresponding echoes, y is used and the intermediate keys are applied in reverse order. The set $Y = (g^y, g^{yk_{n,n-1}}, \ldots, g^{y\Pi^n k_{i,j}})$ corresponds to the public keys of the echoed responses.

Messages in our protocol are unlinkable iff the occurrence of duplicates in $X \cup Y$ is statistically insignificant. As the initiating node must end up with the same key as the other node in the key agreement, the values $k^{i,j}$ picked for a particular message in X are re-used for corresponding return message in Y. This poses no security risk. Recall that the non-initiating node in the key agreement picked y, which results in each element of Y having the same factor y in the exponent, but none contain the factor x. By the reasoning we gave in section 2.8, we know that even though some factors are shared between elements in $X \cup Y$, an adversary is unable to distinguish between elements in $X \cup Y$ and randomly generated variables. As both the public keys and nonces of messages have this type of indistinguishability, it follows any non-trivial pair of messages produced by our protocol are unlinkable by an adversary.

Collusion

Under our security assumption, the second objective holds trivially when $\mathcal{A}$ controls only one node v_A. We have shown messages are unlinkable if their relation is not trivial. The search depth ℓ may give $\mathbb{A}$ some information on the topology besides its direct neighbours. When v_A initiates, $\mathcal{A}$ can derive some information about the density and size of its local neighbourhood. This can even be done by just observing the traffic produced by neighbouring nodes running the protocol. This is arguably not a notable security concern because the functionality of the protocol relies on searching the local neighbourhood. Running instances for increasingly large values of ℓ would be a suitable strategy for an unbounded adversary, but is not feasible for a PPT adversary due to the complexity being exponential in ℓ. The non-trivial scenario we consider is collusion. This typically involves multiple adversaries exchanging information, or may involve a single adversary controlling multiple nodes. For example, an adversary may own more than one bank account. Both scenarios require the same security considerations to be made as data is shared between nodes in either case. We base our proof on the following definition of vertex contraction.

Lemma 5.1.3 (Vertex Contraction)**.** *The vertex pair v_i, v_j can be contracted to a node v for which the set of neighbours $N(v) = N(v_i) \cup N(v_j)$. Similarly, the sets $N(v)^+$ and $N(v)^-$ can be obtained. Next, we alter the graph $G = (V, E)$ by first removing v_i, v_j from the set of vertices and adding v, such that $V' = V \setminus \{v_i, v_j\} \cup \{v\}$.*

Then, for each edge in E we replace occurrences of v_i and v_j with v to obtain E'. This might as a consequence create the edge v, v, which we remove from E' as by our graph definition in section 2.1 self-loops are not allowed.The result is the graph $G' = (V', E')$. Larger sets of vertices can be contracted by sequentially contracting vertex pairs.

Contraction has interesting effects on paths and cycles. Paths can contain any combination of adversarial (A) or honest (H) nodes. Each pattern AH*A with an arbitrary number of adjacent honest nodes, becomes a cycle upon contracting the adversarial nodes.

Corollary 5.1.3.1. *Contracting v_i, v_j to v in a graph containing the path $\pi = (v_i, \dots, v_j)$ produces the cycle $\sigma = (v, \dots, v)$ differing only in its first and last element from π.*

From our perspective, there is a single adversary $\mathcal{A}$ which controls a set of at least two nodes, which we denote by V_A. For a pair of adversarial nodes $v_i, v_j \in V_A$ connected only by a (short) path π, $\mathcal{A}$ is able to find the path length quite easily. Starting with $\ell = 1$, $\mathcal{A}$ initiates instances from v_i and checks if v_i and v_j perform key agreement. This is not the case until $\ell = |\pi|$, from which $\mathcal{A}$ learns the exact length of the path.We show how to reduce an instance in which two nodes collude to a secure single-node scenario in which the adversarial node possesses the same data. The reduction is based on vertex contraction, an operation which merges two nodes and their edges. We finish the proof with an inductive argument, and show that for arbitrary $|V_A|$ the protocol does not reveal non-cycle edges that were previously unknown to $\mathcal{A}$.

Lemma 5.1.4 (Collusion Resistance)**.** *No subset $V_A \subseteq V$ exists which lets $\mathcal{A}$ infer the existence of edges beyond what is collectively known by V_A. Contracting V_A into a single node v_A can be done without impacting security. The reduced graph resulting from this operation has a single adversarial node for which we established inferring edge existence to be infeasible.*

Proof. Let us first consider a graph $G = (V, E)$ which contains the adversarial nodes $v_i, v_j\ inV_A$ and the node $v \in V$. The graph contains no paths connecting v_i to v_j nor does it contain any cycles. The honest node v_n does have a path to v_j, given by $\pi = (v, \dots, v_j)$. The goal of $\mathcal{A}$ is to verify the existence of this path, and determine its length. We allow $\mathcal{A}$ to add exactly one edge to the graph. There are 2 possible edges which $\mathcal{A}$ can add to achieve its goal. The first is (v_i, v), this edge completes the path $\pi' = (v_i, v, \dots, v_j)$ and lets $\mathcal{A}$ derive $|\pi|$. The other edge is v_j, v, which creates the cycle $\sigma = (v_j, v, \dots, v_j)$. Upon executing the protocol for sufficient ℓ, the cycle is found and made public. Either choice results in the same adversarial knowledge. The nodes can be contracted to produce the single-node scenario in which $\mathcal{A}$ cannot infer the existence of edges beyond its direct neighbourhood. □

