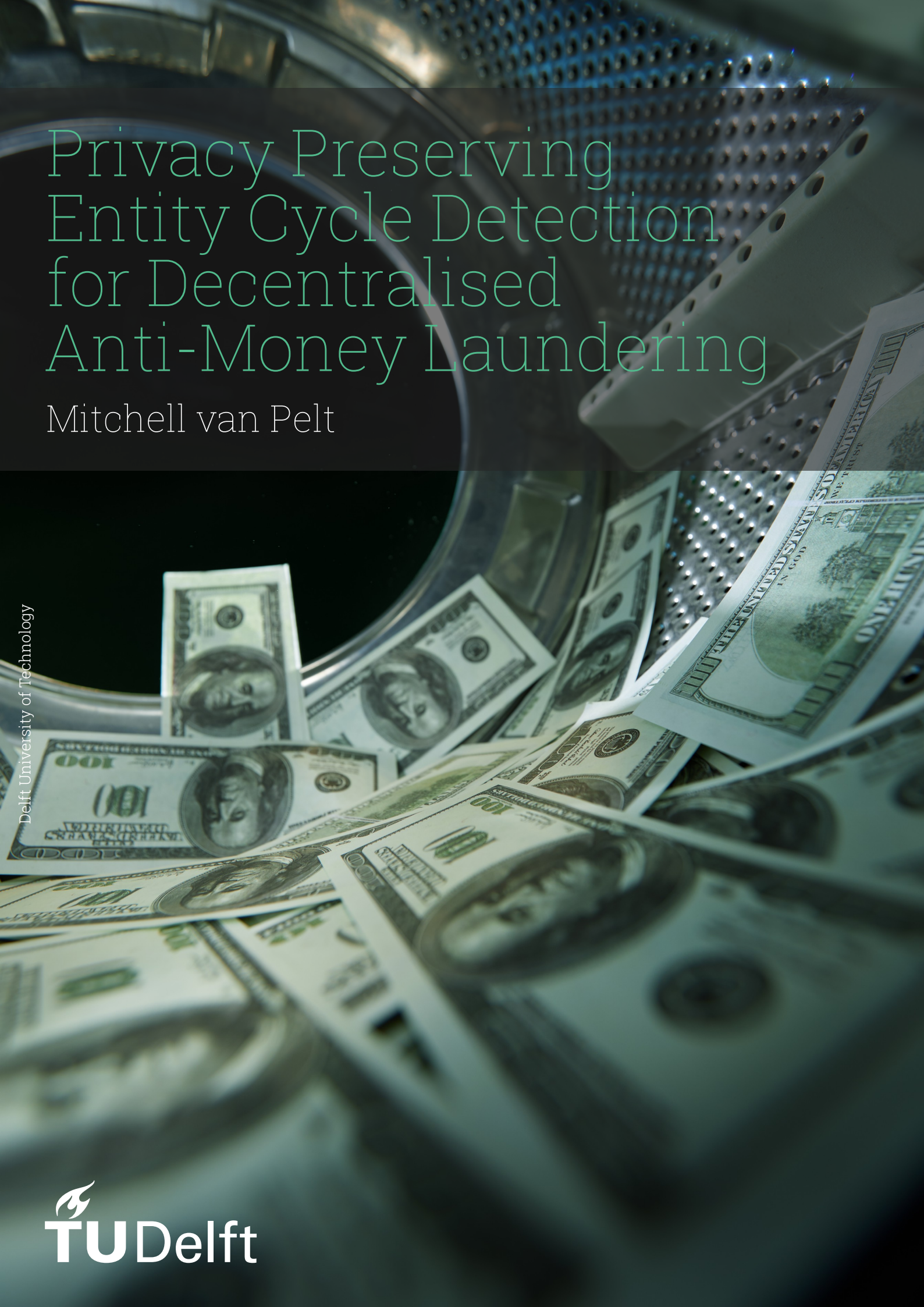
Privacy Preserving Entity Cycle Detection for Decentralised Anti-Money Laundering

Mitchell van Pelt



Delft University of Technology

TU Delft

Privacy Preserving Entity Cycle Detection for Decentralised Anti-Money Laundering

by

Mitchell van Pelt

submitted in partial fulfilment of the requirements to obtain the degree of

Master of Science
in Computer Science

at Delft University of Technology,
to be defended publicly on Tuesday the 30th of June 2026 at 13:45.

Project Duration:	November, 2025 - June, 2026	
Thesis committee:	dr. Z. Erkin	TU Delft, chair
	prof. dr. M.M. de Weerdt	TU Delft

Cover: "Washing banknotes in machine" by iStock/Strixcode
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

No AI tools were used in the creation of this thesis.

TU Delft

Preface

During my six years at TU Delft, I got to experience many of the different domains that computer science has to offer. And, while I'm sure that every one of these has their own value and fascinating use-cases, none have inspired me so much as did the field of cryptography. It amazes me how concepts which for centuries have been classified as 'just' belonging to pure mathematics can now be used for critical tasks in our digital ecosystem, and form a vital pillar of security and trust in the information age.

I'm very happy that I have been able to use these concepts to explore a solution for a kind of problem that nowadays is present in every facet of digital technology. Our society greatly benefits from big data information systems automating our lives and helping us tackle all sorts of problems, but we often forgo or trivialise how the privacy issues they bring affect fairness, equality and personal freedom. I strongly believe that in order to keep our ever more digital society just and fair, we must stop introducing systems that sacrifice privacy to reach some goal. With this thesis, I hope to show that we can create algorithms that use personal data while respecting the persons involved.

I could not have completed this work without the continuous support of those close to me. I would like to thank my family and friends for always being there for me and for supporting me throughout my studies. I would also like to express my gratitude and appreciation to my peers, who were always willing to help me and provide feedback, and who greatly inspired me with their own research and projects. Specifically, I would like to thank Maik de Vries and Miloš Ristić, who, while working on similar problems, helped me gain a better understanding of my own problem and inspired me with new ideas during our discussions. Furthermore, I would very much like to thank my thesis advisor, dr. Zeki Erkin, not only for guiding me throughout the project, but also for helping me become a better academic professional. Finally, I want to express my gratitude to professor De Weerdt for being part of the thesis committee, and for providing me with helpful feedback now and in the past.

Mitchell van Pelt
Delft, June 2026

Abstract

Money laundering is the act of processing criminal proceeds to hide their illicit origin. Banks and financial institutions employ anti-money laundering (AML) techniques to detect and prevent money being laundered. Whereas traditional AML is mainly conducted on the level of individual institutions, modern AML is shifting towards increased collaboration between banks and government agencies. Although greater collaboration is indeed required to combat sophisticated money laundering operations, the lowered legal threshold for sharing customers' financial data raises serious concerns about the privacy of innocents. Privacy preserving anti-money laundering (PPAML) techniques address this problem by allowing banks to collaborate without explicitly sharing sensitive customer data. This work introduces a new PPAML protocol that detects money laundering patterns in a decentralised transaction network. The protocol uses privacy preserving entity resolution to detect money being moved between accounts that are different but owned by the same entity. Such *entity cycles* serve as a strong indicator for money laundering, and their detection can serve as a trigger for banks to further investigate the involved accounts. In the protocol, privacy of the involved entities is preserved by utilising a number of secure multi-party computation techniques. Most notably, partially homomorphic encryption is used to privately compute the distance between two entity identities. Messages in the protocol are unlinkable so that adversaries cannot learn about the topology of the network, making it secure against information leakage in the semi-honest model. Additionally, the protocol is secure against some attacks by covert adversaries. By only attempting to match accounts in the local neighbourhood that transfer money to each other, the protocol achieves runtime and communication complexities that are polynomial on the number of neighbours, being independent of the size of the network. Finally, the protocol is designed to be highly parallelisable, and evaluation shows that this quality leads efficient runtimes.

Contents

List of Illustrations

Figures

Tables

Nomenclature

List of Abbreviations

Abbreviation	Definition
AML	Anti-Money Laundering
AMLA	Authority for Anti-Money Laundering and Countering the Financing of Terrorism
AP	Autoriteit Persoonsgegevens
DH	Diffie-Hellman
ER	Entity Resolution
EU	European Union
FIU	Financial Intelligence Unit.
HE	Homomorphic Encryption
IBAN	International Bank Account Number
ID	Identifier (of a bank account)
IND-CPA	Indistinguishability under Chosen-Plaintext Attack
KYC	Know Your Customer
PET	Privacy Enhancing Technology
PHE	Partially Homomorphic Encryption
PII	Personally Identifiable Information
PPAML	Privacy Preserving Anti-Money Laundering
PPER	Privacy Preserving Entity Resolution
PPRL	Privacy Preserving Record Linkage
SMC	Secure Multi-party Computation
SSN	Social Security Number
STTP	Semi-Trusted Third Party
TTL	Time-To-Live
WGS	Wet Gegevensverwerking door Samenwerkingsverbanden

