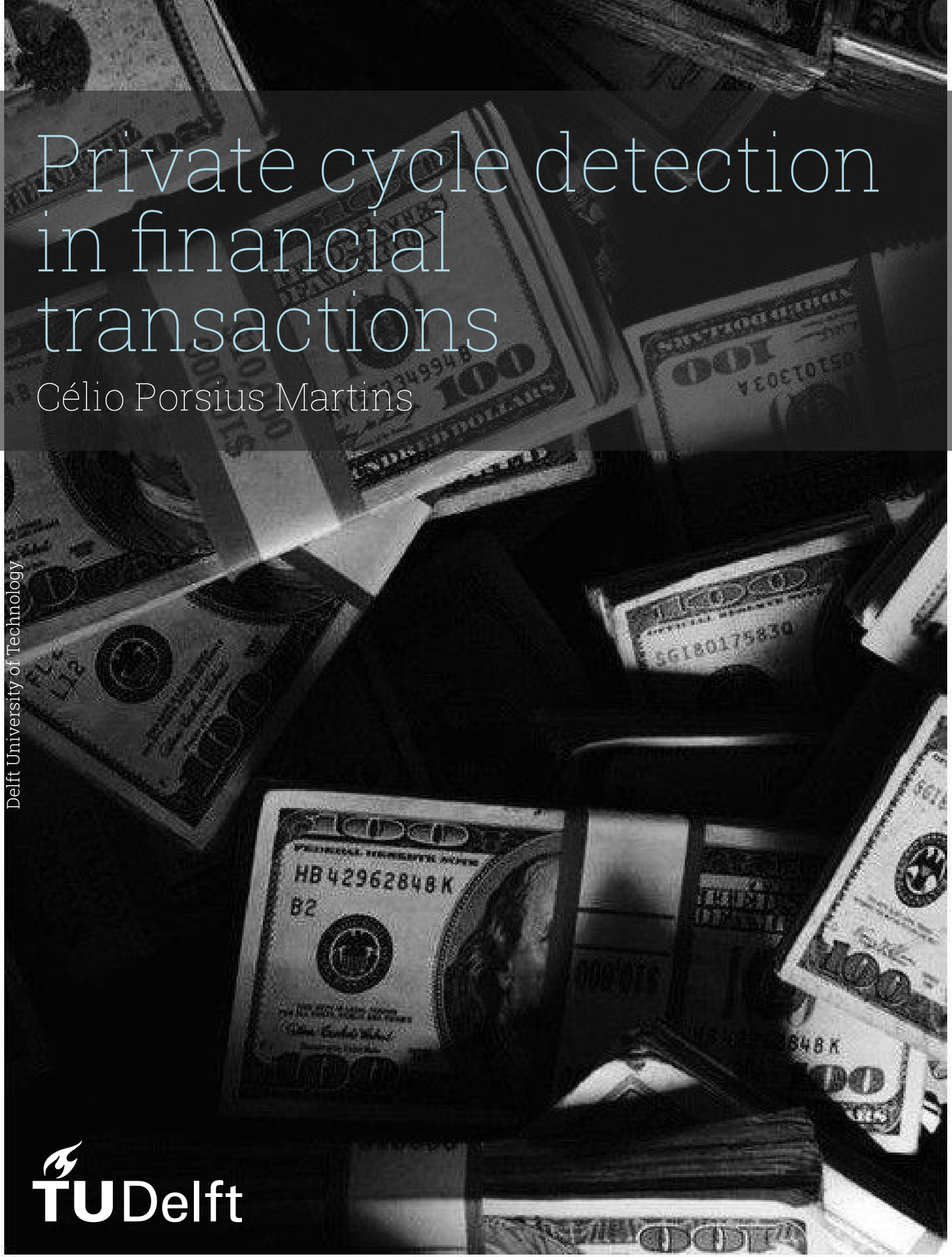
Private cycle detection in financial transactions

Célio Porsius Martins



Delft University of Technology

TU Delft

Private cycle detection in financial transactions

by

Célio Porsius Martins

To obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday, January 11, 2023 at 3:00 PM.

Student number: 4711955

Thesis committee:		
	Dr. Z. Erkin	TU Delft, Supervisor
	Prof. dr. Arie van Deursen	TU Delft
	Prof. dr. Mauro Conti	TU Delft

TU Delft

Abstract

Money laundering is the process of hiding the origin of funds obtained through illicit activities. It is a major problem that has significant impacts on the global financial system and undermines the integrity of financial institutions. To combat this, the Dutch government planning to make it easier for banks to share data to improve the detection of money laundering. However, this approach raises concerns about privacy, as it would allow banks to share sensitive financial information with other banks and institutions. A way to allow banks to still detect money laundering using other banks' data, but without having to share the data would be through the use of multi-party computation. In this work we propose a privacy preserving distributed cycle detection protocol which is meant to find short cycles in financial transactions to help detect money laundering without compromising the privacy of the customers at the bank. Finally, we show that our protocol is significantly faster at detecting short cycles in large financial graphs than current state-of-the-art multi-party computation protocols.

Contents

1

Introduction

Data collection has become increasingly prevalent in modern society, with organizations and businesses gathering vast amounts of personal information from individuals and other sources [3][20][2]. While data collection can provide many benefits, it also poses significant privacy risks and can have serious real-world consequences when data is misused. From oppressive regimes using data to control people's behavior [28], to the Dutch government using data to blacklist people from receiving their benefits [21]. The Dutch government is now considering making it easier for banks to share their customer's bank and transaction information with the goal of improving the prevention and detection of money laundering [29][30]. This banking information is some of the most sensitive information available, as it can for example reveal someone's religion or political views. While preventing money laundering is essential, we should look towards solutions that minimize the impact it has on the privacy of the general population[30]. In this thesis, we show that cycle detection, an operation that banks would like to conduct on this data, can be done in a privacy-preserving manner even on a large amount of data.

1.1. Money laundering

Money laundering is the process of hiding the origin of funds obtained through illicit activities [17]. The term is said to have its origins in the 1920s and 1930s when the mafia in the United States used laundromats for their money laundering schemes [9]. To launder their money they would report their income as coming from cash transactions that happened at these laundromats, making their income seem legitimate.

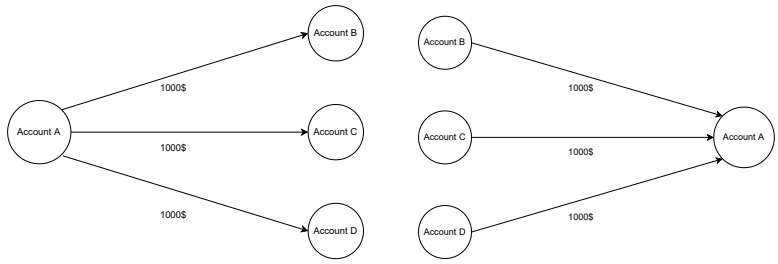
Nowadays, money laundering schemes are often a lot more complex and can generally be split into three stages: placement, layering, and integration [23]. Placement is the first stage of the process, through which the money is moved to the legal financial system. After this stage, the money seems to be acquired through legitimate business practices, e.g. the

laundromats that the mafia used. The second stage, layering, involves moving the funds in order to create more distance from the original source and conceal the money trail. This may involve transferring the money across multiple jurisdictions and countries with strict bank secrecy laws to make it as difficult as possible for authorities to trace the funds. The final stage, integration, involves returning the laundered money to its original source. This stage is the least complex of the three, but also the most challenging to detect, as the origins of the funds have been thoroughly obscured.

1.2. Anti-money laundering

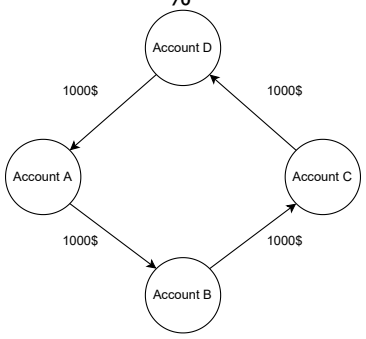
To fight money laundering schemes, the European Union (EU) has put in place several anti-money laundering regulations[14]. One important part of these regulations says that banks must identify and verify a customer's identity, this process is called a know-your-customer (KYC) check. Banks are required by law to perform KYC checks on every new customer, but each bank's process for these checks is slightly different because the law does not specify the exact way banks need to do their KYC process. Lawmakers have, however, written recommendations that banks are generally expected to follow [4]. In the Netherlands, the government expects banks to first get general information about the customer, such as their full name, date of birth, etc. Then, using that information, banks need to make a risk analysis on how likely the customer is to be involved in money laundering activities. If the customer does not pass the risk analysis the bank can request more information or proof before deciding whether to do business with the customer. This risk assessment needs to be repeated periodically to ensure the customer still falls in its original risk category.

Besides performing KYC checks, banks must also monitor all the transactions that they process for unusual transactions and report those suspicious transactions to the financial intelligence unit of the country in which the bank is operating. A financial intelligence unit is a part of the government that each EU country must have, their goal is to determine whether transactions are possibly linked to money laundering or terrorism financing[26]. Once a transaction gets reported to a financial intelligence unit they will conduct an investigation into whether the transaction is suspicious. Financial intelligence units can share information about unusual financial activity with financial intelligence units from other countries and also have access to more information with regard to the people involved in the transactions than a bank would have, such as what other bank accounts they hold. This allows them to do more thorough investigations into the transactions than the banks are able to do on their own. In the end, if the financial intelligence unit suspects a transaction to play a part in money laundering or terrorism financing, they will report it to the police or the secret services who can then start criminal proceedings. However, most transactions never get reported to the financial intelligence unit. Out of the 5.7 billion transactions that happened in the Netherlands in 2021 only 1 million were reported to the financial intelligence unit [27][11]. Thus, most of the work monitoring, filtering, and assessing the risk of



(a) Fan out pattern, where one account sends a lot of smaller transactions to other accounts.