5.2. Complexity Analysis

We evaluate the performance and complexity of three aspects of our protocol, namely the (i) communication complexity, (ii) computational complexity and (iii) storage requirements. For each concept, we first derive the asymptotic bound from the pseudocode. To validate our claims, we implemented the protocol and carried out a performance evaluation on synthetic graphs. As we are analysing space and computational complexity, recall that the subgroup $\mathbb{G}$ is of order q and is a subgroup of $\mathbb{Z}_p$. The private keys and intermediate keys are chosen from $\mathbb{G}$ and have a bit-length of at most κ_q. For the nonce we consider a security parameter κ_r which denotes its bit-length. The public key g^x, where $x \in \mathbb{Z}_q$ is arbitrary, has a bit-length of κ_p.

Communication Complexity

The number of messages a particular instance generates is largely dependent on the search depth ℓ, and the average degree of nodes participating in the instance. To find the worst-case upper bound on the number of messages, each node should send a maximal amount of messages each time it forwards a message. Maximising the number of neighbours for each node produces a fully connected graph of size n, where $n - 1$ is the outdegree of each node. Given an instance, the initialisation consists of the initiator sending a message to $n - 1$ other nodes. Each receiving node continues to forward the message to another $n - 1$ nodes, creating exponential growth in the number of forward and echo messages for increasing values of ℓ. For example, consider two instances of the protocol for a fully connected graph with $\ell = 1, 2$. For $\ell = 1$ the initiate routine sends $n - 1$ messages which results in $n - 1$ returned echoes. The same messages are sent for $\ell = 2$, but now propagate sends an additional $(n - 1)^2$ forward messages.

Finding the number of messages for echoes is trickier, as a single forward message may lead to multiple echoes that need to be returned over the correct path back to the initiator. For $\ell = 1$ the same $n - 1$ echoes are first returned. For the second instance with $\ell = 2$, nodes receiving a message with $\ell = 0$ cannot send an echo directly to the initiator. Instead, the echo traverses a path of length 2 before it is received by the initiator. This results in an additional $2(n-1)^2$ echoes compared to the first instance. For small ℓ and large n, the worst-case upper bound for both propagate and echo is $O(n^\ell)$.

The last message type is sent by the trace routine. For each found cycle σ_i, trace sends exactly one trace message per edge, for a total of $|\sigma_i|$ messages. The number of trace messages can be expressed as the sum of all cycle lengths $c = \Sigma^i |\sigma_i|$, resulting in a total message complexity of $O(n^\ell + c)$ for the whole protocol. The worst-case bound may be too pessimistic, only few real-life processes can be modelled as a fully connected graph. To find the exact performance, one could find the outdegree of nodes in the local neighbourhood of the initiating node. Computing the average degree for each instance imposes a lot of overhead. We considered using graphs in which the degree is constant as this fully eliminates the overhead. The drawback is that these kind of graphs do not occur in practice. We find a midway; we generate graphs using a consistent degree distribution. For each graph, we run one instance per node and aggregate the results. We further elaborate on the experimental set-up and the results in section 5.3. The main takeaway is that we can use the average degree to obtain a best-effort theoretical bound of $O(d_{avg}^l + c)$. We emphasise that this is an approximation which works well for our small-scale experiments. For large graphs, especially if the the degree distribution is highly divergent, this bound is inaccurate. This is because the message complexity explodes whenever a node with high outdegree is encountered. This phenomenon was the main motivation in [50] to base the approach on indexing such nodes.

Computational Complexity

As discussed in section 2.7, exponentiation is the most expensive operation in modular arithmetic and is dependent on the bit-length of the largest value in the computation. All exponentiations in our protocol take as base g and are performed modulo p. As p has the largest size, the cost of the operation depends on the associated bit-length κ_p. For one operation, the computational complexity is $O(\log^3 \kappa_p)$. Across our algorithms, if a routine performs a constant number of exponentiations the computational complexity can be easily derived from the communication complexity. Exponentiation is performed a linear number of times in the following cases: (i) any node calls initiate or (ii) if $\ell \neq 0$, propagate sends a message to all outgoing neighbours. The total computational cost for each algorithm is determined by the amount of modular exponentiations and the number of invocations. The computational cost is closely tied to the message complexity, as all algorithms besides initiate are invoked upon a node receiving one message. For the same reason, the worst-case topology is again the fully connected graph.

To find the computational complexity for the full protocol, we break down the communication complexity by message type and for each type compute its product with the single-execution computational cost of the corresponding algorithm it invokes. The complete overview is shown in table 5.1. The exception is initiate, which is called only once per instance and performs at most $n - 1$ exponentiations. For propagate it depends on whether the node still needs to forward messages. If this is the case, it requires $n + 1$ exponentiations, and when $\ell = 0$, only 2 exponentiations are needed. For echo the initiating node and non-initiating node perform different computations, but in either case only 1 exponentiation is performed. Non-initiating nodes calling trace perform at most n exponentiations because they have to find the neighbouring node which corresponds to the key in the received message, while the initiator performs none at all.