List of Symbols

Symbol	Definition
$\mathbb{Z}$	Set of all integers
n	Arbitrary modulus
$\mathbb{Z}_n$	Additive group of integers modulo n
$\mathbb{Z}_n^*$	Multiplicative group of integers modulo n
pk	Public key in an asymmetric cryptosystem
sk	Secret key in an asymmetric cryptosystem
m	arbitrary plaintext message
c	Arbitrary ciphertext message
$\otimes$	Arbitrary binary operation
$\bigotimes$	Repeated arbitrary binary operation
$\mathbb{Z}_{n^2}^*$	Multiplicative group of integers modulo n^2
p, q	Arbitrary prime numbers of the same bit-length; factors of n
bl	Length in bits of p and q
g	Generator of a group
λ	Main component of the secret key in the Paillier scheme
μ	Additional component of the secret key in the Paillier scheme
$L(u)$	Function used in the Paillier scheme; $L(u) = (u-1)/n$
r	Random number used in Paillier scheme
β	Blinding factor
$\phi_\beta(x)$	Blinding function applying factor β to value x
BF	Bloom filter
b_i	Bit in a Bloom filter
l	Size of the Bloom filter
k	Number of independent hash functions used to create the Bloom filter
$h(x)$	Cryptographically secure hash function
H	Set of size k of all hash functions used create a Bloom filter
X	Arbitrary set of elements to be encoded as a Bloom filter
$p(FP)$	Probability of a false positive in a Bloom filter
e	Euler's number
v_i, v_j, v_k	Arbitrary vertices in a graph
o_ij	Nonce used in communications between v_i and v_j
$g(x)$	Composite cryptographically secure hash function
a, b, c	Intermediate values in computation of distance between two Bloom filters
Ψ	Set of indices in the plain Bloom filter that are set to 1
Ω	Set of indices in the plain Bloom filter that are set to 0
d_J	Jaccard distance
d_H	Hamming distance
d_E^2	Squared Euclidean distance
v_0	Vertex initiating the protocol instance
ℓ	Maximum length of a to be detected entity cycle
ℓ_{min}	Minimum length of a to be detected entity cycle
ttl	Time-to-live value of a protocol message
pk_{sttp}, sk_{sttp}	Public- and secret keys of the STTP
pk_{v_0}, sk_{v_0}	Public- and secret keys of the initiating vertex
I	The identity of the initiating vertex, i.e. a set of string attributes
g_{v_0}	Generator in pk_{v_0} unique to v_0
μ_{v_0}	Component of sk_{v_0} unique to v_0
$E_{v_0}(x)$	Encryption of value x made with pk_{v_0}
$E_{sttp}(x)$	Encryption of value x made with pk_{sttp}

Symbol	Definition
$N^+(v_j)$	Set of out-neighbours of vertex v_j
id_{v_j}	Global identifier (IBAN) of vertex v_j
D	List of distances between two Bloom filter identities
I_{BF}	Bloom filter identity, i.e. a list of attributes encodes as a Bloom filter
β_j	Blinding factor generated by vertex v_j
ids	List of encrypted identifiers
$blinds$	List of encrypted blinding factors β
γ	Blinding factor generated by v_0
ids_γ	List of encrypted identifiers, inner-blinded by factors γ
d	Distance between a signle pair of Bloom filter attributes
$\mathcal{A}_S$	Semi-honest adversary
$\mathcal{A}_C$	Covert adversary
O_R	Runtime complexity
O_C	Communication complexity
O_S	Storage complexity
K	Arbitrary number of constant operations
V	Number of vertices (bank accounts) in the transaction network
N_{max}	Maximum number of out-neighbours of a vertex in the transaction network
N_{avg}	Average number of out-neighbours of a vertex in the transaction network
t_1, t_2	Matching rule thresholds.
V_{last}	Number of vertices in the last layer of vertices explored by a protocol instance

1

Introduction

Money laundering is the act of concealing the illegal origin of criminally obtained funds [63]. Criminals engage in money laundering in order to introduce their profits into the legal financial system. This allows them to spend their profits without attracting suspicion from authorities. In the traditional model, money laundering consists of three stages: placement, layering and integration. The process involves introducing criminal cash into one or many bank accounts (placement), after which the money is transacted through a series of other bank accounts (layering), only to end up in an account owned by the criminal (integration) [18]. To avoid the money laundering process being detected, criminals use bank accounts registered at different banks, usually spread over multiple countries. For this reason money laundering is almost always a cross-border crime.

On its own, money laundering is a financial crime which occurs in a multitude of contexts. It can be used for all sorts of illegal activities involving large sums of money. White-collar crimes refer to activities such as tax evasion, theft and (online) fraud [41]. Here, money laundering is used to cover up the crime itself and prevent authorities from taking notice of any of the involved parties. However, money laundering does not only occur in crimes where the damage is purely financial. In fact, money laundering is a central component of organised crime. Crime syndicates involved in all sorts of serious criminal acts, such as drug trade, human trafficking and smuggling, rely on sophisticated money laundering processes to clean their dirty profits and spend their money in the legal economy [59]. Furthermore, money laundering is also closely related to terrorist financing. Here, the only difference is that in terrorist financing the origin of the money can also be a legal source [59].

This broad presence of money laundering in many areas of crime highlights the need for governments and inter-governmental organisations to set up laws and regulations to prevent money being laundered. Such regulations have been around for decades, being internationally standardised by a United Nations convention in 1988 [63]. However, such regulations need to constantly adapt to criminals changing their money laundering practices [41].

In the European Union, organised crime has become a more serious threat to society in recent years, with Europol having identified 821 distinct criminal networks it labels as most threatening [22]. The agency states that these networks are destabilising European society, not just by their criminal acts, but also by their massive money laundering operations, which create a hidden financial system that weakens economies and erodes trust in governance structures [23]. For this reason, the European Commission has made countering money laundering one of their four key actions to tackle organised crime [21].

The EU's more serious attitude towards combatting money laundering has thus far culminated in an extensive package of new money laundering rules being adopted in 2024 [17]. The package aims to harmonise the anti-money laundering (AML) regulations across member states, move towards a more centralised approach in coordinating AML operations, and lower the legal threshold for sharing financial data. Moreover, the EU plans have also prompted member states to introduce new AML legislation that further centralises national AML systems and intensifies financial monitoring of citizens [54] [7] [6].

1.1. Anti-Money Laundering

Anti-money laundering, often shortened to AML, is an umbrella term that refers to procedures aimed at preventing money laundering and related financial crimes [25]. AML regulations require financial institutions, notably banks and credit card companies, to monitor their customers and their transactions, and report suspicious activity to authorities.

One key element of AML is a set of rules called 'Know Your Customer', or KYC, which requires financial institutions to have a clear record of who their customer is [18]. In a KYC procedure, the financial institution tries to establish the customer's identity by collecting personally identifiable information (PII) from them. That is, information that can be directly linked to one person and can be used to clearly distinguish one person from another, such as full name, social security number and date of birth [46]. This is usually done by asking the customer to provide official identification documents, such as an ID card. It is also important that the institution checks the validity of provided documents. Next to establishing the identity, KYC requires institutions to do a risk assessment of the customer being involved in money laundering, terrorist financing or other financial crimes. Such an assessment may involve requesting additional information from the customer.

Another important task in AML is transaction monitoring. This involves the financial institution analysing the transactions of their customers and flagging individual transactions or larger patterns that are an indication of money laundering. One type of this scrutinising involves checking if the transactions are consistent with the institution's knowledge of the customer [18]. For example, if someone working as a small shopkeeper starts receiving tens of thousands of euros, something must be wrong. The second type of transaction monitoring is anomaly detection [20]. Here, the transactions between many accounts are modelled as a graph, which is then analysed to look for patterns that are known to indicate money laundering. Such patterns include small cycles, cliques and tripartite subgraphs.

While individual financial institutions monitor their customers for suspicious activity, it is in many cases difficult for them to conclude if someone is actually engaging in money laundering. This is because such an institution does not have access to the complete financial and personal situation of its customers. Instead of requiring institutions to launch a further investigation themselves, AML regulations require organisations to submit their initial findings to a Financial Intelligence Unit (FIU) [26]. This is a national organisation that sits between financial institutions and law enforcement agencies. The role of an FIU is to further analyse reports they receive from the private sector, obtain additional information when needed, and compile conclusive evidence of financial crime [27]. FIUs are able to combine reports from financial institutions with data from other organisations and government registries to turn reports of suspicions into concrete intelligence and evidence. Only when an FIU is convinced a crime is being committed will it inform law enforcement, which will start formal prosecution.

Since the aim of criminals is to avoid detection, they will often spread their money laundering operation over multiple institutions, making it difficult for one institution to detect suspicious activity. It is consequently widely advocated that modern AML requires close cooperation and the extensive sharing of information between organisations [29] [8] [55]. The system of FIUs already helps to integrate information across organisations within a country's borders and make AML more effective on a national level. However, the cross-border nature of modern money laundering makes that even with national FIUs in place, many money laundering activities still remain hidden because the complete operation cannot be seen from the perspective of a single nation [8]. It is exactly for this reason that the new EU plan focusses on centralising the AML infrastructure even further and making sharing financial information easier [17].

The most significant plan in the package is the establishment of an EU-wide agency called the Authority for Anti-Money Laundering and Countering the Financing of Terrorism (AMLA) [4]. This authority will take on the role of supervising and coordinating the national FIUs, as well as directly supervising some of the larger private financial institutions. Additionally, the EU wishes to centralise and expand access to financial data: the directive provisions the creation of a so called 'single access point' for financial data, which contains information on bank accounts of all EU citizens [17].