(b) Fan in pattern, where one account receives a lot of smaller transactions from other accounts



(c) Cyclical pattern, where accounts send money to one another creating a cycle.

Figure 1.1: Transaction patterns

transactions happens at the banks.

To monitor the billions of transactions that the banks have to process and to detect unusual activity banks use so-called transaction monitoring systems. These systems detect unusual activities by searching for unusual patterns in the transactions [34]. Examples of patterns that transaction monitoring systems look for are shown in Figure 1.1. Generally, a cycle with less than 7 transactions is a transaction pattern that does not occur much and can be an indicator of fraud or money laundering [31]. A fan-in pattern is when an account receives small amounts of money from many different accounts and a fan-out pattern is when an account sends small amounts of money to many different accounts. These patterns can occur when criminals try to hide large transactions as large transactions are more likely to get reported to the financial intelligence unit. However, which patterns banks look for differs per client, which is why it is important for banks to also understand who their client is and what type of business they do. For certain businesses such as supermarkets, it is for example common to pay out an employee who then spends their money again at the same store. This would create a cycle but is obviously not suspicious. For each abnormal transaction that a transaction monitoring system finds it generates a suspicious activity

report. These suspicious activity reports are then manually checked by an employee of the bank, making the detection of fraudulent transactions very costly.

To make the detection of fraudulent transactions easier, the Dutch government is likely to pass the ”wet gegevensverwerking door samenwerkingsverbanden” (WGS), a law that would allow banks to share the private data of their clients with each other [36]. This law would allow banks to share data once they have a signal that some fraudulent activity is going on. Banks could use this shared data to better determine the origins of a client's money and to detect more complex patterns and cases that would previously have been impossible to detect. For example, a cyclical pattern where the money is transferred between multiple banks is impossible to detect without banks sharing data with each other. To improve this further, the Dutch government is expected to propose the law ”plan van aanpak witwassen” soon, this law would allow the centralized collection and monitoring of transactions [30]. This would mean that all transactions by Dutch account holders would be stored on one central database and all banks would be allowed to run algorithms on it to detect money laundering.

1.3. Privacy

Both the WGS and the ”plan van aanpak witwassen” have raised privacy concerns [29]. The WGS would allow banks and government organizations to share very sensitive data such as a person's criminal record or information about a person's family. Recently with the Dutch childcare benefit scandal, it has been shown how using such sensitive data for important algorithms can lead to algorithms that disproportionally target minorities[21]. Sharing all transaction information in one central database as the ”plan van aanpak witwassen” would allow also adds the risk that if one bank suspects someone to be fraudulent that all the other banks would also refuse to do business with that person [30]. Even for people who are not part of a minority group and do not run the risk of being suspected of being fraudulent by the banks, it is still a great breach of privacy. One's transactions can reveal exactly where they have been, and which political party or religion someone supports [30]. This opens up the possibility of mass surveillance through the use of people's transactions.

Privacy concerns with regard to anti-money laundering legislation are not new, back in 2013 the European data protection board gave an opinion on “the directive of the European Parliament and of the Council on the prevention of the use of the financial system for the purpose of money laundering and terrorist financing” [15] in which the European data protection board said that the directive did not address data protection issues and asked for the introduction of safeguards [37]. Later in 2018, the General Data Protection Regulation (GDPR) came into effect [1]. GDPR was meant to give individuals more control over their own data, improve data protection and give everyone the right to erasure, meaning that individuals have the right to have their personal data erased. Banks are however largely exempt from GDPR since the data that banks collect falls under their legal obliga-

tions. The right to erasure and the right to object, which would allow an individual to object to the processing of their data, do usually not apply to banks [37][15]. Thus leaving room for the individual member states of the European Union to give banks more room to share their customers' data, as the Netherlands is looking to do.

1.4. Secure data sharing

However, there are ways to allow banks to work together without having to naively share their customers' data with other banks and financial institutions. Instead of sharing the data, it is possible to perform operations on data while keeping every bank's data secret through secure multi-party computation. A secure multi-party computation (MPC) is a computation in which parties jointly compute a function while not revealing their own data and inputs to each other. These computations can be accomplished using multiple different techniques such as homomorphic encryption [33] and oblivious transfers[32]. A classic example of a multi-party computation is Yao's Millionaires problem introduced in 1982 by Andrew Yao [40]. The problem describes a scenario of two millionaires who want to know who is richer, but do not want to reveal how much money they have. More formalized the problem is given a and b, determine whether $a > b$, without revealing a and b.
Banks could use similar schemes to collectively detect patterns in transactions without having to outright share customer data. There are multiple multi-party computation protocols that can find the shortest paths in a graph [8][10]. By modeling the customer accounts as vertices and the transactions between them as edges, banks can create a graph that they can use in such protocols. They can also use those protocols to for example find cycles. Once a cycle has been detected and there is a possible suspicion of some sort of criminal activity all the banks could collectively decide to reveal the accounts involved. Limiting the amount of information that gets revealed to the other banks to only the accounts that are suspicious instead of revealing data about all the accounts to find suspicious accounts.
However, a big problem with current multi-party computation schemes that could be used to detect cycles is that they, to the best of our knowledge, all scale quadratically with the total number of vertices in a graph and have a high cost associated with updating the graph. As the graph of all the bank accounts and all the transactions that are happening in the world is very large, there are more than 5 billion bank accounts registered worldwide [12], it is infeasible to use these protocols currently.

1.5. Research objective

This thesis focuses on detecting cycles in financial transactions and improving on the current ways that this can be done using MPC. We chose to focus on cycle detection as that is one of the patterns that are hard to detect for a single bank. As [31] states, a length of 6 is a realistic length for cycles that are interesting for money laundering detection, we focus on finding cycles with a length ≤ 6. To the best of our knowledge, no specialized

protocol for this problem has been proposed in the literature so far. Our goal is to develop a privacy-preserving protocol for finding short cycles that scales better than existing protocols, which leads us to our research question:

"How can we efficiently find short cycles in financial transactions?"

The objective is to design a protocol that does not share any information between the parties unless a cycle has been detected. This does not only include personal information but also the topology of the transactions since this could be used to identify connections and relations between accounts that are not part of the bank.

1.6. Our Contribution

In this thesis, we propose a privacy-preserving distributed protocol that detects cycles with small lengths in transactions that are split between banks without revealing the topology of those transactions. To the best of our knowledge, this is the first time a custom protocol for privacy-preserving cycle detection has been proposed. Our protocol scales exponentially with the length of the cycle that one wishes to detect, but we show that with realistic parameters for cycle detection in financial transactions our protocol scales and performs better than other algorithms which scale quadratically. In our tests, we show that while the other algorithms which can be used to detect cycles can only be executed if the total amount of vertices in the graph is smaller than 20 thousand, our protocol can find whether a vertex is part of one or multiple short cycles in graphs with 1 million vertices in 50 minutes in the worst case with the median case only taking 10 minutes.

1.7. Outline

In Chapter 2, we go over some important concepts necessary to understand the protocols mentioned in this thesis. In Chapter 3, we present the related work to our thesis. In Chapter 4, we describe our privacy-preserving distributed cycle detection protocol. In Chapter 5, we go over our tests and results. And in Chapter 6, we provide our final discussion, possible future work and closing remarks.

2 Preliminaries

In this chapter, we discuss the preliminaries for our thesis, such as one-time pads, Arithmetic Black-Boxes and graph problems and algorithms.

2.1. One-time pad

A one-time pad is an encryption technique that requires a pre-shared key with a length that is at least as large as the message that will be encrypted. It works by then performing an xor operation on each bit of the message and each corresponding bit of the key, the result is the ciphertext. The ciphertext can then later be decrypted by performing an xor operation on each bit of the ciphertext and each corresponding bit of the key, which will result in the plaintext.

The one-time pad operation is widely used in cryptography because it is perfectly secure. Meaning that after applying a one-time pad to a message the resulting ciphertext contains no information about the original message. Essentially making it impossible to find out from the ciphertext what the original message was without knowing the key. To encrypt a message using a one-time pad what one needs to do is convert the message to bits and generate a secret key of at least the same amount of bits as the message. Followed by applying the one-time pad operation to the message and the secret key. The result is the ciphertext and it can be decrypted by doing the same operation again with the same secret key.

The one-time pad operation has a couple of properties. On top of being perfectly secure it is also:

- Commutative, meaning that the order of the inputs can be changed without altering the results. $A \oplus B = B \oplus A$

- Associative, meaning that the order of operations does not matter. $(A \oplus B) \oplus C = A \oplus (B \oplus C)$
- Self-inversable, meaning that if the operation is performed on two of the same values, they negate each other and result in 0. $A \oplus A = 0$
- Has an identity element, 0. $A \oplus 0 = A$

These properties allow us to apply multiple one-time pads one after another and be able to decrypt the ciphertext by applying those same one-time pads in any order.

2.2. Graph problems

2.2.1. Cycle Detection

Cycle detection is the problem of finding a cycle in a graph. Formally, let $G = (V, E)$ be a directed or undirected graph, where V is the set of vertices and E is the set of edges. The problem is to determine whether there exists a path from a vertex $v \in V$ to itself. Many solutions have been proposed to this problem such as Tarjan's strongly connected components algorithm [39] based on depth-first search with a complexity of $O(|V| + |E|)$ and Fleischer et al.[18] divide and conquer based approach with a complexity of $O(|V| \log(|V|))$. Finding a cycle with a length $\leq k$ can also be solved using a depth-first search-based approach by limiting the depth to which the algorithm continues.