Storage Complexity

The storage bounds for a single instance are given in Table 5.1 and largely depend on the selection of group parameters. The construction in section 2.8 uses two primes, p and q. These must be sufficiently large, as otherwise attacks may become feasible for an adversary. This poses a trade-off, a higher degree of security requires larger key sizes which in turn consumes more computational power and storage. Security parameters allow for balancing this trade-off. The security of our protocol can be tuned by the security parameters κ_p, κ_q, which control the bit-size of p and q respectively. By definition, p dominates the storage requirements for a single variable as $\kappa_p > \kappa_q$ by our group construction. We also use a third parameter, κ_r, which controls the bit-length of nonces. This parameter is conventionally set to a much lower value than κ_p. To find the global worst-case upper bound storage complexity, we first determine the maximum required amount of storage for a single invocation of each algorithm.

Extending this to the full protocol, recall the number of function calls corresponds roughly to the number of sent messages. The exception is initiate, which is called only once. For the fully connected graph, this routine is called only once and stores $\kappa_q + \kappa_r$ bits for each of its $n-1$ neighbours. The other algorithms do not always store the same amount of data. The propagate routine always uses κ_p bits to store the public key. For each neighbour it propagates a message to it stores approximately $\kappa_q + 2\kappa_r$ in `routes`. When the initiator calls echo it stores the shared key of size κ_p in `keys` if it did not already contain it. Other nodes calling echo store a tuple containing two nonces, a private key and the public key, for a total size of $\kappa_p + \kappa_q + 2\kappa_r$. The trace routine does not store any information. However, given cycle σ_i, the initiator publishes σ_i by broadcasting $|\sigma_i|$ edges after the trace completes. Edges contain two identifiers in the range $1 \ldots n$ which represent nodes. Accounting for the bit-length of n, storing the set σ of all cycles and a total of c edges requires $2c\,\lceil log_2(n) \rceil$ bits. This is mostly insignificant as the bit-length of keys is magnitudes larger than the bit-length needed to represent node identifiers. The propagate routine requires the most storage, as it is called n^ℓ times and stores n values of size κ_q for each call. The storage complexity, for large n, is therefore $O(\kappa_q n^{\ell+1})$ under the assumption that κ_r is small.

Table 5.1: The number of function calls, together with the communication, computational, and storage complexity broken down by algorithm. The propagate routine has two subcases based on ℓ, echo and trace both have different complexity bounds for the initiator and non-initiator nodes. For each complexity type, we state the worst-case bound for one instance of the full protocol. We omit κ_r because of its small size. For exponentiations we substitute $\lambda = \log^3 \kappa_p$ for readability.

	Initialise	**Propagate**		**Echo**		**Trace**		**Protocol**
	any	$\ell \neq 0$	$\ell = 0$	*initiator*	*other*	*initiator*	*other*	*instance*
Function calls	1	n^ℓ	n^ℓ	n^ℓ	n^ℓ	$\lvert\sigma\rvert$	c	
Communication	$O(n)$	$O(n^\ell)$	$O(n^\ell)$	$O(n^\ell)$	$O(n^\ell)$	$O(\lvert\sigma\rvert)$	$O(c)$	$\mathbf{O(n^\ell + c)}$
Exp. per call	$n-1$	$n+1$	2	1	1	0	n	
Exp. per instance	n	$n^{\ell+1}$	n^ℓ	n^ℓ	n^ℓ	0	nc	
Computational	$O(\lambda n)$	$O(\lambda n^{\ell+1})$	$O(\lambda n^\ell)$	$O(\lambda n^\ell)$	$O(\lambda n^\ell)$	$O(1)$	$O(\lambda nc)$	$\mathbf{O(\lambda n^{\ell+1})}$
Bits per call	κ_q	$\kappa_p + n\kappa_q$	κ_p	κ_p	$\kappa_p + \kappa_q$	0	0	
Storage in bits	$O(\kappa_q n^\ell)$	$O(\kappa_q n^{\ell+1})$	$O(\kappa_p n^\ell)$	$O(\kappa_p n^\ell)$	$O(\kappa_p n^\ell)$	$O(1)$	$O(1)$	$\mathbf{O(\kappa_q n^{\ell+1})}$

5.3. Performance Evaluation

The protocol is implemented in C++, and tested on synthetically generated graphs. For our graph generation we use the Barabási-Albert model from [3]. The underlying principle of this model is preferential attachment. New nodes tend to connect to nodes with high degree, and as the generation method adds nodes one by one, this creates a scale-free distribution of degrees among the nodes. Many real-world phenomenons, such as the world-wide web, social networks, biological processes, and financial markets [43], are thought of to be scale-free. However, an empirical analysis performed in [15] showed such networks are rare in the real-world setting. Yet, similarly to how [50] observed high-degree nodes form bottlenecks, we consider the power-law distribution of degrees in our synthetic data to be accurate enough for performing experiments with realistic outcomes.