While this approach of increased personal and financial data integration would help improve the effectiveness of anti-money laundering, it also raises deep concerns over the privacy of innocent citizens.

1.2. Privacy Issues with Sharing Financial Data

Financial data is information about a person's or organisation's financial activities. This mainly includes information on financial transactions, i.e. what someone spends money on, from whom money is received, and the amounts. Additionally, it includes records of a person's larger financial assets, such as total income, loans, mortgages and investments. Since financial data is directly related to one person or organisation, it often includes personal identifiable information (PII) or is easily linkable to PII via identifiers such as bank account number (IBAN) or credit card info [50].

Furthermore, from the transaction data, a wide array of other details about one's personal life can be deduced. In the first place, purchases and expenditures directly expose personal preferences, interests and activities. This reveals much about a person's private life: certain stores visited reveal a person's hobbies, and donations to certain organisations can expose one's political or religious beliefs. The knowledge of where money is spent and how much can even be used to deduce what products are bought. This in turn uncovers even more. For example, knowing what groceries someone buys reveals if they have children, and what medications a person buys reveals their medical conditions.

Secondly, in our financially interconnected society, transactions to and from other people can be used to reconstruct a large part of one's social network. For example, frequent transactions between people of varying amount in the range of say fifty to a hundred euros highly suggest two people who buy groceries for each other, indicating that these people are likely living together. In cultures where it is common for people to share bills and other expenditures, such as the Netherlands, someone is likely to have many transactions to and from people they have socially interacted with. The transactions reveal these interactions and the frequency can even reveal the degree of social closeness.

Finally, payments at physical locations reveal where money is spent and at what time. This reveals where someone has been and when they have been there. Hence, transaction data can be used to reconstruct someone's exact movements and allow for physical tracking of a person.

All these points considered, financial transaction data contains extremely detailed and deeply sensitive information about a person's personal life, social connections, and physical movements. Given that the Cambridge Dictionary defines privacy as "someone's right to keep their personal matters and relationships secret", it is indisputable to conclude that the sharing of a person's financial data without their knowledge or consent is a clear violation of their privacy [15].

Given this, it is important to consider the impact of rules and regulations governing the exchange of financial data on the privacy of citizens. Traditionally, people only share details about their financial activities with their bank. Banks have always realised that safeguarding the confidentiality of financial information is a requirement for maintaining the confidence of their customers. This concept is historically known as banking secrecy [62]. In contrast, AML regulations force banks to share customer financial data with third parties, breaching banking secrecy. For a long time, the system where financial data can only be shared with trusted national FIUs has struck an acceptable balance between respecting privacy and preventing money laundering. However, the recent trend in new regulations aiming to strengthen AML effectiveness further erodes the already fragile financial privacy of innocent citizens.

For instance, the establishment of the EU's Single Access Point for financial data poses a significant threat to citizens privacy [17]. This is not because all European FIUs can access it, but because this new mechanism also explicitly allows law enforcement agencies to access the data. This is a big escalation of the sharing of financial data because it allows such agencies to circumvent the FIU and access financial records directly. This makes it possible for a person's financial data to end up in the hands of law enforcement agencies even when no evidence of financial crime has been found.

Furthermore, following the EU's example of lowering the threshold for data sharing, member states are introducing new legislation that take away more financial privacy safeguards. In the Netherlands, parliament recently adopted the 'Wet gegevensverwerking door samenwerkingsverbanden' (WGS) [43]. The WGS creates a legal framework for public and private organisations to freely share personal and financial data with each other when one of the organisations has a 'suspicion' that some crime has been committed [7]. The law has been fiercely criticised by the national data protection authority AP, stating that both the vast scale of the data exchanged and the shallow threshold for commencing the exchange opens the door for mass surveillance by both government agencies and private organisations [5].

Finally, not even a year after the adoption of the WGS, the Dutch government announced that in order to make it easier for banks to comply with the new European AML rules, new legislation is being prepared to allow banks to directly access the 'Basisregistratie Personen', the national database of citizens' personal data [54]. This is another significant escalation that further diminishes the privacy of innocent people by sharing the most sensitive of data with the private sector [14].

These are just three examples of how the increased threat of organised crime is pushing governments towards sacrificing their citizens' privacy. While collaboration in AML is necessary to fight this threat, the ongoing deprecation of privacy laws is an unacceptable violation of the rights of innocents. Therefore, a new approach is needed where the benefits of collaborative AML do not have to come at the cost of reduced privacy.

1.3. Privacy Preserving Anti-Money Laundering

Privacy preserving anti-money laundering (PPAML) can be considered a subset of the field of private financial crime detection, which itself is a domain specific application of privacy enhancing technologies (PETs). The goal PPAML is to enable collaborative AML without involved organisations having to sacrifice the privacy of the people involved. That is, PPAML techniques make it possible to detect money laundering effectively using the information from multiple parties, without the parties having to explicitly share any actual customer- and transaction data with the others. Such techniques show that privacy and effective AML are far from incompatible, and can both be achieved without sacrificing one or the other.

One of the first PPAML applications is the Ma^{3}tch technology developed by Kroon (2013) [39]. Ma^{3}tch is a framework that allows for decentralised data integration and analysis. It was specifically developed for financial intelligence units (FIUs) in the European Union, such that these units can analyse their combined data without it having to leave their own premises. The framework uses anonymisation techniques and distributed agent computation to run analysis tasks on data from different decentralised sources. Ma^{3}tch is the underlying framework for the EU's FIU.NET system: a decentralised computer system that allows FIUs in the EU to work together on AML and related financial crime detection.

Furthermore, much research in PPAML is aimed at finding privacy preserving technologies for transaction monitoring. This being one of the two core tasks of AML, finding methods to do it collaboratively while preserving privacy would allow financial institutions to greatly improve their money laundering detection capabilities. The reason for this is that in the traditional setup, each bank can only monitor the transactions for the bank accounts it operates, allowing criminals to hide their activities by laundering money through multiple banks. If banks can collaborate, they can collectively monitor all transactions and detect suspicious patterns that would otherwise remain hidden. In collaborative transaction monitoring methods, banks detect features and patterns in a decentralised transaction graph. Each bank has its own partition of this graph, representing the transactions it knows about. Sangers et al. (2019) [56] use secure multi party computation techniques to run a private decentralised PageRank algorithm on such a graph. In a related work, Van Egmond et al. (2025) [64] use homomorphic encryption for the propagation of account risk scores over the graph.

Porsius Martins (2023) [49] introduced a protocol for the private detection of short cycles in the decentralised transaction graph. Jense (2024) [34] refined this method. The private detection of short cycles in transactions between multiple banks is significant, because the presence of such a pattern serves as a great indicator for money laundering detection. A cycle in a transaction graph represents an entity transferring money to other accounts, but the money eventually ending up in the same account as it started, which is exactly what also happens in money laundering. In a large financial ecosystem, this can happen by chance, but the number of transactions in the cycle being low (3-6) is extremely rare, it being almost always the result of money laundering [32].

1.4. Research Objective

While the private small cycle detection protocols are a great step in PPAML, they only cover a very basic case. The current state-of-the-art protocols only detect cycles between bank accounts. This detection method can be easily bypassed by criminals by simply using two different accounts for the source of the money and its destination. In such a case, no cycle between bank accounts exists.

However, when the source and destination account are both owned by the criminal, another cycle still exists. That is, a cycle between entities. In a single bank setup, such a cycle would be easily detectable because the bank would link two accounts with the same owner. However, linking accounts operated by different banks requires exchanging information about account ownership between banks. This exposes the banks' customers and severely compromises their privacy. To the best of the author's knowledge, no solution currently exists that achieves detecting such entity cycles while also preserving privacy.

Therefore, the goal of this thesis is to investigate this research gap and find a solution to privately detect entity cycle patterns. This prompts the following research question:

How can money laundering cycles in transactions be detected privately when criminals use multiple bank accounts at different banks?

To briefly elaborate, the research objective is to first investigate current privacy preserving methods related to anti-money laundering, specifically those covering private transaction pattern detection, and techniques for the private linking of entity data. These techniques can then be used to construct a new method that covers the detection of the more complex money laundering pattern where a criminal moves money from one of his accounts to another.

1.5. Contributions

The research question is answered by the introduction of a novel protocol that allows banks to privately link the owners of accounts that have a layer of transactions between them. The protocol is able to uncover that money is being laundered by detecting small cycles in transactions between entities. Furthermore, it is demonstrated that the protocol has adequate security and that the design allows for practical runtime performance. To be more precise, this thesis contributes the following:

1. A review of the current state-of-the-art PPAML methods for private cycle detection. The current techniques are analysed, and it is explained why they only cover a very basic case. It is further motivated why this basic approach is likely not effective to improve AML in practice, and what other solution is required to cover a more complex and realistic money laundering scenario.
2. The introduction of a new privacy-preserving anti-money laundering protocol. The protocol operates in a decentralised transaction network, meaning bank accounts, although operated by a bank, are the primary agents that interact. The accounts privately exchange messages that propagate to all other accounts in the local neighbourhood. By including a secure representation of the account owner's identity in the message, the similarity between account owners is computed using a privacy preserving entity resolution technique. To ensure no account (or bank) learns the anything about the identity of others, a semi-trusted third party is employed to match accounts based on their similarity. When a match is found, the matched account, and all accounts in the path of the laundering, are immediately revealed, eliminating the need for any extra messages.
3. An analysis of both the security and complexity of the protocol. It is shown that the protocol is designed in such a way that prevents revealing any information about the transaction network to parties which they do not already know. Similarly, it is demonstrated that the protocol is secure against possible identity-related attacks in the covert security model. Furthermore, the theoretical runtime, communication and storage complexities are deduced and analysed.
4. An evaluation of the practical performance of the protocol. The protocol is implemented and run on a synthetic transaction network to show its performance and practical efficiency. It is demonstrated that the protocol is highly parallelisable, indicating it is suitable for operation in a distributed setting.