2.2.2. Shortest path problem

The shortest path problem is the problem of finding the shortest path between two vertices in a graph, if it exists. Formally, let $G = (V, E)$ be a weighted, directed or undirected graph, where V is the set of vertices and E is the set of edges. Each edge $e \in E$ has a length or weight $w(e)$. The shortest path problem is to find a path P from a source vertex s to a destination vertex t such that the sum of the weights of the edges in P is minimized. This can be expressed mathematically as:

$$P^* = \min_{P \in \mathcal{P}(s,t)} \sum_{e \in P} w(e)$$

where $\mathcal{P}(s, t)$ is the set of all possible paths from s to t. The solution to the shortest path problem is the path P^* that minimizes the sum of edge weights. To solve this problem many different solutions have been proposed, two of which are Dijkstra's algorithm [16] and the Bellman-Ford algorithm [13]. Dijkstra's algorithm has a complexity of $|E|log(|V|)$ which is better than the Bellman-Ford algorithm which has a complexity of $O(|V||E|)$. However, Dijkstra's algorithm does not work for applications in which edges can have negative lengths making the Bellman-Ford algorithm more flexible.
The single source shortest path problem is the problem of finding the shortest path from

one vertex to all other vertices. This can be used to find cycles by first performing a shortest path algorithm and then checking if there is a path from the source vertex to any of its neighbors.

2.2.3. All-pairs shortest path problem

The all-pairs shortest path problem is the problem of finding the shortest path between all pairs of vertices in a graph. Just like for the general shortest path problem, multiple algorithms for this problem have been proposed. Two of these algorithms are Johnson's algorithm and the Floyd-Warshall algorithm [22][19].
The Floyd-Warshall algorithm has a worst-case time complexity of $O(n^3)$. Johnson's algorithm uses the Bellman-Ford algorithm to remove all the negative edges and reweight the graph, after which it uses Dijkstra's algorithm to find the shortest paths. Both of these algorithms can also be used to find the transitive closure of a graph, which is the problem of finding which vertices are reachable from every vertex. Both Johnson's and the Floyd-Warshall algorithms give unreachable vertices a distance value of infinite, meaning that the transitive closure of a graph can be found by identifying all the vertices for which there is a distance between them that is not infinite.

2.3. Secure multi-party computation

Secure multi-party computation (MPC) is a cryptographic technique to allow multiple parties to compute a function $f(x_1, ..., x_n)$ over their inputs x_i without revealing their inputs and without the need for a third party. MPC protocols use cryptographic techniques such as homomorphic encryption and secret sharing to achieve this functionality. We will give a short summary of what homomorphic encryption and secret sharing are.

2.3.1. Secret Sharing

Secret sharing is a cryptographic technique that allows users to split a secret into multiple shares, distribute the shares among different users, and then combine the shares to recover the secret. To reveal the secret a minimum number of shares are required, this is called a threshold denoted as t.
One such secret sharing scheme is Shamir's secret sharing scheme which was introduced in 1979 by Adi Shamir [35]. In Shamir's secret sharing scheme, the secret is represented as a polynomial of degree $t - 1$ in a finite field F. The constant value of the polynomial is the secret value, while the other coefficients are chosen randomly. Shares of the secret are created by evaluating the polynomial at n distinct points $x_1, ..., x_n$ in F, resulting in the shares $(x_1, f(x_1)), ..., (x_n, f(x_n))$.

The secret can be reconstructed from t or more shares by interpolating the polynomial $f(x)$ using Lagrange interpolation. The secret is equal to the constant term of $f(x)$, which

can be computed using the following formula:

$$S = \sum_{i=1}^{t} y_i \prod_{j=1, j \neq i}^{t} \frac{x_j}{x_j - x_i}$$

where S is the secret value, t is the number of shares, and (x_i, y_i) are the coordinates of the ith share.
Linear secret sharing is a type of secret sharing in which users can perform linear operations on their shares [24]. This technique can be used to achieve secure multi-party computation (MPC), where multiple users can jointly compute a function on secret inputs without revealing the inputs to each other. To perform MPC using linear secret sharing, the secret inputs $x_1, ..., x_n$ are first secret shared among all the users. Then, each user applies the function f to their shares, and the result of $f(x_1, ..., x_n)$ can be recovered by combining the resulting shares.

2.3.2. Homomorphic encryption

Homomorphic encryption is a form of encryption that allows operations to be performed on ciphertext, with the resulting ciphertext corresponding to the same operations performed on the plaintext. For example, given two ciphertexts c_1 and c_2 encrypted using homomorphic encryption, multiplying c_1 and c_2 will result in a new ciphertext c_3 that, when decrypted, will contain the sum of the original plaintexts corresponding to c_1 and c_2. The operations that can be performed on the ciphertext depend on the homomorphic encryption scheme that is used. Partially homomorphic encryption schemes are schemes that allow only one kind of operation to be performed on the ciphertext, such as addition or multiplication. Meanwhile, fully homomorphic encryption schemes allow multiple different operations to be performed on the ciphertext. Generally, operations using partially homomorphic encryption schemes are faster than those same operations on fully homomorphic encryption schemes due to fully homomorphic encryption schemes being more complex.
Homomorphic encryption can be used for secure multiparty computation by allowing multiple users to encrypt their secret inputs using a common public key. The encrypted inputs can then be shared among the users, who can jointly compute a function on the encrypted data without revealing the underlying plaintexts to each other. The result of the computation can be obtained by sending the resulting ciphertext to a user who holds the secret key, who can then decrypt it to reveal the result of the function applied to the original inputs.

2.3.3. Arithmetic Black-Box

In multiparty computation, operations on encrypted values can be performed using various techniques, such as homomorphic encryption [33] or secret sharing [24]. An arithmetic black box is an ideal functionality that can store and return encrypted information as well as perform arbitrary arithmetic operations on the encrypted data without revealing any information about the underlying plaintexts. This concept allows algorithm designers to

focus on the design of the algorithms without worrying about the details of the underlying techniques. It also allows the same algorithm to be securely implemented using different techniques, depending on the specific security requirements and threats.

3

Related work

In this chapter, we look at other works on detecting cycles in a privacy-preserving manner. We first look at work done on cycle detection in financial transactions. Since to the best of our knowledge, there are no papers about secure cycle detection papers in a multi-party setting, we then look at privacy-preserving ways to find the shortest path between two points and at Topology-hiding computation which is a technique to hide the topology of a network during a multi-party computation protocol.

3.1. Cycle detection in financial transactions

In 2018 Qui et al.[31] described a system called GraphS which is used by Alibaba to actively monitor and detect fraudulent transactions based on cycle detection. As the name implies GraphS models all the financial transactions and accounts as a graph. Accounts are modeled as vertices and transactions and relations as edges. Relations are for example a friendship relation between two accounts or an ownership relation if one person owns two accounts. Relations are modeled as static edges, while transactions are modeled as dynamic edges and disappear after a certain amount of time. From the graph GraphS then marks the vertices with the highest amount of outgoing edges as hot points and computes all the possible paths between the hot points. Then to find the cycles, GraphS takes as input a maximum length k and starts by performing a depth-first search from the source vertex. During the depth-first search, a branch stops if:

1. The destination is reached.
2. The maximum length is reached.
3. A hot point is reached.

For each hot point that is reached GraphS records all the paths from the source vertex to the hot points. Following this step, GraphS use the reverse graph, which is the graph but with all the edges reversed, and perform another depth-first search from the source vertex. Now they have all the paths from the source to the hot points and all the paths from a hot point to the source vertex, since all the paths between hot points are precomputed they can now find cycles by only checking whether paths exist between hot points that have a path to the source and hot points that have a path from the source.
In the experiments, they show that a query with a k of 6 takes less than 10 milliseconds to complete in 99.9 percent of cases on a graph with over half a billion vertices and nearly 2.1 billion edges. Their system has a throughput of almost 16 thousand transactions per second when running on an Intel(R) Xeon(R) E5-2650 server with 32 cores and 128GB of memory. However, it requires that one party has all the data and can not find cycles once transactions leave their system. Because it is assumed that all the data is owned and controlled by one party they also pay no attention to privacy.

3.2. Secure Shortest Path protocols

This section introduces previous work on secure shortest path protocols. Secure shortest path protocols can be used to find the shortest cycle in a graph. Each of these protocols returns a secret shared array of distances to each vertex from a chosen source vertex. This array can be used to determine whether the source vertex is part of a cycle by revealing the distances from the source vertex to each of the vertices that have an edge to the source vertex. If this value is infinite then there is no cycle if the value is anything else there is. We also provide a comparison of all the protocols discussed in this section in Table 3.1.
Aly et al. [7] proposed two single source shortest path protocols modeled as arithmetic black-boxes one based on Dijkstra's algorithm and the other based on the Bellman-Ford algorithm. Both of these protocols find the shortest path from the source vertex to all other vertices. The values returned by the protocol are the secret shared paths and the secret shared distances to each other vertex from the source vertex.
Their protocol based on Dijkstra's algorithm performs $O(n^3)$ multiplications and $O(n^2)$ comparisons. The protocol based on the Bellman-Ford algorithm requires $O(n^3)$ comparisons, multiplications, and additions. All these operations require at least $O(1)$ communication rounds meaning that the round complexity is also $O(n^3)$ for both of these protocols. In terms of privacy, these protocols are as secure as the underlying protocols that are used for the operations.
Aly and Cleemput [8] directly build on top of [7] and propose another secure shortest path protocol modeled as an arithmetic black-box based on Dijkstra's algorithm. They improve the complexity of the protocol and require $O(n^2 log(n))$ multiplications, which can be parallelized to only need $O(n^2)$ rounds of communication.