The model specifies two parameters, m_0 and m_1 for which $m_0 \leq m_1$. These form the basis for our implementation, together with the desired size of the graph n. To start, we generate a fully connected graph (i.e. each node has a node directed to every other node) of size m_0. Then, we create a new node, and probabilistically create m_1 new edges to or from this node. These edges may only occur once, and have an equal chance to be in either direction. This process is repeated until the desired graph size n is reached. The distribution arises from the fact that each node has an equal chance of being sampled by any node in future iterations. In other words, the first m_0 nodes are likely to have much higher degree than nodes created in the last iterations. For our experiments we are interested in the average degree of the graph. We generate graphs using $m_0 = m_1$, and perform measurements for increasing values of m_0. To find d_{avg} we first find the number of edges in the fully connected graph, which has size m_0. There is one edge between each pair of nodes, this equates to $m_0(m_0 - 1)$ edges. The remainder of the nodes together create $m_0(n - m_0)$ new edges. The total number of edges is therefore equal to $nm_0 - m_0$, which lets us compute $d_{avg} = m_0(-n^{-1} + 1)$ as the average degree.

Results

Before we evaluate the performance, we verify the correctness of our protocol by comparing our results with those of standard non-secure cycle detection tools on the same inputs. The group parameters are randomly generated, with $\kappa_p = 60, \kappa_q = \kappa_r = 20$ as bit-lengths. The goal of our experiments is to obtain a benchmark which verifies the time and communication complexity stated in section 5.2. Verifying the bound on the required storage was not part of our set-up, as it can be directly derived from the communication complexity. For each value in the interval $m = [3, 9]$, we generate a new graph with size $n = 50$ and record the total number of cycles $|\sigma_{all}|$. For each value $\ell \in \{2, 3, 4\}$, we execute the protocol once for every node in the graph and record the measurements as an average of the 50 instances. For each instance we sum the length of each found cycle, which we denote by $c = \Sigma|\sigma_i|$. Then, we measure the average execution time of the protocol t_{avg} and record the total number of messages as $|M_{type}|$ for each type of message.

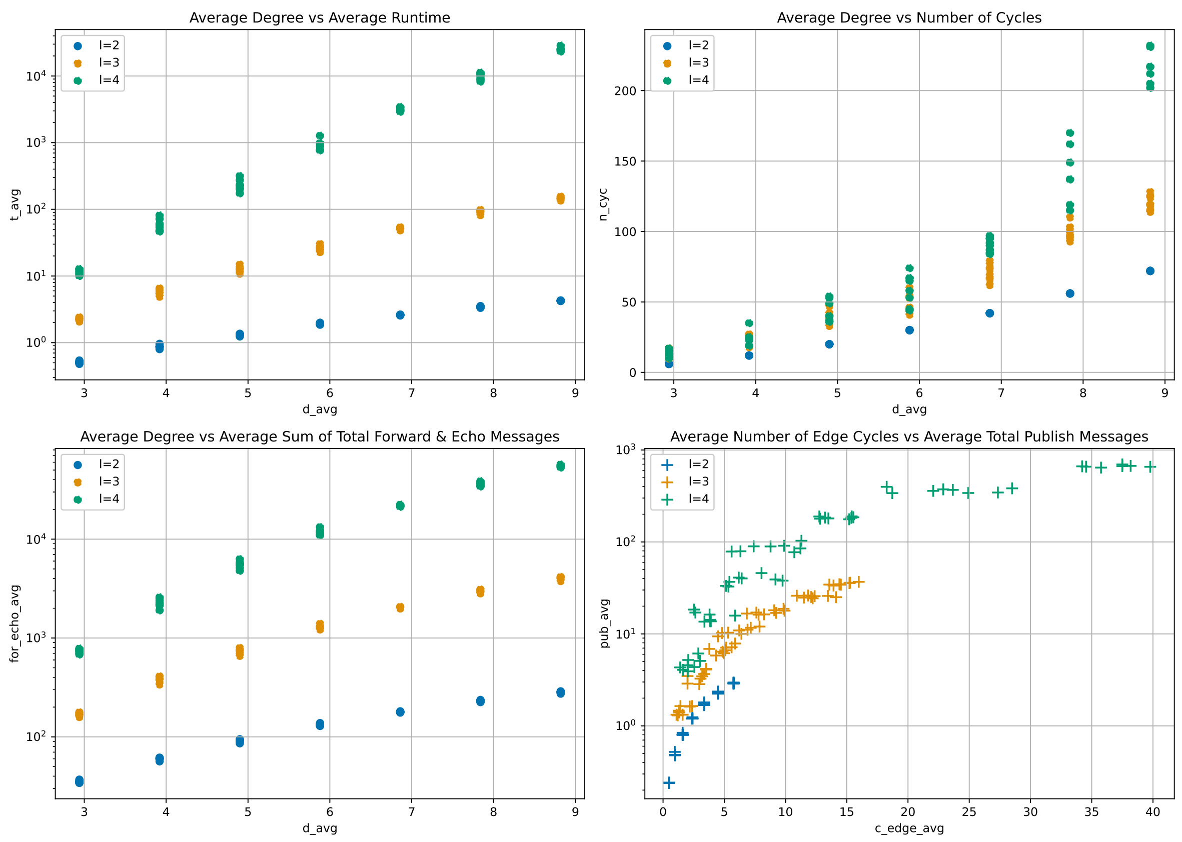

Figure 5.1: Results for $n = 50$, clustered by ℓ. Top left: t_{avg} (log) and d_{avg}. Top right: number of cycles $|\sigma_{all}|$ and d_{avg}. Bottom left: average amount of messages $|M_f \cup M_e|$ (log) and d_{avg}. Bottom right: average amount of messages $|M_t|$ (log) and c.