1.6. Outline

The remainder of this report is structured is follows. Chapter 2 contains an explanation of technical prerequisites that play an important role throughout the report. Then, in Chapter 3, further background and related works are explained and analysed. The protocol itself is introduced and explained in Chapter 4. Next, in Chapter 5, the analyses of the security and complexity of the protocol are presented. Chapter 6 contains the evaluation of the protocol. Finally, Chapter 7 includes a discussion of the research, and a conclusion in given in Chapter 8.

2

Preliminaries

The section introduces the preliminary concepts which are used throughout the report. Firstly, in Section 2.1 more is explained about bank transactions and the network that they collectively form. Then, Sections 2.2, 2.3, 2.4 and 2.5 explain the most important cryptographic background. Section 2.6 explains the Bloom filter data structure, and Section 2.7 gives a brief introduction on security models.

2.1. Transaction networks

A transaction network, also called a money transfer network, is the structure formed by monetary transactions between bank accounts [32]. Such a transaction has an origin (the account sending the money) and a destination (the account receiving the money). In transaction data, the accounts are usually identified by their IBAN number. A transaction has different attributes associated with it, most notably the amount of money transferred and a timestamp. A transaction network can be modelled as a graph where the vertices are the bank accounts and the edges are the transactions. Such a transaction graph is directed, with the edges following the flow of the money, and can also be weighted with the amount of money on the edges. Furthermore, transaction graphs are usually scale-free networks, meaning that they include some vertices that are hubs or hot-points [42]. Table 2.1 shows an example of some transaction data, which is modelled as a transaction graph in Figure 2.1.

id	Source account	Destination account	Timestamp	Amount
1	A	B	799878997	42.00
2	A	F	910848019	3.14
3	A	G	992343299	14.41
4	B	E	993656399	2718.28
5	B	F	994707499	161.80
6	C	E	995111599	127.00
7	D	A	999727999	23.14

Table 2.1: Example of bank transaction data.

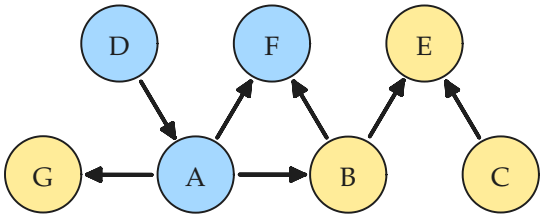


Figure 2.1: Basic transaction network based on the data in Table 2.1. Blue vertices are accounts operated by bank α, and yellow vertices are accounts operated by bank β.

Banks manage the transactions where one of their customers' accounts is the source or destination of the money. This means that banks have the transaction data of only accounts they manage [64]. Because banking secrecy prevents banks for sharing such data, no bank or other centralised party has access to all transaction data in the world. The result is that no one party knows the entire transaction network: the network is partitioned. Banks only have a partition of the entire transaction graph [13]. Figure 2.2 shows the graph partition of the transaction data from Table 2.1 that bank α can see.

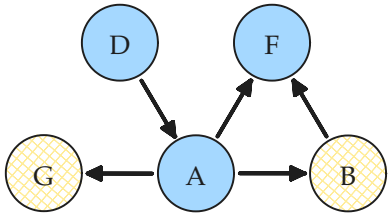


Figure 2.2: Partition of the complete transaction network as seen from the perspective of bank α. The bank can see all accounts it operates (A,D,F) and all transactions involving these accounts. Bank α also knows some accounts operated by other banks exist (B,G), but only if they are involved in transactions with its own accounts.

Such transaction graph partitions can be disconnected and can include many small components. Furthermore, in such a partition there are two types of vertices: accounts operated by the bank 'owning' the partition and account that are operated by other banks. In order to create a simple representation of the partitioned transaction network, it can be modelled as a fully decentralised graph [34]. In such a decentralised transaction graph, each account or vertex is considered to be an individual party. Each account only knows of the transactions from or to itself. It knows who its neighbours are, but knows nothing of other transactions of its neighbours, or, more general, knows nothing about the topology of the graph other than its adjacent edges. Figure 2.3 shows the perspective of account B in a decentralised transaction graph based on Table 2.1.

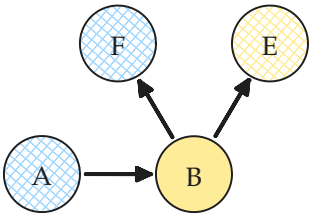


Figure 2.3: Fully decentralised view of the transaction network from the perspective of account B. Note that, in this model, account B has no knowledge of accounts C or G, even though they are operated by the same bank.

2.2. Modular Arithmetic

Public-key cryptosystems are cryptographic schemes that use two keys: a public key and a secret key [37]. The public key can be shared with everyone without compromising the scheme's security, while the secret key is kept private by the party that owns it. Practical public-key systems are usually implemented using modular arithmetic. Modular arithmetic is a system of mathematical operations using only integers. Rather than using all the integers in $\mathbb{Z}$, in modular arithmetic one uses only the integers from 0 up to a given number n. If an operation results in a number equal to or greater than n, the numbers 'wrap around'. Specifically, the number must always be a remainder of a division by n. This is denoted using the modulo operator 'mod'. That is, $y = x \mod n$ indicates that y is the remainder of x when divided by n.

In modular arithmetic, one formally works only with numbers in a given group. A group is a set (of integers) combined with an operation (e.g. addition, multiplication) with the following properties: closure, existence of an identity, existence of inverses and associativity. By applying the operation 'mod' to every statement, the result of an operation is always in the additive modulo group of n, which is denoted as $\mathbb{Z}_n = \{0, 1, ..., n-1\}$. Another group often used for public-key cryptosystems is the multiplicative modulo group of n, denoted $\mathbb{Z}_n^*$. This group is a subset of $\mathbb{Z}_n$ where all the numbers that do not have a multiplicative inverse in the group are removed. Formally, this is expressed as

$$\mathbb{Z}_n^* = \{x \in \{1, ..., n-1\} \mid \gcd(x, n) = 1\} \tag{2.1}$$

Here, $\gcd(a, b)$ is the greatest common divisor of a and b, which is the largest positive integer that divides both a and b. An operation related to this is the least common multiple $\text{lcm}(a, b)$, which is the smallest positive integer divisible by both a and b. Finally, some groups are cyclic, which means that at least one of the elements is a generator. One can obtain every element in a cyclic group by repeating an operation on a generator, e.g. by exponentiating it.

2.3. Partially Homomorphic Encryption

An encryption scheme is a cryptographic system that transforms plaintext messages into ciphertexts (also called 'encryptions') [37]. When a message is encrypted, its contents are hidden. This provides the cryptographic property of confidentiality. In an encryption scheme, the original message can be recovered from the ciphertext by decrypting it with the secret key.

Some encryption schemes posses a property called homomorphism. In a homomorphic encryption (HE) scheme, the ciphertexts are malleable. This means that when a plaintext is encrypted, its structure is preserved in the ciphertext in such a way that a certain operation on the ciphertext will change the underlying plaintext in a predictable way. This allows a party who only has access to the ciphertexts (and usually also the public encryption key) to change the messages by combining or operating on the ciphertexts.