Anagreh et al.[10] further improved on Aly et al.[7] and Aly and Cleemputs [8] work. In their work, they propose two other secure shortest path protocols again based on Dijkstra's protocol and the Bellman-Ford protocol. They improved the round complexity of the Dijkstra-based protocol to only need $O(nlog(n))$ rounds and with a total bandwidth of $O(n^2)$. The Bellman-Ford protocol was also improved to have a round complexity of $O(nlog(n))$ with the total bandwidth used being $O(mn)$. Additionally, they show that for the Bellman-Ford based protocol the total bandwidth can be reduced to $O(m(k+log(n)))$ in $O(klog(n))$ rounds if the shortest path is known to be of a length shorter than k. They also give a possible different version of their Bellman-Ford protocol which reduces the round complexity in half of the original protocol but increase the bandwidth usage by a factor of $log(n)$.
In their tests they show that the graphs on which the protocol can be executed can be significantly larger than the protocols proposed by [7] and [8], testing their Dijkstra's based protocol on graphs with up to 15 thousand vertices and 112 million edges and taking 13.4 thousand seconds to complete the protocol execution. For their Bellman-Ford based protocol they test on graphs with up to 9.5 thousand vertices and 500 thousand edges and show that their protocol takes 151 thousand seconds to complete. Both of these protocols also require significant precomputation, the Dijkstra's based protocol requiring $O(n^2)$ bandwidth and the Bellman-Ford based protocol requiring $O(mlog(n))$ bandwidth.

Anagreh et al.[10] also proposed four privacy-preserving all pair shortest distance protocols of which 2 based on the Johnsons algorithm, one based on the Floyd-Warshall algorithm, and a transitive closure protocol. The all pair shortest distance protocols return the distances between each pair of vertices meaning that one can use the result to find the cycle that each vertex is involved in in a single execution of the protocol. Both the protocols based on Johnson's algorithm have a bandwidth consumption of $O(n^3)$ with a round complexity of $O(log(n))$. The difference between their two Johnson based protocols is that the version 1 Johnson protocol uses the standard version of their Bellman-Ford protocol and the version 2 Johnson protocol uses the version of their Bellman-Ford protocol with reduced rounds of communications but with increased bandwidth usage. The Floyd-Warshall based protocol also has a bandwidth consumption of $O(n^3)$, but with much better constant factors and a round complexity of $O(n)$. Lastly, the transitive closure protocol has a bandwidth consumption of $O(n^3log(n))$ with a round complexity of $O(log^2(n))$.

3.3. Topology-hiding computation

Topology-hiding computation is a cryptographic technique that aims to hide the network topology in secure multiparty computation (MPC) protocols. This is different from the problem addressed in this thesis, which focuses on hiding information about the data and transaction graph held by banks in a network.

Topology-hiding computation was first proposed by Moran et al. [25], who showed

Table 3.1: Summary of all the protocols mentioned

Protocol	Rounds	Bandwidth	Finds all paths
Dijkstra's [7]	$O(n^3)$	$O(n^4)$	No
Dijkstra's [8]	$O(n^2)$	$O(n^3)$	No
Dijkstra's [10]	$O(nlog(n))$	$O(n^2)$	No
Bellman-Ford [7]	$O(n^3)$	$O(n^4)$	No
Bellman-Ford [10]	$O(nlog(n))$	$O(mn)$	No
Bellman-Ford for depth k [10]	$O(klog(n))$	$O(m(k+log(n)))$	No
Johnson [10]	$O(log(n))$	$O(n^3)$	Yes
Floyd-Warshall [10]	$O(n)$	$O(n^3)$	Yes
Transitive closure [10]	$O(log^2(n))$	$O(n^3log(n))$	Yes
Our protocol	$2l$	$O(d^l)$	No

that it is possible to hide the topology of the communication graph underlying an MPC protocol from semi-honest adversaries if the diameter of the graph is $O(log(n))$, where n is the number of nodes in the network. They proposed a protocol in which each node and its neighbors, called a neighbourhood, perform a local MPC protocol, and the result is secret-shared with the other neighbourhoods in the network.

This was later improved by Akavia et al. [6] to work on networks with cycle or tree topologies, regardless of the diameter. Akavia et al. showed that using a homomorphic, privately key-commutative, and re-randomizable encryption, each node can perform an OR operation on an encrypted bit and add another layer of encryption before sending it to its neighbors. After the bit has been sent to the next node the same number of times as the depth of the tree or the length of the cycle, the decryption is performed by every node removing the layer of encryption that they added and sending the encrypted bit back to the initial sender. After each layer of encryption is removed, the original sender knows the result of the OR operation.

This was further improved by Akavia et al. [5] to work on arbitrary graphs by using correlated random walks to send the encrypted bit across the network.

4

Privacy-preserving distributed cycle detection

In this chapter, we introduce our privacy-preserving distributed cycle detection algorithm that is tailored toward financial transactions. We describe the assumptions, the entities and the protocol itself in this chapter.

4.1. Setting and assumptions

Before presenting the protocol, we will describe the setting and the cryptographic assumptions. The setting includes all of the entities involved in the protocol, their capabilities, and the relationships between them.

4.1.1. Entities and relations

Banks are entities that manage accounts for their clients. When a client requests a transaction, the bank of the sender account works with the bank of the receiving account to execute the transaction. We assume that banks are aware of all transactions sent and received by the accounts they manage.
Accounts are bank accounts managed by a bank but owned by a client of the bank.
Transactions are the transfer of money from one account to another and are performed by banks on behalf of their clients.

4.1.2. Assumptions

General assumptions

In our protocol, we assume banks can add or remove accounts at any point during the protocol's execution. Additionally, we assume that new transactions may be made and

added to a bank's database, or removed from the database, during the execution of the protocol. We also assume that the accounts involved in any transaction are valid and exist.

Cryptographic assumptions

In our protocol, we assume that each bank is semi-honest, meaning that they will try to learn as much information as possible but will not deviate from the protocol. We also assume that each bank has a line of communication with every other bank and that each message's integrity, authenticity, and security are guaranteed.

4.2. Objectives

The goal of our protocol is to protect the privacy of banks and their clients. Specifically, we aim to protect the information about a bank's clients' transactions and the number of clients that a bank has from the other banks participating in the protocol. Hence we aim to satisfy the following criteria:

- Banks should be unable to learn of the existence of bank accounts.
- Banks should be unable to learn of the existence of transactions between bank accounts.

4.3. Design

In this section, we present the design of our algorithm. We define the banking system as a graph and then use those definitions to write our protocol. The protocol uses a directed graph, $G = (V, E)$, to represent the banking system. The set of vertices, V, represents accounts, and the set of edges, $E \subseteq V \times V$, represents transactions. Each edge (u, v) in the graph is associated with a secret 128-bit integer value, $s(u, v)$, which could be a hash of the transaction, for example.

A vertex v is an out-neighbor of another vertex u if there exists an edge (u, v) in the graph. The set of all out-neighbors of v is denoted as $N^+(v)$. A path p is a sequence of unique vertices $v_1, ..., v_n$. A cycle is a path for which the first and last vertices, v_1 and v_n, are the same.

The set of banks is represented as a partition of the graph, $B = B_1, B_2, ...$ Each $B_i \in B$ represents an individual bank. The bank that a given vertex v belongs to is denoted as $b(v)$. Each bank B_i also maintains a set, S_i, to keep track of the random values which the bank has generated during the protocol.

The idea behind our protocol is to detect cycles in the graph by flooding it with a special value from the nodes that a bank wants to check for cycles. If the same vertex receives the value back, it means there is a cycle involving that vertex. Since there is no communication between the accounts, only between the banks it means that banks could learn of the

Table 4.1: Summary of the symbols.

Symbol	Definition
G	Graph of all accounts and transactions
V	The set of vertices
E	The set of edges
B	A partition of G
v	A vertex
(u, v)	An edge from vertex u to vertex v
B_i	Bank number i
$b(v)$	Bank that Vertex v is a part of
$N^+(v)$	The outgoing neighbors of v
$s(u, v)$	Random 128 bit value associated with edge (u, v)
M	A message
M_v	A message with target v
l	Maximum cycle length
R	Random 128-bit value
S_i	The set of generated numbers for B_i

existence of certain transactions by seeing if they receive the special value on any of the other vertices as shown in Figure 4.1. To prevent other banks from learning about the existence of certain transactions, we encrypt each message with a one-time pad before sending it. However, if a message follows the same path twice, the one-time pads will cancel each other out and the message will become recognizable again. This can only happen if there is a cycle, so if a bank detects the original message, it means a cycle has been found. To control the maximum length of the cycles that a bank wants to find we also sent a length along with the message, which gets decreased each time it gets sent to the next vertex. When a bank receives a message with the length zero it will stop forwarding that message.

To find all the accounts that are part of a cycle, all the banks need to store the values that they receive and the bank that each value came from. After the protocol has finished, the initiating bank can communicate with the bank that it received the last message from for each cycle found, and ask that bank where the message was received from. This process can then be repeated, with each bank communicating with the bank that it received the previous message from, until all the banks involved in the cycle have been asked. At this point, the banks can report the accounts involved to each other and reveal the full set of accounts involved in the cycle. This would allow the banks to uncover all the accounts involved in a cycle, without revealing any additional information about accounts or transactions that are not part of the cycle.