Our results are split into clusters corresponding to different values of ℓ, and are shown in Figure 5.3. The plots show the following relations: (i) t_{avg}/d_{avg}, (ii) $|\sigma_{all}|/d_{avg}$, (iii) $|M_f \cup M_e|/d_{avg}$, and (iv) $|M_t|/c$. The relation in (ii) is not related to the performance of the protocol as both axes are fixed graph properties, but is included due its significance in the analysis of the other relations. The remaining three plots show our measurements on a vertical log-scale for different graph properties on the horizontal scale. For growing d_{avg}, (i) shows t_{avg} grows polynomially. Likewise, in (iii) the same growth rate can be observed for the sum of forward and echo messages. The growth rate of $|M_t|$ corresponds directly to c, as for each instance, each found cycle results in one trace message for each of its edges. Increasing the number of cycles in a graph increases c as well. From the relation between $|\sigma_{all}|$ and d_{avg} in (ii) we can conclude $|M_t|$ grows for increasing d_{avg}. The rate at which $|\sigma_{all}|$ grows for larger d_{avg} is at least linear, but may be of larger order for increasing ℓ and different types of graphs.

6 Discussion

We start this chapter with a comparison of our original objective in section 1.3 and the results we obtained in chapter 5. After this, we address any findings with scientific significance that are not related to the original research objective. We further discuss some potential changes to make the protocol more efficient and suitable for the dynamic setting.

6.1. Revisited Research Objective

To answer the initial research objective, we decompose the original statement into four sub-objectives we can give a clear answer to. We add a fifth objective regarding the accuracy of our complexity estimations. The objectives are as follows.

1. Can the protocol find all cycles, when given unbounded computational power?
2. For what ℓ does the protocol become too infeasible, when given limited computational power?
3. Is the protocol privacy-preserving?
4. How is efficiency affected by security parameters?
5. Do our experiments show a growth rate matching the theoretical complexities?

The first objective can be answered rather easily. In the analysis in chapter 5 we stated that we verified the correctness by verifying the results with non-secure tools. When running our experiments, we performed a DFS on each graph we generated for our protocol to obtain all cycles. We compared this to the cycles the protocol was enumerating and found the sets to be identical.

The second objective is trickier, because feasibility of running large experiments depends on the capabilities of the machine running the protocol. When adapted for a real-life decentralised setting, the computational load as well as the required storage is distributed across multiple parties. The main bottleneck depends on the time it takes for a message to be sent from source to destination. The results shown in Figure 5.3 at the end of chapter 5 show that increasing ℓ by one leads to the order increasing by one for both run-time and number of messages. For example, the number of messages for a constant d_{avg} is in the order of $10^2, 10^3, 10^4$ for $\ell = 2, 3, 4$ respectively. For our experiments, we tried values $\ell = 5, 6$ which produced instances that timed out after running for 4 hours. Taking communication overhead into account makes these instances run even slower.

The third objective has mostly been discussed in section 5.1 for PPT adversaries. The protocol is not secure against UB adversaries because in this setting the DDH assumption fails to hold. Some other adversarial behaviour for which our protocol is not secure include side-channel attacks, modifying message content sent by either adversarial or honest nodes, and halting execution by blocking network traffic or crashing nodes. Lastly, when the adversary gains access to the secret internal values of some honest nodes, it can use these values to derive internal values of other honest nodes in the network.

The fourth objective has mostly been answered in section 5.3. The same section does fully answer the fifth objective. For a real-life setting, the size of messages can quickly lead to bandwidth becoming a bottleneck, especially because the number of messages grows exponentially. Messages contain public keys of 1024 – 3072 bits, we assume the best-case of 1024 bits and ignore the rest of the message content. For $\ell = 4$ and d_{avg} there are about 10^4 messages exchanged per instance producing 1MB of messages. It is easy to see that increasing $\ell = 5, 6, 7$ produces traffic equal 10MB, 100MB, 1GB per instance. Similarly, increasing d_{avg} produces polynomial growth. For real networks d_{avg} and n are much larger compared to our experimental set-up and there are often nodes with much higher degrees than the average. For example, a node with 10^4 outgoing edges has to send 1MB of traffic for *each* forward message it receives.

6.2. Future Work

When the protocol is executed once for each node in a graph the same cycles are broadcast by multiple nodes. For example, for a graph containing only the cycle $\sigma = v_0, v_1, v_2, v_0$, each node will find the cycle exactly once if $\ell = 3$. The number of broadcasts becomes even larger for $\ell = 6$. This is a result of the unlinkability of messages. Say the initiator is v_0, running an instance with $\ell = 6$ finds the cycle $\sigma' = v_0, v_1, v_2, v_0, v_1, v_2, v_0$. The initiator does not know if a cycle already occurred when receiving a forward message. This can be solved by executing the protocol for each lower value first. This means that the cycle σ was already found before the instance with $\ell = 6$ is initiated. The initiator could use this information to prevent σ' from being detected as a cycle.