Partially homomorphic encryption (PHE) is a type of HE where only one kind of operation on the ciphertext is possible [37]. There is no limit for how many times the operation can be applied for PHE systems. The operation that is applied to the ciphertext does not have to be the same operation that consequently gets applied to the plaintext. For example, depending on the encryption scheme, multiplying two ciphertexts can result in either the multiplication or the addition of the two corresponding plaintexts. The two most common variants of PHE are multiplicative homomorphism and additive homomorphism. A cryptosystem is multiplicatively homomorphic if the plaintexts can be multiplied by applying some operation ($\otimes$) to the ciphertexts [38]:

$$\text{Enc}_{pk}(m_1 \cdot m_2) = \text{Enc}_{pk}(m_1) \otimes \text{Enc}_{pk}(m_2) \tag{2.2}$$

A cryptosystem is additively homomorphic if the plaintexts can be added by applying some operation ($\otimes$) to the ciphertexts:

$$\text{Enc}_{pk}(m_1 + m_2) = \text{Enc}_{pk}(m_1) \otimes \text{Enc}_{pk}(m_2) \tag{2.3}$$

Furthermore, in a multiplicatively homomorphic system, exponentiation by a plaintext constant k is possible by repeating the operation ($\otimes$) on the same ciphertext k times to do repeated multiplication of the plaintext.

$$\text{Enc}_{pk}(m^k) = \text{Enc}_{pk}(\prod_{i=1}^{k} m) = \bigotimes_{i=1}^{k} \text{Enc}_{pk}(m) \tag{2.4}$$

Similarly, in an additively homomorphic scheme, multiplication by a plaintext constant k is possible by repeating the operation ($\otimes$) on the same ciphertext k times to do repeated addition of the plaintext.

$$\text{Enc}_{pk}(k \cdot m) = \text{Enc}_{pk}(\sum_{i=1}^{k} m) = \bigotimes_{i=1}^{k} \text{Enc}_{pk}(m) \tag{2.5}$$

2.4. Paillier Cryptosystem

A prominent public-key encryption scheme is the system developed by Pascal Paillier (1999) [47]. The scheme encrypts messages from $\mathbb{Z}_n$ to a multiplicative quadratic ciphertext space $\mathbb{Z}^*_{n^2}$. A keypair is obtained by first taking two uniformly random prime numbers p and q of a given secure bit-length (bl). Currently, the minimal secure length is 1024 bits [9]. The modulus n is then computed as the product of p and q. Then, a uniformly random generator g needs to be selected from $\mathbb{Z}^*_{n^2}$, with the condition that n divides the order of g. A basic case that always satisfies this condition is $g = n + 1$. Combined, $\langle n, g\rangle$ form the public key. The secret key is found by computing $\lambda = \text{lcm}(p-1, q-1)$. While not specified by Paillier himself, a common shortcut for his scheme is to pre-compute $\mu = (L(g^\lambda \mod n^2))^{-1} \mod n$ [61] [67]. The secret key then becomes $\langle \lambda, \mu\rangle$. $L(u)$ is a function defined by Paillier which should always return an integer in correct operation of the system:

$$L(u) = \frac{u-1}{n} \tag{2.6}$$

Formally, the key generation process is defined as [48]:

$$p, q \xleftarrow{R} \{x \in \{0,1\}^{bl} \mid x \text{ is prime}\} \tag{2.7}$$

$$n = p \cdot q \tag{2.8}$$

$$\lambda = \text{lcm}(p-1, q-1) \tag{2.9}$$

$$g \xleftarrow{R} \{x \in \mathbb{Z}^*_{n^2} \mid \gcd(L(x^\lambda \bmod n^2), n) = 1\} \tag{2.10}$$

$$\mu = (L(g^\lambda \bmod n^2))^{-1} \mod n \tag{2.11}$$

$$\langle pk, sk\rangle = \langle\langle n, g\rangle, \langle \lambda, \mu\rangle\rangle \tag{2.12}$$

Any plaintext can be encrypted as long as it is an element of $\mathbb{Z}_n$. The Paillier scheme is non-deterministic and incorporates a random number r in the encryption which must be taken from $\mathbb{Z}^*_n$. The plaintext message is mapped from plaintext space $\mathbb{Z}_n$ to ciphertext space $\mathbb{Z}^*_{n^2}$, meaning that there are significantly more ciphertexts than plaintexts. Both encryption and decryption require exponentiations in $\mathbb{Z}^*_{n^2}$, making the scheme not as efficient as other public-key schemes. Formally, Paillier encryption of message m is defined as follows:

$$m \in \mathbb{Z}_n \tag{2.13}$$

$$r \xleftarrow{R} \mathbb{Z}^*_n \tag{2.14}$$

$$c = \text{Enc}_{pk}(m) = g^m \cdot r^n \mod n^2 \tag{2.15}$$

Decryption of ciphertext c is defined as:

$$c \in \mathbb{Z}^*_{n^2} \tag{2.16}$$

$$m = \text{Dec}_{\langle pk, sk\rangle}(c) = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} = \frac{L(c^\lambda \bmod n^2)}{\mu} \mod n \tag{2.17}$$

Additionally, an important characteristic of the Pallier encryption scheme is that it is additively homomorphic [47]. Specifically, plaintexts can be added by multiplying the ciphertexts:

$$\text{Enc}_{pk}(m_1 + m_2) = \text{Enc}_{pk}(m_1) \cdot \text{Enc}_{pk}(m_2) \mod n^2 \tag{2.18}$$

$$= (g^{m_1} \cdot r_1^n) \cdot (g^{m_2} \cdot r_2^n) = g^{m_1+m_2} \cdot (r_1 \cdot r_2)^n \mod n^2 \tag{2.19}$$

A useful consequence of this quality is that the randomness on a ciphertext can be refreshed. That is, to introduce a new random value which changes the ciphertext, but not the underlying plaintext. In the Paillier scheme, refreshing randomness is the same as homomorphically adding zero:

$$\text{Enc}_{pk}(m)' = \text{Enc}_{pk}(m) \cdot \text{Enc}_{pk}(0) = (g^m \cdot r_1^n) \cdot (g^0 \cdot r_2^n) \mod n^2 \tag{2.20}$$

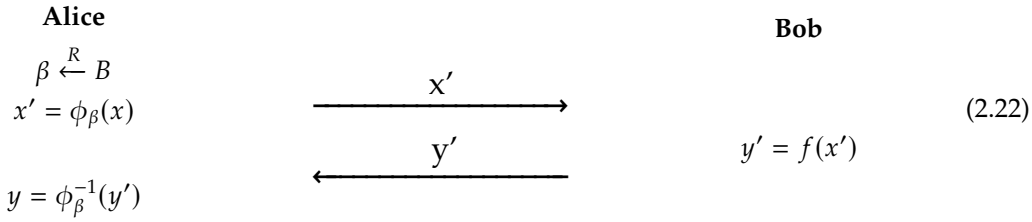
$$= (g^m \cdot r_1^n) \cdot r_2^n = g^m \cdot (r_1 \cdot r_2)^n = g^m \cdot (r')^n \mod n^2 \tag{2.21}$$

Note that in order to refresh the randomness, one does not actually require g, only n.

2.5. Blinding

Blinding is a secure multi-party computation technique that allows for the outsourcing of a computation to another party, while keeping the input and output of the computation private [1]. It is used in settings where one party needs the result of a computation, but cannot do the computation itself, either because the party does not have the computational resources or because for other reasons the party cannot perform the operation, e.g. a decryption with a secret key it does not have. Blinding works by applying a blinding factor to the input which cannot be removed by the other party and is carried over in a structured way to the output. The other party can apply a function to the blinded input, obtaining a blinded output. The other party should not be able to learn the actual input and output from these blinded values. The blinding should be done in such a way that the original output can be obtained when one knows the blinded output and the blinding factor.

Specifically, when Alice has input x and requires y, and needs Bob to apply a function $f(x) = y$, Alice can use a blinding function $\phi_\beta(x)$, which blinds the input with a blinding factor β. This blinding factor needs to be picked from the set of possible blinding factors B. She can then send this blinded input to Bob who applies the function and obtains the blinded output $\phi_\beta(y)$, which he sends back to Alice. Finally, Alice removes the blinding factor from the blinded output by applying the inverse blinding function ϕ_β^{-1}. Formally, parties compute the following:



(2.22)

Often, the blinding function can be a simple mathematical operation, such as addition of a factor, i.e. $\phi_\beta(x) = x + \beta$ and $\phi_\beta^{-1}(y') = y' - \beta$, or multiplication of a factor, i.e. $\phi_\beta(x) = x \cdot \beta$ and $\phi_\beta^{-1}(y') = y' \cdot \beta^{-1}$.

2.6. Bloom Filters

A Bloom filter is a probabilistic data structure that is used as a space-efficient encoding for sets [45]. It is named for Burton Bloom who introduced the concept in 1970 [12]. It represents a set as a vector of bits (filter) of length l, where the presence of elements causes certain bits to be toggled in the filter. Which bits are toggled is determined by hashing each element in the set with multiple (k) independent hash functions H. Hashing an element with a hash function results in an index of the filter that will be toggled. Formally, a Bloom filter BF can be described as:

$$BF = \langle b_1, b_2, ..., b_l \rangle \tag{2.23}$$

$$\forall b_i \in BF : (\exists h \in H \land \exists x \in X : h(x) = i) \rightarrow b_i = 1 \tag{2.24}$$

That is, a bit in the Bloom filter is only set to 1 if at least one of the k hash functions H outputs the bit's index for at least one element in the set X. The Bloom filter is probabilistic because it can contain false positives: a number of bits being set to 1 does not necessarily indicate the presence of an element in the set which hashes to exactly those positions. These bits can be set by the presence of other elements that each hash to one or more of the positions. Conversely, there can be no false negatives in a Bloom filter: An element being present always causes the corresponding bits to be set, so if one of these bits is not set, it can only mean that the element is not present. The probability of a false positive depends on the length of the filter l, the number of hash functions k and the number of elements in the set $|X|$ [58]:

$$p(FP) = (1 - e^{-k|X|/l})^k \tag{2.25}$$

Bloom filters can be used to approximate the similarity between two sets: if the sets have many elements in common, then the Bloom filters will have great overlap in the bit positions that are set to one. Conversely, if the Bloom filters have high overlap, the sets are likely similar. Therefore, a Bloom filter can be considered a distance-preserving encoding of a set [58].