Input: Vertex u, Length of the cycle l and Banks B
Output: Integer $c =$ amount of cycles u is a part of.

1. A bank B_i generates 128 bits of randomness R.
2. Add R to the banks set S_i.
3. For each $v \in N^+(u)$, B_i generates a message M_v as follows:
$$M_v = R \oplus s(u, v)$$
4. For each $v \in N^+(u)$ B_i sends message $M_v, 2l$ and v to $b(v)$.
5. Upon receiving a message a bank B_j generates a message M_w for $w \in N^+(v)$ if $M_v \notin S_i$ and $l \neq 0$:
$$M_w = M_v \oplus s(v, w)$$
6. B_j sends message $M_w, (l-1)$ and w to $b(w)$ for each $w \in N^+(v)$.
7. Repeat from step 5 until no more messages get sent.
8. B_i **outputs** the number of messages received which were in S.

Protocol 1: Privacy preserving distributed cycle detection

4.3.1. Protocol efficiency

The total bandwidth usage of this protocol is $O(d^l)$ for degree d as there is one message for each outgoing neighbor of a vertex and the protocol has $O(l)$ rounds. The computational complexity is also $O(d^l)$ as each message has one one-time pad operation applied to it which takes $O(1)$ time. To find all the cycles that each vertex is a part of we would have to execute the protocol for each vertex increasing the bandwidth and computational complexity by a factor n, but the round complexity stays the same as all the executions can happen in parallel allowing the banks to send all the messages at the same time. While the protocol bandwidth and computational complexity are exponential as noted by [31] the length of typical cycles to look for in financial transactions are small being ≤ 6 additionally the degree of a vertex is also shown to be relatively small with 80 percent of the vertices having a degree < 10. This makes our protocol much more efficient in real-world scenarios.

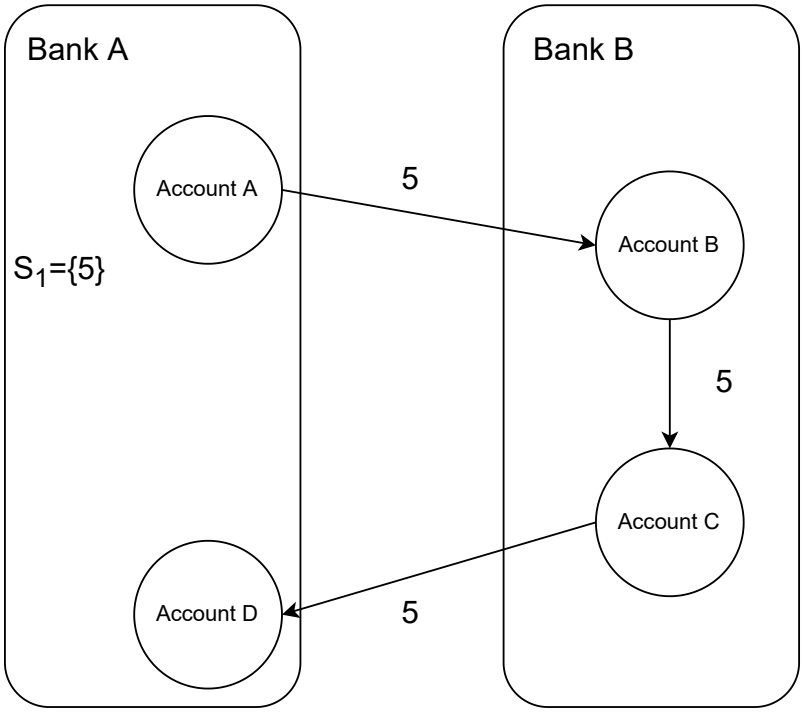


Figure 4.1: Bank A sends out 5 as a special value from account A to account Band then receives 5 on account D from account C. From this Bank A learns the message must have gone from B to C.

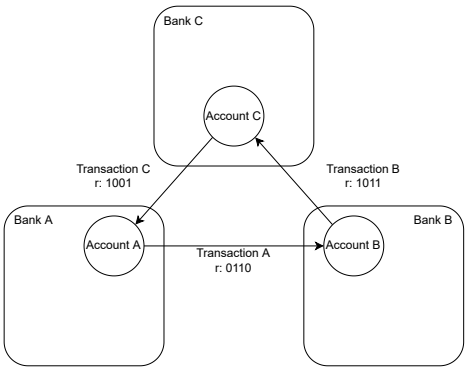
4.3.2. Security analysis

In this section, we argue that the protocol is secure in the semi-honest model with private channels, we also mention a couple of possible attacks against the protocol and discuss those.

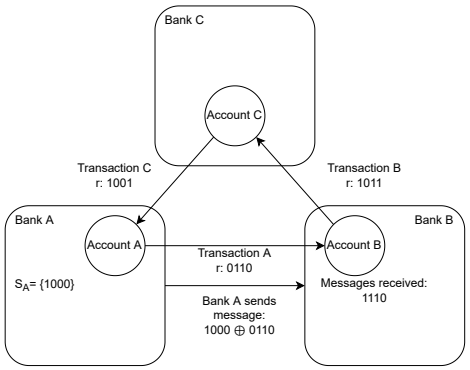
During the protocol, the banks can be split into two groups, the initiating bank, which starts the protocol and forwards messages, and the forwarding banks which only forward messages. The initiating bank knows the random value, the length that it sends with the first messages, all the messages that it sends, all the messages that it has received, and the respective lengths for the sent and received messages. In contrast, forwarding banks only have knowledge of the messages they receive and send, as well as the lengths of those messages.

For a bank to be able to learn new information about the graph, it needs to know where a received message came from. However, the messages sent and received by both the initiating bank and the forwarding banks are encrypted using a one-time pad, which removes any information about the previous message from the received messages. This means that as long as there is at least one one-time pad applied that the bank does not know the key for, the bank cannot learn where the message came from. If the bank does know the

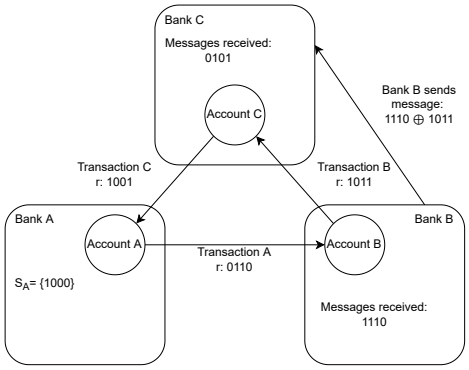
key, it means that they already knew about the edge and therefore already knew about that transaction. In this case, the bank would not learn any new information.
The length that is attached to messages can be used to identify a message. Looking at the amount of time that has passed and the length, a bank can estimate the expected amount of times that the protocol has been started around the time that they started it themselves. Using this information a bank can guess whether a message is their message with a probability of $\frac{1}{\text{expected amount}}$ which would be better than random guessing unless each bank initiates the protocol for every account all the time.
If a bank can guess that a message is one that they receive they learn that there is a series of transactions that goes from the account from which the message was originally sent to the account on which it is now received. This reveals information about transactions that it previously did not know existed.
To prevent this type of attack, banks could add random delays to the messages that they send. This would make it harder to estimate when a message was sent, and would therefore decrease the attacker's ability to identify messages. However, even with a delay, a bank could still estimate the total number of times that the protocol has been initiated and then guess with a probability of $\frac{1}{\text{total amount}}$. For this not to be better than random guessing, the protocol needs to be initiated at least as often as there are vertices over the delay amount that is chosen.
The additional information that the initiating bank has in comparison to the forwarding banks, namely the random value it has generated, can also not be used to link two messages together since each message gets encrypted with a one-time pad all information with regards to the previous value is removed from the messages and thus makes it impossible to use the knowledge of the generated value to link messages together or identify messages as being sent by the initiating bank.



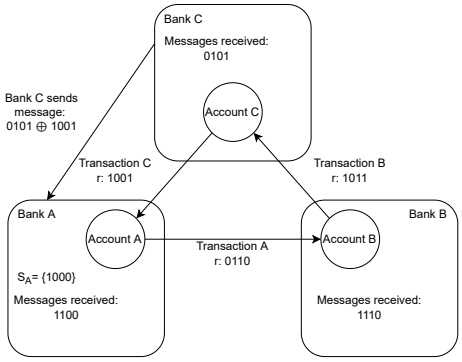
(a) The initial state



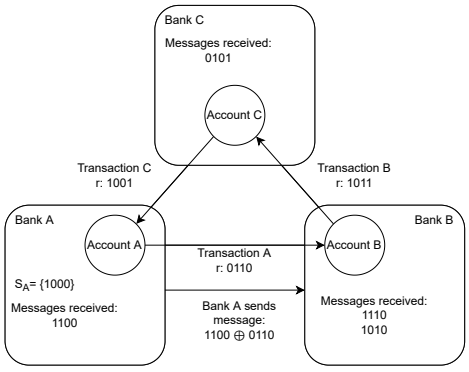
(b) Bank A initiates the protocol for Account A, % generates randomness R and sends the first message.



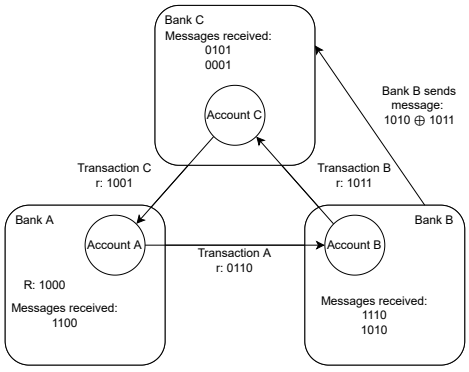
(c) Bank B applies a one-time pad and forwards the message.