Another improvement is related to the static topology of the model. We assume there is no known cycles when initiating the protocol. If we assume all cycles have already been found, we can extend our protocol to the dynamic setting quite easily. Cycles can only be formed when a new node or edge is introduced, which will always be part of the new cycle if it exists. Thus, executing the protocol on the newly introduced node, or the two connected nodes in the case of a new edge, is sufficient to maintain the set of all cycles in the network.

6.3. Concluding Remarks

The importance of AML efforts has grown significantly over the part years as the increasing complexity of financial crimes created new challenges for financial institutions. Traditional systems, which rely heavily on centralised databases and extensive data sharing, cannot be adapted to the cross-organisational scale of modern threats without raising privacy concerns. Our proposed decentralised protocol offers a privacy-conscious addition to the existing AML landscape.

We base our construction on the Diffie-Hellman key exchange, and show how the exchange can be extended to a decentralised setting with multiple intermediaries. The key exchange is used to determine the presence of short cycles in the local neighbourhood of a node, which we efficiently reconstruct and broadcast to other nodes in the network. We prove that the security of the protocol is sound by showing messages are unlinkable and edges stay secret, even when nodes collude. Our performance evaluation shows that the protocol is viable for small cycles and matches the growth of the theoretical complexity bounds. Future research and development should focus on streamlining decentralised AML methods and expanding their capabilities, while emphasising the right to privacy.

References

[1] European Counter Terrorism Centre (ECTC). "Europol joins forces with EU FIUs to fight terrorist financing and money laundering". In: *Europol Media & Press* (Jan. 2016). https://www.europol.europa.eu/media-press/newsroom/news/europol-joins-forces-eu-fius-to-fight-terrorist-financing-and-money-laundering.

[2] Adi Akavia, Rio LaVigne, and Tal Moran. "Topology-Hiding Computation on All Graphs". In: *Journal of Cryptology* 33.1 (Mar. 2019), pp. 176–227. ISSN: 1432-1378. DOI: 10.1007/s00145-019-09318-y.

[3] Réka Albert and Albert-László Barabási. "Statistical mechanics of complex networks". In: *Rev. Mod. Phys.* 74 (1 Jan. 2002), pp. 47–97. DOI: 10.1103/RevModPhys.74.47.

[4] Abdelrahaman Aly. "Network flow problems with secure multiparty computation". PhD thesis. Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2015.

[5] Abdelrahaman Aly and Sara Cleemput. "A fast, practical and simple shortest path protocol for multiparty computation". In: *European Symposium on Research in Computer Security*. Springer. 2022, pp. 749–755.

[6] Abdelrahaman Aly et al. "Securely solving simple combinatorial graph problems". In: *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*. Springer. 2013, pp. 239–257.

[7] Mohammad Anagreh, Eero Vainikko, and Peeter Laud. "Parallel Privacy-Preserving Shortest Paths by Radius-Stepping". In: *2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 2021, pp. 276–280. DOI: 10.1109/PDP52278.2021.00051.

[8] Anders Åslund and Julia Friedlander. *Defending the United States against Russian dark money*. Report. Nov. 2020. URL: https://www.atlanticcouncil.org/in-depth-research-reports/report/defending-the-united-states-against-russian-dark-money/ (visited on 06/30/2024).

[9] Paolo Balboni and Milda Macenaite. "Privacy by design and anonymisation techniques in action: Case study of Ma3tch technology". en. In: *Computer Law & Security Review* 29.4 (Aug. 2013), pp. 330–340. ISSN: 02673649. DOI: 10.1016/j.clsr.2013.05.005. (Visited on 02/22/2024).

[10] Elaine Barker and Allen Roginsky. *Transitioning the Use of Cryptographic Algorithms and Key Lengths*. Mar. 2019. DOI: 10.6028/NIST.SP.800-131Ar2.

[11] BBC. "Russian oligarchs: Where do they hide their 'dark money'?" In: *BBC World* (Mar. 28, 2022). URL: https://www.bbc.com/news/world-60608282 (visited on 06/30/2024).

[12] Guy E. Blelloch et al. "Parallel Shortest Paths Using Radius Stepping". In: *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '16. Pacific Grove, California, USA: Association for Computing Machinery, 2016, pp. 443–454. ISBN: 9781450342100. DOI: 10.1145/2935764.2935765.

[13] Dan Boneh. "The Decision Diffie-Hellman Problem". In: *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*. Ed. by Joe Buhler. Vol. 1423. Lecture Notes in Computer Science. Springer, 1998, pp. 48–63. DOI: 10.1007/BFB0054851.

[14] Fabrice Boudot et al. *Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment*. 2020. arXiv: 2006.06197 [cs.CR].

[15] Anna D. Broido and Aaron Clauset. "Scale-free networks are rare". In: *Nature Communications* 10.1 (Mar. 2019). ISSN: 2041-1723. DOI: 10.1038/s41467-019-08746-5.

[16] Zhiyuan Chen et al. "Machine learning techniques for anti-money laundering (AML) solutions in suspicious transaction detection: a review". en. In: *Knowledge and Information Systems* 57.2 (Nov. 2018), pp. 245–285. ISSN: 0219-1377, 0219-3116. DOI: 10.1007/s10115-017-1144-z. (Visited on 02/29/2024).