2.7. Security Models

The security model (or adversary model) of a multi-party computation protocol refers to the worst-case type of adversary that can be present for the protocol to be secure [33]. More simply, it refers to how much each involved party trusts the others. In a completely honest security model, all the parties would trust each other completely and no adversaries are present among the parties. In such a case, one can use a protocol without any security, and it is therefore rarely used in the context of secure multi-party computation and privacy enhancing technologies. In cases where the parties cannot be completely trusted, there are two main security models defined.

The first is the semi-honest security model. In such a setting, it is assumed that the other parties all follow the protocol correctly, but may attempt to learn additional information whenever possible. This means that they will analyse all data they receive, and try to learn more information than just the inputs and outputs of the protocol. For example, by performing attacks on the data they received by following the protocol. A protocol is secure in the semi-honest security model if no information (other than the inputs and outputs of the protocol) is leaked to such adversarial parties.

The second is the malicious security model. In this setting, there are adversaries that may deviate from the protocol. That is, they may skip steps in the protocol, and send incorrect outputs to other parties. Such adversaries can perform attacks to learn more information that involve additional interactions with the other parties. Furthermore, they may try to corrupt the overall result of the protocol or the results obtained by one party. A protocol is secure in the malicious security model if it is not possible for such adversaries to successfully complete their attacks, i.e. to learn more than they should or compromise the protocol or other parties.

In practice, however, the use of protocols that are secure in the malicious model is often infeasible, because the required secure computations make the protocols very expensive to run [3]. On the other hand, the semi-honest model is often viewed as a weak setting, because it involves only a very basic adversary that will not attempt any malicious interactions at all. In reality, there are many settings where parties are willing to do all sorts of attacks and even deviate from the protocol, given that their malicious behaviour goes unnoticed by the other parties. That is, a setting with adversaries that perform undetectable malicious actions. This has led to the development of a new security model: the covert model. A protocol is secure in the covert model if it is not possible for an adversary to successfully complete attacks without it going unnoticed by other parties. It is reasoned that this a more realistic model than the other two, because in many practical settings there are adversaries more malicious than semi-honest, meaning they will try secretly try some attacks, but not fully malicious because being detected would have too great consequences for them.

3

Related Work

In this chapter, relevant works relating to the topics of AML, PPAML, and entity resolution are discussed. First, relevant background on AML and works covering AML transaction monitoring are explained in Section 3.1. Then, current privacy preserving transaction monitoring methods are covered in Section 3.2. Moreover, state-of-the-art privacy preserving transaction cycle detection methods are explained and analysed in Section 3.3. Next, the domain of entity resolution is explained in Section 3.4. Finally, privacy preserving entity resolution and relevant works are discussed in Section 3.5

3.1. Transaction Monitoring

Transaction monitoring is one of the two core branches of anti-money laundering, the other being the 'know-your-customer' (KYC) procedure. Whereas KYC is mostly a static step that is executed once (usually when the financial institution and the customer initiate their relationship), transaction monitoring is a dynamic and long term procedure that involves keeping an eye on the customer's financial activities involving the institution.

Cox, in his Handbook of Anti-Money Laundering (2014) [18], defines transaction monitoring as scrutinising customer transactions, and checking if the transactions are consistent with the institution's knowledge and risk profile of the customer. He explains banks should monitor for specific types of transactions that are deemed unusual or suspicious. Such transactions include transfers of large amounts, cross-currency transactions and cross-border transactions. Also, Cox mentions that transactions that are not noteworthy on their own, but that are contrasting to the customer's previous activity should also be considered suspicious. For example, a customer who shows a sudden increase in the number of cash transactions should be further investigated. Another part of transaction monitoring is analysing who the other parties are that a customer is involved with. The handbook explains that if a customer has a transaction with another person or business that is known to have high risk, the customer themself should also be assigned a higher risk score. This concept is known as risk propagation [64]. The book also lists a number of other characteristics of transactions that can be used as indicators of money laundering, such as geographic area of the payment and the nature of the product that is being bought.

Overall, Cox describes transaction monitoring as the analysis of transaction data relating only to one bank account. This can be explained by the fact that in the traditional intra-bank AML system, very little data on the neighbouring accounts and their transactions is available. Put differently, banks only have a very limited view of the complete transaction network. They can only model a graph of their own accounts and the accounts that their own accounts have transactions with.

In contrast, more recent transaction monitoring methods aim to make the AML process more effective by looking beyond the behaviour of just one account and instead focus patterns in transactions between many accounts. These methods, sometimes collectively termed 'anomaly detection', analyse a transaction graph to find patterns, or activity by multiple accounts, that indicate money laundering [20]. These patterns are specific subgraphs or graph structures that are the result of different money laundering techniques. Examples of these patterns are shown in Figure 3.1 below [2].

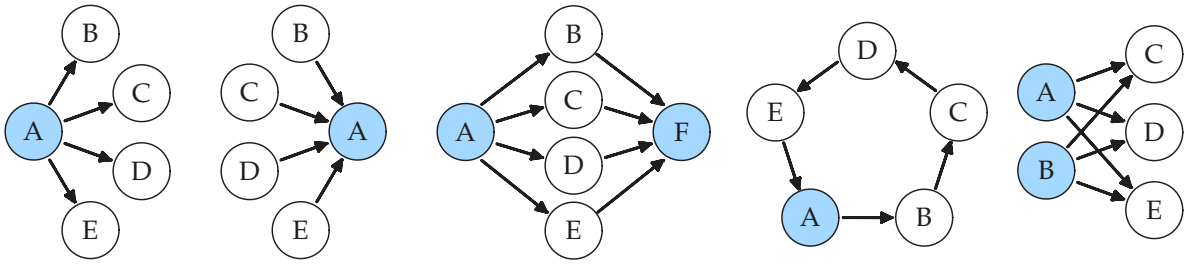


Figure 3.1: Five transaction graph patterns that indicate money laundering. From left to right: fan-out, fan-in, gather-scatter, simple cycle, bipartite.

Many of these anomaly detection methods are based on machine- and deep-learning techniques, aiming to leverage the power of such techniques to efficiently find patterns and to find more complex patterns that are more difficult to detect with traditional algorithms. Machine- and deep-learning based transaction graph anomaly detection methods include the work by Blanuša et al. (2024) [11], who create a library to extract suspicious patterns from a transaction graph, which are then used as additional features in a graph machine learning pipeline. This approach increases the performance of graph neural networks for detecting money laundering patterns. Similarly, Dumitrescu et al. (2022) [20] introduce a method where one first computes node features from a transaction graph, which are then used as input for a machine learning based anomaly detection algorithm.

While such learning based methods show good potential for empowering AML, traditional graph algorithm based solutions are more relevant for this thesis. Such methods find anomaly patterns by simply traversing the transaction graph. In the transaction monitoring context, most works focus on finding the simple cycle pattern (second from right in Figure 3.1). The most basic solution is to ignore the intricacies of the financial domain and just apply a general-purpose cycle detection algorithm to the transaction graph, such as Johnson's algorithm (1975) [35]. However, solutions developed specifically for the purpose of transaction monitoring are more suited to the domain and can achieve better detection.

Qiu et al. (2018) [51] are one of the first to construct such a pattern detection algorithm in an AML context. In their work, the transaction graph of a large e-commerce platform is enhanced by adding edges to the graph that represent relations between account owners. They reason that a cycle in this graph, which consists of accounts connected by transactions and relations, is a very strong indicator of fraud, is because money is moved between accounts, but ends up with the same person. They argue that while a direct transaction cycle between accounts may not exist, if there is a relation between the person owning the source account and the person owning the destination account, it is likely that the money has been passed between them outside the platform. Their algorithm for detecting these cycles works in real-time: it runs for every new edge coming in, and it uses a sliding window over the timestamps of the transactions to ignore transactions that are no longer valid. They also explore the scale-free nature of the network, and introduce a mechanic called the Hot-Point Index: a pre-computed index of all paths between hot-points, i.e. vertices with a very large out-degree. Using this index prevents their real-time depth-first search to follow paths including a hot-point, which would explode the search space.

Hajdu & Krész (2020) [32] developed an algorithm to detect transactions which also takes into account the values on the edges. Such values are the attributes of the transaction, such as amount and timestamp. The algorithm is given parameters, and only traverses edges based on their values and these parameters. This makes that the algorithm can, for example, only look for cycles which have a constant amount transacted between the accounts or which have a maximum amount of time between each subsequent transaction. This mechanism makes the depth-first search approach more efficient and the money laundering detection more accurate. Additionally, the Hajdu & Krész observed that in a transaction network, multiple accounts might be owned by the same entity. They state that a cycle between entities is as much an indicator of money laundering as a cycle between bank accounts. Hence, their algorithm does not run directly on a transaction graph, but on a hypergraph of entities. In their method, this hypergraph is created by merging the vertices representing accounts owned by the same entity, obtaining new vertices that represent only the entities. The edges are still the transactions, but they now link the entities. Evaluation of their method on real transaction data shows that it can detect more instances of money laundering, because instances where a culprit uses multiple accounts can now also be detected.