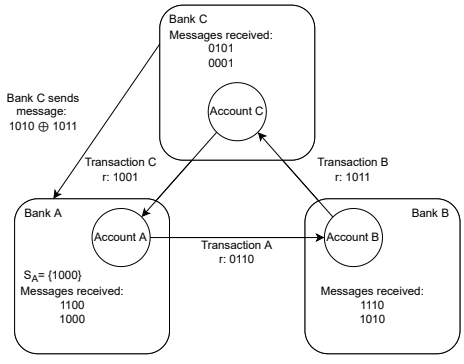
(d) Bank C applies a one-time pad and forwards the message.



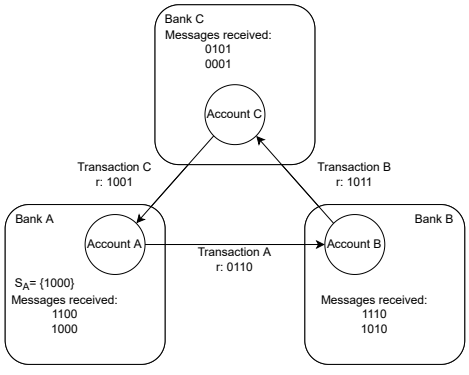
(e) Bank A does not recognize the message, so it applies a one-time pad and forwards the message.



(f) Bank B applies a one-time pad and forwards the message.



(g) Bank C applies a one-time pad and forwards the message.



(h) Bank A recognizes the last received message and stops forwarding it.

Figure 4.2: Example protocol execution with a cycle of length 3.

5

Results

In this chapter, we discuss the results of our experiments. For our experiments, we used Google Cloud and ran our experiments on an n2-standard-2 instance powered by Intel Cascade Lake CPUs[1]. All our experiments were performed on one core with no network delay and instant communication. For our experiments, we measured the total work and the total worst-case bandwidth. For our experiments, we use datasets from AMLSim [38]. AMLSim is a simulation tool built to simulate financial transactions and known money laundering typologies. AMLSim has provided datasets with already transaction graphs with suspicious patterns in them, namely fan-in, fan-out and cyclical patterns. We use these datasets for our analysis. The datasets have graph sizes ranging from 100 accounts with 10 thousand transactions to 1 million accounts with 100 million transactions.

5.1. Runtime on different graph sizes

To measure the effect of different graph sizes on our performance we tested our protocol for a cycle length of 5 on 3 different graph sizes from 100 accounts and 10000 transactions to 1 million accounts and 100 million transactions, each being 100 times larger than the previous one. We ran our tests by randomly selecting accounts for which we wanted to detect cycles from the graph, for the larger graphs we chose 1000 accounts, and for the small graph we ran the protocol for all 100 accounts. The results in Figure 5.1 show that with the graphs becoming 100 times larger the time it takes to run the protocol increases by one order of magnitude. This is likely because the bigger graphs have more accounts with a high degree, once those are reached the time one execution takes increases dramatically.

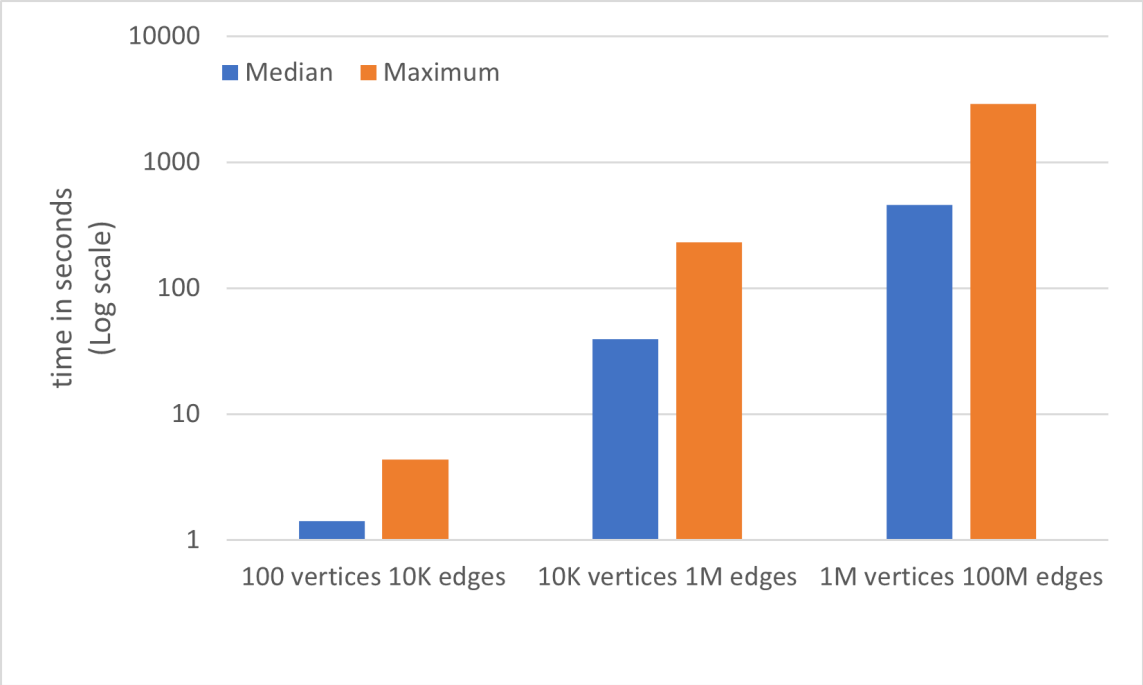


Figure 5.1: Total run time on differing graph sizes, averaged over 1000 runs, except for the graph with 100 accounts for which we only did 100 runs.

5.2. Runtime for different cycle lengths

Cycle length is the main bottleneck for our protocol. To measure the impact of the cycle length on performance we ran our protocol on the graph with one million accounts and 100 million edges with varying cycle lengths. As can be seen in Figure 5.2, for each increase in the cycle length, the execution time of our protocol increased by two orders of magnitude.

5.3. Bandwidth use for differing cycle lengths

Another important metric besides the run time is the amount of bandwidth the protocol requires. All of our tests were performed without taking bandwidth into account however, in real-world scenarios bandwidth is often limited and can be a bottleneck. To test this we ran our protocol on the graph with one million accounts and 100 million transactions with differing cycle lengths.

Our tests show that bandwidth scales in the same way as the run-time, meaning that every increase in cycle length causes our bandwidth to increase by two orders of magnitude. Our bandwidth experiments show that detecting cycles of length 5 uses about 6 Terabytes

[1] https://github.com/C3lio/RustCycleDetection

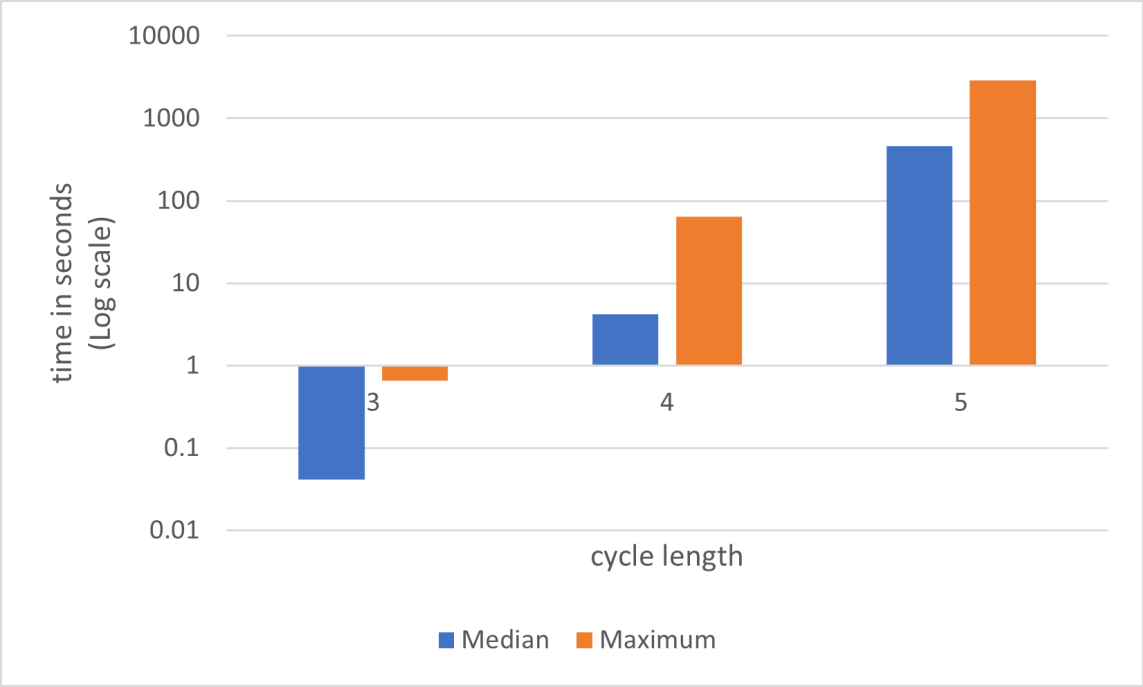


Figure 5.2: Total run time for differing maximum cycle lengths, averaged over 1000 runs.

of bandwidth in the worst case. With 100 Gigabyte Ethernet, this can be transferred in 8 minutes, which would increase our worst-case time by 16 percent.