[17] European Commission. "Commission welcomes political agreement on the Regulation to establish the new Anti-Money Laundering Authority (AMLA)". In: *Finance News* (Dec. 2023). `https://finance.ec.europa.eu/news/commission-welcomes-political-agreement-regulation-establish-new-anti-money-laundering-authority-2023-12-13_en`.

[18] European Commission. *Frequently asked questions about the new EU Anti-Money Laundering Authority (AMLA)*. 2024. URL: `https://finance.ec.europa.eu/financial-crime/amla/frequently-asked-questions_en` (visited on 06/13/2024).

[19] Ivan Damgård and Jesper Buus Nielsen. "Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption". In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 247–264. DOI: `10.1007/978-3-540-45146-4\_15`.

[20] W. Diffie and M. Hellman. "New directions in cryptography". In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654.

[21] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[22] The United Nations Office on Drugs and Crime (UNODC). *Money-Laundering and Globalization*. Archive. Aug. 2020. URL: `https://www.unodc.org/unodc/en/money-laundering/globalization.html` (visited on 02/21/2024).

[23] Bogdan Dumitrescu, Andra Băltoiu, and Ştefania Budulan. "Anomaly Detection in Graphs of Bank Transactions for Anti Money Laundering Applications". In: *IEEE Access* 10 (2022), pp. 47699–47714. ISSN: 2169-3536. DOI: `10.1109/ACCESS.2022.3170467`.

[24] Andrew Elliott et al. "Anomaly Detection in Networks with Application to Financial Transaction Networks". In: *CoRR* abs/1901.00402 (2019). arXiv: `1901.00402`.

[25] European Commission. *Money laundering*. English. Migration and Home Affairs. Feb. 2024. URL: `https://home-affairs.ec.europa.eu/policies/internal-security/organised-crime-and-human-trafficking/money-laundering_en` (visited on 02/21/2024).

[26] European Commission, Directorate-General for Financial Stability, and Financial Services and Capital Markets Union. *Impact assessment SWD/2021/190 final*. CELEX number 52021SC0190. Brussels, July 2021. URL: `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A52021SC0190&qid=1718281678724` (visited on 06/13/2024).

[27] European Data Protection Supervisor. *Comprehensive Union policy on preventing money laundering and terrorism financing*. Opinion 5/2020. July 2020. URL: `https://www.edps.europa.eu/sites/default/files/publication/20-07-23_edps_aml_opinion_en.pdf` (visited on 06/13/2024).

[28] FATF. *History of the FATF*. English. June 2024. URL: `https://www.fatf-gafi.org/en/the-fatf/history-of-the-fatf.html` (visited on 06/13/2024).

[29] FATF. *Terrorist Financing*. English. June 2024. URL: `https://www.fatf-gafi.org/en/topics/Terrorist-Financing.html` (visited on 06/13/2024).

[30] Robert W. Floyd. "Algorithm 97: Shortest path". In: *Commun. ACM* 5.6 (June 1962), p. 345. ISSN: 0001-0782. DOI: `10.1145/367766.368168`.

[31] W. Fokkink. *Distributed Algorithms: An Intuitive Approach*. The MIT Press. MIT Press, 2013. ISBN: 9780262026772.

[32] M.L. Fredman and R.E. Tarjan. "Fibonacci Heaps And Their Uses In Improved Network Optimization Algorithms". In: *25th Annual Symposium on Foundations of Computer Science, 1984*. 1984, pp. 338–346. DOI: `10.1109/SFCS.1984.715934`.

[33] O. Goldreich, S. Micali, and A. Wigderson. "How to play ANY mental game". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: Association for Computing Machinery, 1987, pp. 218–229. ISBN: 0897912217. DOI: `10.1145/28395.28420`.

[34] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Vol. 2. Cambridge university press, 2001.

[35] László Hajdu and Miklós Krész. "Temporal network analytics for fraud detection in the banking sector". In: *International Conference on Theory and Practice of Digital Libraries*. Springer. 2020, pp. 145–157.

[36] Mark D Harrop and Lemuel Brewster. *Thomson Reuters 2017 Global KYC Surveys Attest to Even Greater Compliance Pain Points*. English. Oct. 2017. URL: https://www.thomsonreuters.com/en/press-releases/2017/october/thomson-reuters-2017-global-kyc-surveys-attest-to-even-greater-compliance-pain-points.html (visited on 02/29/2024).

[37] Saroja Kanchi and David Vineyard. "An optimal distributed algorithm for all-pairs shortest-path". In: *Information Theories and Applications* 11 (Jan. 2004).

[38] Donald E Knuth. "Big omicron and big omega and big theta". In: *ACM Sigact News* 8.2 (1976), pp. 18–24.

[39] Udo Kroon. "Ma^{3}tch: Privacy and knowledge: 'Dynamic networked collective intelligence'". In: *2013 IEEE International Conference on Big Data*. Silicon Valley, CA, USA: IEEE, Oct. 2013, pp. 23–31. ISBN: 978-1-4799-1293-3. DOI: 10.1109/BigData.2013.6691683. (Visited on 02/22/2024).