3.2. Privacy Preserving Transaction Monitoring

The methods mentioned in the previous section all detect money laundering patterns, but only in a centralised setting. This means that they only provide a realistic solution for domains where all the transaction data is already centralised, as is the case for Qiu et al. [51]. While many works ([32], [11], [20]) introduce their methods as solutions for money laundering and financial fraud, they forgo that these are largely inter-bank and cross-border problems. So, while these methods are effective in detecting financial crime involving only a single bank, they can only contribute significantly to money laundering detection when financial data is shared and centralised, which, as mentioned in Section 1.2, is ethically undesirable and even illegal in jurisdictions with adequate privacy laws in place.

In contrast, works based on privacy preserving transaction monitoring acknowledge that money laundering detection can only be effective on a large scale when it is done collaboratively. Instead of arguing for increased financial data sharing at the cost of privacy, such works introduce technical solutions employing secure multi-party computation (SMC) techniques. This way, they demonstrate collaborative transaction monitoring is possible and effective without financial data having to be explicitly shared or centralised.

Brand et al. (2023) [13] introduced FinTracer, a protocol for authorities to privately discover if a path exists in a transaction network between two given accounts. The protocol is mainly created for one specific use case, namely the detection of paths between a set of source accounts (bank accounts receives large disability benefits) and a set of destination accounts (accounts known to transfer large amounts of money to other countries). However, since the protocol highly configurable in the conditions for the accounts and transactions to be included in the path, it can be used to detect all sorts of money flows and even specific topologies in the transaction graph. The protocol uses queries to specify the sets of source and destination nodes, and the conditions for the path to the participating banks. The protocol is run by propagating a tag, which is a bitset representing all bank accounts in the transaction network. A bit being set indicates it is present in the path. To prevent other banks learning any information about the path from the tag, the bits in the tag are encrypted with a semi-homomorphic encryption scheme. Specifically, the ElGamal scheme over an additive twisted Edwards elliptic curve is used. However, even with the encrypted tag, information can be inferred: unchanged ciphertexts allow banks to link multiple tags together, leaning the existence of transactions between other banks, and to learn which bits have been flipped. To prevent this, the protocol involves refreshing the randomness on all ciphertexts when a tag is propagated by homomorphically adding zero to it. In the FinTracer setup, there is a centralised party creating the query that can initiate the protocol. For the work's context of benefit fraud, this is the national financial intelligence unit (FIU). The FIU is also the party generating the keys (distributing the public key to the banks and keeping the private key) and decrypting the result to learn the path.

Sangers et al. (2019) [56] present a method to compute the PageRank value for accounts in the transaction network of multiple banks. The PageRank value is a measure for vertex centrality in a directed graph that banks can use as an indicator of transactional fraud. The PageRank algorithm can be formulated in almost exclusively linear operations. In the method, computations are divided over the involved parties in such a way that when a division is required this can be done by a party that knows the divisor, making that all multiplications required are multiplications by known constants, not by ciphertexts. This makes that the algorithm can be computed on encrypted values using an additively homomorphic encryption scheme. The protocol by Sangers et al. uses the Damgård-Jurik encryption scheme, a generalisation of the Paillier encryption scheme. This only requires the PageRank algorithm to be lightly modified to operate in $\mathbb{Z}_N$ instead of its normal operation over the real numbers. To prevent one party having access to the decryption key, breaking the security of the system, the protocol employs distributed key generation and decryption methods. The authors show that their protocol scales linearly with the number of bank accounts and the average out-degree (outgoing transactions) of the accounts, and that the number of parties (banks) only has minor impact on the complexity. Furthermore, it is demonstrated that such a protocol is highly parallelisable, making it suited for a setting where the parties collectively have a large number of computing cores to run the protocol.

In a related work, Van Egmond et al. (2025) [64] developed a private collaborative risk propagation protocol. Risk scores are values assigned to accounts by their bank based on a number of factors, including analyses from the KYC procedure and other factors described at the start of Section 3.1. Risk scores need not be one value, there can be a whole set of different kinds of risk scores per account. The concept of risk propagation is that the risk associated with one account is affected by the accounts it has transactions with. If an account sends money to or receives money from another account with a high risk score, its own risk score goes up. How much a risk score changes is determined by parameters and the weight on the edge. This weight can be a value associated with the transaction, such as the value or total number of transactions. Van Egmond et al. use secure multi-party computations to run a risk score propagation algorithm on a transaction graph distributed among multiple banks. Specifically, additively homomorphic encryption is used to encrypt each account's risk score attributes and share them securely with the other parties. For each account, the received risk scores of its neighbours can be homomorphically multiplied by the weights on the transactions (constant multiplication), and homomorphically added to its own encrypted risk scores to update them. Like with [56], the key generation and decryption happens in distributed manner, such that no one party can decrypt on their own. Van Egmond et al. evaluate their protocol and show that the runtime is linear in the number of accounts.

3.3. Privacy Preserving Transaction Cycle Detection

In addition to the privacy preserving transaction monitoring methods mentioned so far, there exist two related works that focus specifically on the detection of one common money laundering transaction pattern, the simple cycle. Such a simple (or short) cycle is considered a strong indicator of money laundering because such a pattern is very unlikely to be the result of legitimate transactions in the normal financial ecosystem [32]. Therefore, if a cycle of six or fewer transactions exists in a transaction graph, it is very likely to be the result of money laundering activities.

Porsius Martins (2023) [49] introduced the first protocol for privacy preserving cycle detection in a distributed transaction graph. The protocol makes use of the one-time pad encryption scheme, wherein a bit sequence is encrypted by XORing it with a key that is also a bit sequence of the same length. A property of this scheme is that encryption and decryption is the same operation, i.e. XORing a message with the key gives the ciphertext, and XORing the ciphertext with the key gives the message.

Porsius Martins utilises this property as follows: each edge in the transaction graph is associated with a random 128 bit value. One vertex (bank account) initiates the protocol by taking another random 128 bit value, making sure to remember it. Then, a message is sent to all the vertex's out-neighbours which consists of the random value encrypted (XORed) with the value on the edge connecting the neighbour. Each other vertex, when receiving such a message, propagates it to all its neighbours by repeating the process of re-encrypting the message with the value on the edge to the neighbour. If there is a cycle in the graph, the vertex that sent the original message will receive it back from one of its neighbours. However, it does not recognise it because it was encrypted multiple times for every edge it passed though. So, the vertex propagates it as normal. The result is that the message will go around the cycle again, being re-encrypted with the same keys (values on edges). However, since re-encrypting in the one-time pad scheme is the same as decrypting, the message will actually be decrypted the second time it goes around. The result is that the message has been fully decrypted once it arrives at the original vertex for the second time. The vertex now recognises it and concludes that there must be a cycle. The protocol is very efficient because the one-time pad XOR operation is computationally inexpensive compared to other SMC-based techniques.

When a cycle is found, the initiating account does not know which other accounts are part of the cycle. To discover this, a second step is executed to retrace the steps in the cycle. Each vertex communicates with the vertex it had previously received a message from. This way a list is constructed of all the accounts in the cycle. Furthermore, an important detail of the protocol is that the messages are tagged with a time-to-live (TTL) counter, which is decreased by 1 every time a message is forwarded. This is used to prevent the transaction network being flooded with messages and to only detect short cycles of a maximum length. Since the messages must go around the cycle twice, the initial TTL is set to twice the max cycle length, i.e. 12.

In the work, it is claimed the protocol is secure because the only data shared are the messages, which consist only of encrypted random values that change at every hop, preventing other accounts (and banks) from linking messages to learn about the topology of the transaction graph.

Jense (2024) [34] builds on Porsius Martins' ideas by presenting a different protocol with the same objective. This work shows the need for a different approach by highlighting a security flaw in the existing method: layers of encryption can be removed by semi-honest vertices by XORing messages retrieved as part of the same cycle. This allows the account to recover the original random number and by doing so link together different messages to learn about the topology of the transaction network.

To address the problem, Jense introduces a new protocol, which is similar in that it also involves the propagation of messages and the detection of a cycle by the initiating vertex. The protocol utilises a property of the Diffie-Hellman key exchange mechanism. In such a key exchange, two parties construct a shared key by exchanging the building blocks required, without any other intermediate parties being able to use these building blocks to reconstruct the key for themselves. In the protocol, a vertex (bank account) initiates DH-key exchange by sending all its out-neighbours the first building block for the shared key. All vertices receiving such a message then propagate the message, and also construct the shared key and reply with the other building block such that the initiating vertex can also construct the shared key. The initiating account concludes that a cycle is present if it finds that it has completed a key exchange with itself. That is, it has found that it obtained the same key from using both a building block it received as a response to its own message, as well as from using a building block it received as a message initiated by another vertex (which now turns out to be itself).

For this system to work, the vertices need to remember which of their in-neighbours to forward a response to. This is achieved by tagging each message with a random nonce. The mechanism is shown in Figure 3.2 below. Say vertex v_i initiates an instance of the protocol by sending a message to its neighbour v_j. v_i tags this message with random nonce o_{ij} (1). When v_j forwards this message to its neighbour v_k, it takes a new random nonce o_{jk} and replaces the other nonce on the message with this (2). v_j will remember that o_{jk} for v_k corresponds to o_{ij} for v_i (3). When v_j receives a response from v_k, it is tagged with o_{jk} (4). v_j looks up o_{jk} in its memory and finds that it corresponds to v_i with nonce o_{ij} (5). v_j can now forward the response to the correct neighbour and tag it with the same nonce that was previously used (6).