5.4. Comparison

Table 5.1: Time comparison of our protocol with the shortest path protocols mentioned in [10] on graph sizes of around 100 vertices

Protocol	Vertices	Edges	time precomp. (s)	time total (s)
Dijkstra's [1]	10 thousand	49.9 million	1572.9	6061.6
Bellman-Ford [1]	9500	500 thousand	6600	151 thousand
Our protocol	10 thousand	1 million	0	232
Our protocol	1 million	100 million	0	2910

In Table 5.1, we compare our results to the results from [10] on differing graph sizes. Overall it shows that even on the largest graph the median time to complete a cycle detection with $length = 5$ only took 457 seconds with the worst case being 2910 seconds. We were not able to reproduce the results from [10] but if we compare our results on the

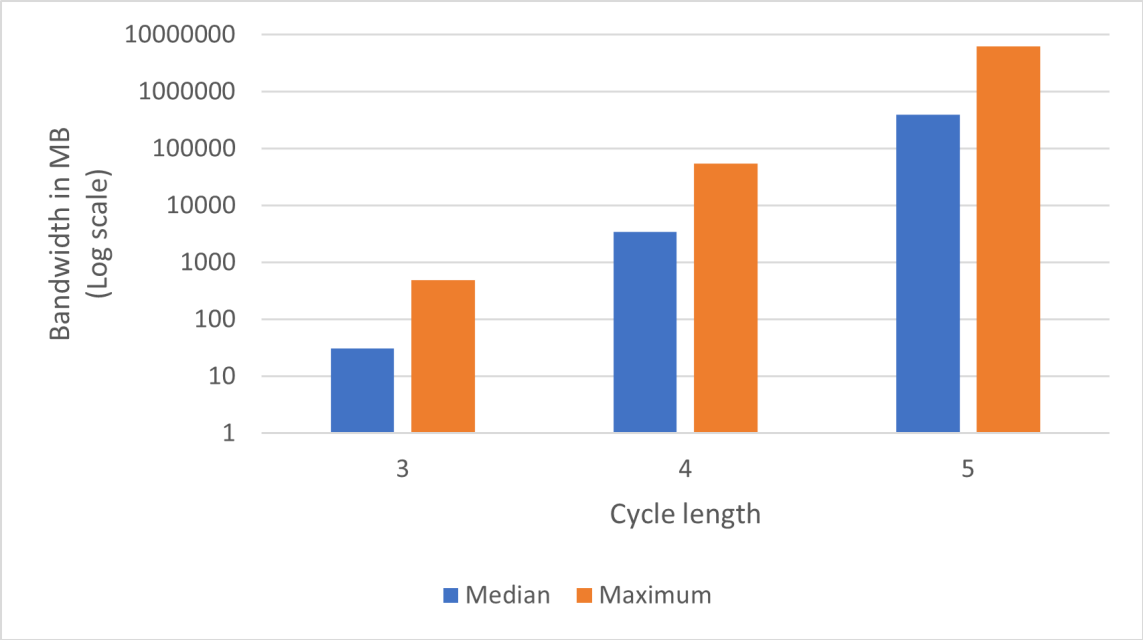


Figure 5.3: Total bandwidth on a graph with 1 million accounts and 100 million edges for differing cycle lengths, averaged over 1000 runs.

graph with 10 thousand accounts and 1 million transactions with their Bellman-Ford and Dijkstra's based protocols on graphs of a similar size, we can see that our highest measured run time of 232 seconds is 7 to 10 times faster than just their precomputation steps on a graph with a similar amount of vertices. While the Dijkstra's based protocol does have a lot more edges, their precomputation step only scales with the number of vertices. In their paper, they state that their Bellman-Ford algorithm can be shortened if the maximum length of the path is known. This would reduce the number of iterations that are necessary for the loop, however, precomputations would still have to be performed. In Table 5.2, we

Table 5.2: Time comparison of our protocol with the shortest path protocols mentioned in [10] on graph sizes of around 10 thousand vertices

Protocol	Vertices	Edges	Time total (s)
Dijkstra's [10]	150	11 thousand	3.0
Bellman-Ford [10]	85	1200	8.1
Our protocol	100	10 thousand	4

compare our results on our smallest graph of 100 vertices and 10 thousand transactions to their results on similar-sized graphs however we see that while their Bellman-Ford protocol

is 2 times slower than our slowest execution of 4 seconds, their Dijkstra's based protocol performs significantly better taking 3 seconds on a slightly bigger graph than ours.

In Table 5.3, where we compare our results to the all paths shortest distance protocols

Table 5.3: Time comparison of our protocol with the all pair shortest path protocols mentioned in [10] on graph sizes of around 10 thousand vertices

Protocol	Vertices	Edges	Time total (s)
Johnson version 1 [10]	100	4950	138.1
Floyd-Warshall [10]	100	4950	6.9
Transitive-Closure [10]	100	4950	51.1
Our protocol	100	10 thousand	166

results from [10] we can see that for small graphs using a privacy-preserving all paths shortest distance protocol can make sense. Finding all the cycles that each of our accounts is a part of took us a total of 166 seconds. Compared to their execution times on similar-sized graphs of 138 seconds using their Johnson version 1 protocol to only 6.9 seconds using their Floyd-Warshall based protocol. Comparing our protocol for different cycle lengths with the results from [10] we can see that while we are faster than any of their privacy-preserving shortest distance protocols with $length = 5$, if we do the same experiments with $length = 6$ we are likely to be slower on small and medium graphs. On larger graphs we believe that we would still be faster but as the biggest graph in [10] has 15 thousand vertices we are not able to do such comparisons. However, since their protocols always take the same amount of time, there is always a maximum cycle length for which their privacy-preserving shortest-distance protocols perform better. Making each viable in specific situations.

In Table 5.4, we compare our bandwidth results with the privacy-preserving shortest dis-

Table 5.4: Bandwidth comparison of our protocol with the all pair shortest path protocols mentioned in [10] on graph sizes of around 10 thousand vertices

Protocol	Vertices	Edges	Bandwidth (GB)	Cycle length
Bellman-Ford [1]	3000	50 thousand	133	∞
Our protocol	1 million	100 million	6228	5
Our protocol	1 million	100 million	55	4

tance protocols in [10] it shows that our bandwidth usage is similar to their Bellman-Ford algorithm with our graph being much larger for a cycle length of 4 and only one order of magnitude more for a cycle length of 5 on a graph that has 1 million accounts and 100 million transactions compared to a graph with only 3 thousand vertices and 50 thousand edges. Based on bandwidth figures presented for their Bellman-Ford based protocol we

think it is safe to say that if both were executed on the same graph with 1 million accounts and 100 million transactions then the Bellman-Ford protocol would use multiple orders of magnitude more bandwidth. Bandwidth figures for their Dijkstra's based privacy-preserving shortest distance protocol were not given so we are not able to compare with those.

6

Conclusion

It is clear that money laundering is a problem and that banks play an important role in combatting it. To do this more effectively it is important for banks to work together. The Dutch government is trying to make way for this by allowing banks to share data with each other in their newly proposed laws. The way that this is currently proposed is a great breach of people's personal privacy and introduces the risk of the information being abused.
An alternative approach is to focus on detecting money laundering in a privacy-preserving manner. In this thesis, we focused on detecting short cycles in a transaction graph, a process that requires cooperation among multiple banks as not all transactions are known by a single bank. Our privacy-preserving protocol enables banks to detect such cycles more efficiently than other state-of-the-art privacy-preserving solutions. In this chapter, we discuss our results, the limitations of our design, and potential areas for future work.

6.1. Discussion

Our main research question was:

> "How can we efficiently find short cycles in financial transactions?"

To address this question, we developed a protocol that detects short cycles in financial transaction graphs without revealing any additional information about the graph beyond the cycles that are found. Our testing shows that our protocol allows for the detection of cycles in large graphs even on consumer-level hardware. Whereas previous works were only able to run graph problem protocols on graphs smaller than 20 thousand vertices, our protocol can detect cycles of length less than 6 in graphs with a million nodes, with a median runtime of under 10 minutes. In terms of security, banks in our protocol are unable to learn of any edge or accounts existence that they were not already previously aware

of. Additionally by keeping track of the messages received our protocol gives the banks the possibility to backtrack and find the entire cycle without the need to share additional information. Our goal was to make a protocol that does not scale with the number of vertices in a graph. While this is achieved in theory in practice we show that a larger graph does seem to impact performance, we believe this to be because the larger graphs also have vertices with a larger degree. The average degree of all our datasets is the same but the highest theoretically possible outdegree in the dataset with 100 vertices is 99 while the dataset with 1 million accounts has multiple accounts that have an outdegree higher than 1000.

6.2. Limitations

In this thesis, we work with some assumptions and simplifications which leads us to have some limitations which we discuss here.
Our datasets are synthetic and only have three different types of money laundering patterns added to them. Testing on datasets from actual banks could reveal more hidden patterns and would likely affect our performance in some way. A real dataset would also allow us to better measure the bandwidth that the protocol uses in the real world as transactions from one account of a bank to another account of the same bank do not add any additional bandwidth and all the operations for it can be done locally.
The messages are indistinguishable from one another, but this means that our protocol is only as secure as the number of times that it is invoked at the same time. Since the bandwidth required for each invocation of the protocol is very high running multiple at the same time slows the execution down by a lot making it likely not practical.

6.3. Future work

In this section, we discuss some potential future research based on insights we obtained during our research.