[40] Miriam Lanskoy and Dylan Myles-Primakoff. "The Rise of Kleptocracy: Power and Plunder in Putin's Russia". In: *Journal of Democracy* 29.1 (2018), pp. 76–85. ISSN: 1086-3214. DOI: 10.1353/jod.2018.0006.

[41] Rutger Leukfeldt and Jurjen Jansen. "Cyber Criminal Networks and Money Mules: An Analysis of Low-Tech and High-Tech Fraud Attacks in the Netherlands." In: *International Journal of Cyber Criminology* 9.2 (2015). DOI: 10.5281/ZENODO.56210.

[42] Yingxin Li, Fukang Liu, and Gaoli Wang. *New Records in Collision Attacks on SHA-2*. Cryptology ePrint Archive, Paper 2024/349. 2024. URL: https://eprint.iacr.org/2024/349.

[43] Felipe Lillo and Rodrigo Valdés. "Dynamics of financial markets and transaction costs: A graph-based study". In: *Research in International Business and Finance* 38 (Sept. 2016), pp. 455–465. ISSN: 0275-5319. DOI: 10.1016/j.ribaf.2016.07.024.

[44] Alfred J Menezes, Paul C van Oorschot, and Scott A Vanstone. *Handbook of Applied Cryptography*. 1st ed. CRC Press, Dec. 1996.

[45] Ian M. Molloy et al. "Graph Analytics for Real-Time Scoring of Cross-Channel Transactional Fraud". In: *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*. Vol. 9603. Lecture Notes in Computer Science. Springer, 2016, pp. 22–40. DOI: 10.1007/978-3-662-54970-4_2.

[46] Foivi Mouzakiti. "Cooperation between Financial Intelligence Units in the European Union: Stuck in the middle between the General Data Protection Regulation and the Police Data Protection Directive". In: *New Journal of European Criminal Law* 11.3 (2020), pp. 351–374. DOI: 10.1177/2032284420943303.

[47] Transactie Monitoring Nederland. *TMNL in brief*. July 2024. URL: https://tmnl.nl/en/about-tmnl/tmnl-in-brief/ (visited on 07/01/2024).

[48] NOS. "Autoriteit Persoonsgegevens kritisch over database betaalgegevens: 'Bancair sleepnet'". nl. In: *NOS Nieuws* (Oct. 21, 2022). URL: https://nos.nl/l/2449281 (visited on 07/01/2024).

[49] Célio Porsius Martins. "Private cycle detection in financial transactions". English. MA thesis. Delft: Delft University of Technology, Jan. 2023. (Visited on 02/09/2024).

[50] Xiafei Qiu et al. "Real-time constrained cycle detection in large dynamic graphs". en. In: *Proceedings of the VLDB Endowment* 11.12 (Aug. 2018), pp. 1876–1888. ISSN: 2150-8097. DOI: 10.14778/3229863.3229874. (Visited on 02/12/2024).

[51] Jian Ren and Jie Wu. "Survey on anonymous communications in computer networks". In: *Computer Communications* 33.4 (2010), pp. 420–431. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2009.11.009.

[52] M Salomon. *Illicit Financial Flows to and from 148 Developing Countries: 2006-2015.* Washington, DC, Jan. 2019. URL: https://gfintegrity.org/report/2019-iff-update/ (visited on 06/13/2024).

[53] Oliver Schirokauer, Damian Weber, and Thomas Denny. "Discrete logarithms: The effectiveness of the index calculus method". In: *Algorithmic Number Theory*. Ed. by Henri Cohen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 337–361. ISBN: 978-3-540-70632-8.

[54] Adi Shamir. "How to share a secret". In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176.

[55] Nigel P Smart. *Cryptography: An Introduction*. en. 3rd ed. Maidenhead, England: McGraw Hill Higher Education, Sept. 2002.

[56] Michele Starnini et al. "Smurf-Based Anti-money Laundering in Time-Evolving Transaction Networks". en. In: *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track*. Ed. by Yuxiao Dong et al. Vol. 12978. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 171–186. DOI: 10.1007/978-3-030-86514-6_11. (Visited on 02/19/2024).

[57] Statista. *Europe: number of banks by country 2024*. en. Accessed: 2024-2-21. 2024. URL: https://www.statista.com/statistics/940867/number-of-banks-in-europe-by-country/.

[58] Robert Tarjan. "Depth-first search and linear graph algorithms". In: *SIAM journal on computing* 1.2 (1972), pp. 146–160.

[59] S. Toueg. *An All-pairs Shortest Path Distributed Algorithm*. IBM RC. IBM Thomas J. Watson Research Division, 1980.

[60] Benjámin Villányi. *Money Laundering: History, Regulations, and Techniques*. Apr. 2021. DOI: 10.1093/acrefore/9780190264079.013.708.

[61] Marc Vorstermans. "Secure Graph Algorithms and Oblivious Data Structures for Multiparty Computation". English. MA thesis. Eindhoven: Eindhoven University of Technology, Mar. 2023.

[62] Stefan Wüller et al. "Using Secure Graph Algorithms for the Privacy-Preserving Identification of Optimal Bartering Opportunities". In: *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*. CCS '17. ACM, Oct. 2017. DOI: 10.1145/3139550.3139557. URL: http://dx.doi.org/10.1145/3139550.3139557.

[63] Andrew C. Yao. "Protocols for secure computations". In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.