6.3.1. Other patterns and operations

In addition to detecting cycles, banks may also be interested in identifying other patterns in financial transactions, such as identifying whether any individuals within a group of connected accounts have been labeled as fraudulent. To do this effectively, it would be necessary for banks to be able to group accounts that are closely connected and share whether anyone in that group has been labeled as fraudulent, without revealing which specific account has been labeled or for which account the inquiry was being conducted. This approach would minimize the leakage of sensitive information and could potentially aid banks in their risk assessments.

6.3.2. Relations between accounts

Our current system finds cycles in financial transactions, but it does not account for people having multiple accounts. If a person holds multiple accounts then the money can still end up with the same person but in different accounts. In [31], they account for this by adding static edges to indicate relationships between accounts. Future work could look at ways to take these relationships into account in a privacy-preserving way to improve the detection of money laundering patterns.

6.4. Closing remarks

Combating money laundering is essential for disrupting organized crime. However, the Dutch government's proposed solutions to improve its detection involve granting banks broad powers that raise concerns about privacy. In this thesis, we demonstrate that some of the operations that banks may wish to perform on transaction data can be done while significantly reducing the amount of information that needs to be shared. While further work is needed to make such solutions practical, we believe we have presented a solution that moves in the right direction of combating money laundering without compromising privacy.

References

[1] *2018 reform of EU data protection rules*. European Commission. May 25, 2018. URL: https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf.

[2] Google Account. *Manage your Location History*. Accessed: 31-10-2022. URL: https://support.google.com/accounts/answer/3118687?hl=en.

[3] Google Ads. *Track clicks on your website as conversions*. Accessed: 31-10-2022. URL: https://support.google.com/google-ads/answer/6331304?hl=en.

[4] AFM. *Leidraad Wwft en Sanctiewet*. Accessed: 26-09-2022. URL: https://www.afm.nl/~/profmedia/files/wet-regelgeving/beleidsuitingen/leidraden/wwft.ashx.

[5] Adi Akavia, Rio LaVigne, and Tal Moran. "Topology-Hiding Computation on All Graphs". In: *Advances in Cryptology – CRYPTO 2017*. Ed. by Jonathan Katz and Hovav Shacham. Cham: Springer International Publishing, 2017, pp. 447–467. ISBN: 978-3-319-63688-7.

[6] Adi Akavia and Tal Moran. "Topology-Hiding Computation Beyond Logarithmic Diameter". In: *Advances in Cryptology – EUROCRYPT 2017*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Cham: Springer International Publishing, 2017, pp. 609–637. ISBN: 978-3-319-56617-7.

[7] Abdelrahaman Aly. "Network flow problems with secure multiparty computation". PhD thesis. Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2015. URL: https://hdl.handle.net/2078.1/165269.

[8] Abdelrahaman Aly and Sara Cleemput. "An Improved Protocol for Securely Solving the Shortest Path Problem and its Application to Combinatorial Auctions". In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 971.

[9] ABN Amro. *Detecting financial crime and money laundering*. Accessed: 26-09-2022. URL: https://www.abnamro.com/en/about-abn-amro/landing-page/detecting-financial-crime-and-money-laundering.

[10] 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.

[11] Dutch Payments Association. *Facts and figures on the Dutch payment system in 2021*. Accessed: 26-09-2022. URL: https://factsheet.betaalvereniging.nl/en/.

[12] The World Bank. *Gains in Financial Inclusion, Gains for a Sustainable World*. Accessed: 07-12-2022. URL: https://www.worldbank.org/en/news/immersive-story/2018/05/18/gains-in-financial-inclusion-gains-for-a-sustainable-world?cid=ECR_TT_worldbank_EN_EXT.

[13] Richard Bellman. "On a routing problem". In: *Quarterly of Applied Mathematics* 16 (1958), pp. 87–90.

[14] European Commission. *EU context of anti-money laundering and countering the financing of terrorism*. Accessed: 07-12-2022. URL: https://finance.ec.europa.eu/financial-crime/eu-context-anti-money-laundering-and-countering-financing-terrorism_en.

[15] European Commission. *Proposal for a directive of the European Parliament and of the council on the prevention of the use of the financial system for the purpose of money laundering and terrorist financing*. Accessed: 26-09-2022. URL: https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2013:0045:FIN:EN:PDF.

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

[17] FATF. *Money Laundering - Financial Action Task Force*. Accessed: 26-09-2022. URL: https://www.fatf-gafi.org/faq/moneylaundering/.

[18] Lisa Fleischer, Bruce Hendrickson, and Ali Pinar. "On Identifying Strongly Connected Components in Parallel". In: May 2000, pp. 505–511. ISBN: 978-3-540-67442-9. DOI: 10.1007/3-540-45591-4_68.

[19] Robert W. Floyd. "Algorithm 97: Shortest Path". In: *Commun. ACM* 5.6 (June 1962), p. 345. ISSN: 0001-0782. DOI: 10.1145/367766.368168. URL: https://doi.org/10.1145/367766.368168.

[20] Smart Energy gb. *Smart meters explained*. Accessed: 31-10-2022. URL: https://www.smartenergygb.org/about-smart-meters.

[21] Amnesty International. *Dutch childcare benefit scandal an urgent wake-up call to ban racist algorithms*. Accessed: 27-10-2022. URL: https://www.amnesty.org/en/latest/news/2021/10/xenophobic-machines-dutch-child-benefit-scandal/.

[22] Donald B. Johnson. "Efficient Algorithms for Shortest Paths in Sparse Networks". In: *J. ACM* 24.1 (Jan. 1977), pp. 1–13. ISSN: 0004-5411. DOI: 10.1145/321992.321993. URL: https://doi.org/10.1145/321992.321993.

[23] J Madinger. *Money Laundering : A Guide for Criminal Investigators*. Boca Raton Fla, 2011.

[24] E. Makri. *Co6gc: Linear secret sharing schemes - lsss*. Accessed: 07-12-2022. URL: https://www.esat.kuleuven.be/cosic/blog/lsss/.

[25] Tal Moran, Ilan Orlov, and Silas Richelson. *Topology-Hiding Computation*. Cryptology ePrint Archive, Paper 2014/1022. https://eprint.iacr.org/2014/1022. 2014. URL: https://eprint.iacr.org/2014/1022.

[26] Financial Intelligence Unit - Nederland. *Over de FIU*. Accessed: 07-12-2022. URL: https://www.fiu-nederland.nl/nl/over-de-fiu.

[27] FIU - The Netherlands. *2021 annual review of FIU - The Netherlands*. Accessed: 26-09-2022. URL: https://www.fiu-nederland.nl/sites/www.fiu-nederland.nl/files/documenten/7306_fiu_jaaroverzicht_2021-eng-web_v3.pdf.

[28] Alfred Ng. *How China uses facial recognition to control human behavior*. Accessed: 31-10-2022. URL: https://www.cnet.com/news/politics/in-china-facial-recognition-public-shaming-and-control-go-hand-in-hand/.

[29] Authoriteit Persoonsgegevens. *AP adviseert Eerste Kamer: neem WGS niet aan*. Accessed: 26-09-2022. URL: https://autoriteitpersoonsgegevens.nl/nl/nieuws/ap-adviseert-eerste-kamer-neem-wgs-niet-aan.

[30] Authoriteit Persoonsgegevens. *Nieuwe wet opent deur naar ongekende massasurveillance door banken*. Accessed: 27-10-2022. URL: https://www.autoriteitpersoonsgegevens.nl/nl/nieuws/nieuwe-wet-opent-deur-naar-ongekende-massasurveillance-door-banken.

[31] Xiafei Qiu et al. "Real-Time Constrained Cycle Detection in Large Dynamic Graphs". In: *Proc. VLDB Endow.* 11.12 (Aug. 2018), pp. 1876–1888. ISSN: 2150-8097. DOI: 10.14778/3229863.3229874. URL: https://doi.org/10.14778/3229863.3229874.

[32] Michael O. Rabin. *How To Exchange Secrets with Oblivious Transfer*. Harvard University Technical Report 81 talr@watson.ibm.com 12955 received 21 Jun 2005. 2005. URL: http://eprint.iacr.org/2005/187.

[33] Ronald L. Rivest and Michael L. Dertouzos. "On data banks and privacy homomorphisms". In: 1978.

[34] SAS. *What is Transaction Monitoring in AML?* Accessed: 07-12-2022. URL: https://www.sas.com/en_ie/insights/articles/risk-fraud/what-is-transaction-monitoring-in-aml.html.

[35] 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. URL: https://doi.org/10.1145/359168.359176.

[36] Tweede Kamer Der Staten-Generaal. *Wetsvoorstel Wet gegevensverwerking door samenwerkingsverbanden*. Accessed: 26-09-2022. URL: https://www.tweedekamer.nl/kamerstukken/wetsvoorstellen/detail?cfg=wetsvoorsteldetails&qry=wetsvoorstel%3A35447.

[37] European Data Protection Supervisor. *Opinion of the European Data Protection Supervisor*. Accessed: 26-09-2022. URL: https://edps.europa.eu/sites/default/files/publication/13-07-04_money_laundering_en.pdf.

[38] Toyotaro Suzumura and Hiroki Kanezashi. *Anti-Money Laundering Datasets: InPlus-Lab Anti-Money Laundering DataDatasets*. http://github.com/IBM/AMLSim/. 2021.

[39] Robert Tarjan. “Depth-first search and linear graph algorithms”. In: *12th Annual Symposium on Switching and Automata Theory (swat 1971)*. 1971, pp. 114–121. DOI: 10.1109/SWAT.1971.10.

[40] A. C. Yao. “Protocols for secure computations”. In: *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. Los Alamitos, CA, USA: IEEE Computer Society, Nov. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.88. URL: https://doi.ieeecomputersociety.org/10.1109/SFCS.1982.88.