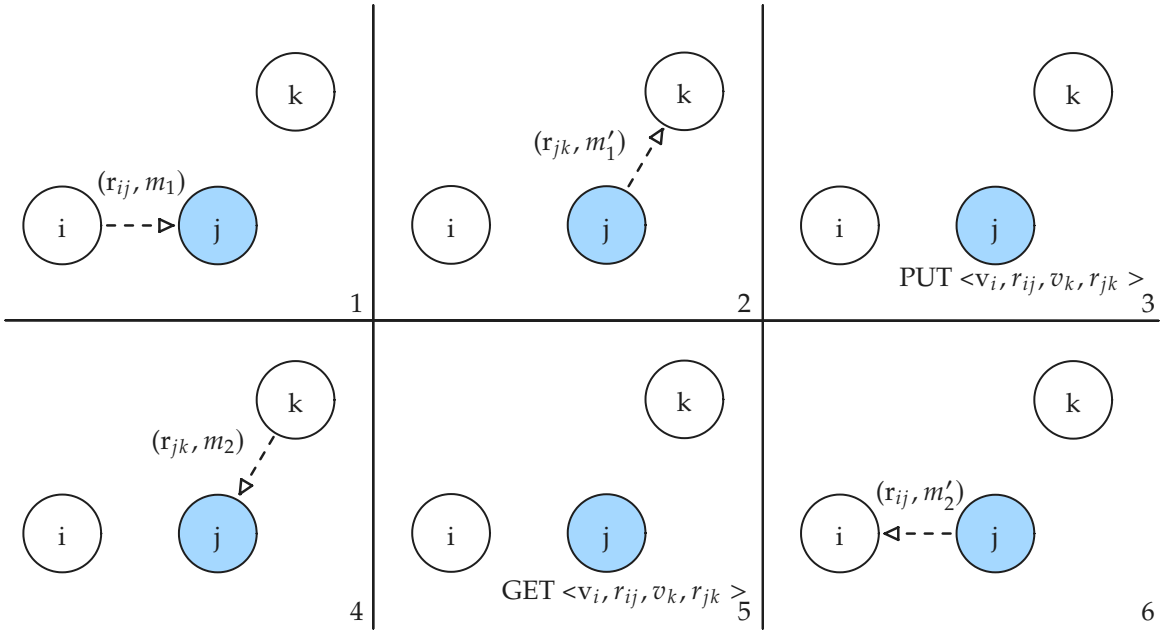


Figure 3.2: Jense's (2024) method for privately routing messages in a decentralised transaction network. 1: v_j receives a message with a nonce o_{ij}. 2: v_j propagates the message and includes a new nonce o_{jk}. 3: v_j stores both nonces. 4: Later, v_j receives a response with a previously used nonce o_{jk}. 5: v_j looks up this nonce to find the neighbour whereto to forward the response and the original nonce o_{ij}. 6: v_j forwards the message to the correct neighbour with the nonce signalling the instance of protocol.

Jense's protocol avoids the problem associated with Porsius Martins' protocol by guaranteeing the unlinkability of messages. This is achieved by ensuring all messages and responses use different values or have new randomness introduced to them. For the exchanged building blocks this involves exponentiating them with a random number each time they are forward. This does not break the functionality because the same randomness ends up in the shared key for both parties. Similarly, the nonces used to keep track of messages and responses are changed every time a message is propagated. A nonce is unique for each edge and each instance of the protocol. This makes that messages cannot be linked based on the nonces. The only part of the messages that can be linked is the TTL counter, which here starts at only once the max cycle length (6). However, since there are only six different values for the counter and there are many messages from many instances of the protocol, it becomes very difficult for semi-honest vertices to accurately link messages based on the TTL value. So, while the messages leak a little bit of information, this is not significant and is accepted in order to make the protocol work.

While these two protocols are able to effectively detect cycles between bank accounts in a transaction graph, there is reason to doubt their effectiveness in detecting cycle based money laundering on a larger scale. When criminals are laundering money in a cycle, their objective is for the money to end up with the same owner as it started with. It is important to realise that in practice it does not matter that the money ends up in the same or in a different bank account, just that the money ends up in a bank account that is controlled by the criminal. Hence, this means that the transactions in the money laundering process do not necessarily have to from a cycle of bank accounts, but rather a cycle of entities. In essence, a cycle of bank accounts is simply a special case of an entity cycle where it just so happens that all entities use one bank account. Figure 3.3 illustrates how money can still be laundered in a cycle, even though no cycle in the transaction graph exists.

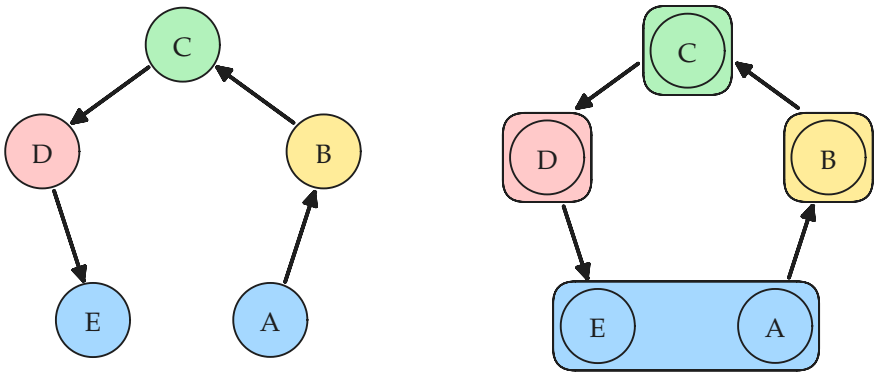


Figure 3.3: Transaction pattern with five bank accounts owned by four different entities. Left: no cycle is present in the normal transaction graph. Right: a cycle is present if the vertices are combined based on the entities owning the accounts. This indicates money laundering for the same reason as a cycle in the normal graph does: one entity is moving money from itself to itself with a number of accounts in between (layering). The existing protocols do not detect this kind of money laundering.

This concept is also explained in the work by Qiu et al. [51]. In their application, they enhance the transaction graph with edges representing relations between accounts, such as a seller and buyer account being owned by the same person. By introducing such an edge, they link accounts owned by the same entity. Consequently, they are detecting cycles that are not just between accounts, but between entities. The idea is even more explicitly described in the work by Hajdu & Krész [32], whose method does not run on a normal transaction graph, but on a hypergraph where bank accounts of the same owner are combined into entities. They state that, even within one bank, a person or organisation can have multiple accounts, and that cycles between such entities are as much an indication of financial fraud as cycles exclusively between accounts.

Furthermore, from an adversarial perspective, criminals are aiming to hide their money laundering activity. From their perspective, when they need to launder money in a cycle, it might very well make sense to add another layer of obscurity to the process by making sure to use a different destination account for the money. Criminals might be especially motivated to take this little extra step when they realise the current state-of-the-art methods cannot detect money being laundered when the source and destination accounts are different.

3.4. Entity Resolution

Entity resolution (ER) is the problem of identifying records that refer to the same entity [10]. Records, in this context, are elements of a dataset that describe an entity. A record consists of attributes, which are individual pieces of data, such as a string, number or date, denoting one specific aspect of the entity. An entity is an object, individual or concept that exists in the real word. The entities in a dataset depend entirely on the context. A common type of entity is a person, where its attributes are qualities like name, date of birth, phone number, etc. An important characteristic of entity resolution is that the records do not contain an attribute that serves as a unique identifier within the dataset. Instead, two records can only be distinguished by the values of their attributes.

A systematic overview of the entity resolution tasks is given by Christophides et al. (2020) [16]. They explain that ER is typically used to combat issues arising from data on entities in single or combined datasets, such as incompleteness, overlapping and conflicting records. When ER is used on the same database or -set it can be called de-duplication, and when it is used to combine multiple data sources it is often referred to as record linkage. The authors identify four key stages in the entity resolution process. The first stage is blocking. This stage deals with the problem that in modern big-data applications, the volume of records can be so large that doing a direct comparison between two records for every possible pair in the data takes up a gigantic amount of resources and time, while typically the vast majority of records are unlikely to match. Blocking involves grouping together records that are likely to describe the same entity. Then, only the records grouped in the same block have to be compared with each other, greatly reducing the required number of comparisons. The blocking process should use only a lightweight mechanism to determine which records are similar, such as using the value of one attribute or length. The second step is block processing. Here, each block is restructured to minimise the number of required comparisons further. The third phase, the matching step, is the core of the ER process. It involves deciding for each pair of records in a block, if the two records describe the same entity or not. The core idea of matching is the records are judged to refer to the same entity if their attributes have high similarly. The main approach is to compute the similarity for every attribute using a similarity or distance function. By adding up the distance scores and using a pre-defined threshold, a pair is judged to match or not. This process can be further refined by using a linear combination of the attribute scores or using more complex rules with thresholds for the individual attributes. The last step in the ER pipeline is clustering. This involves combining matched records into one to obtain a dataset where one entity is described by exactly one record, which holds all combined information. The process is transitive: two records that have not matched directly but have both matched with the same record will also be combined.

Binette and Steorts (2022) [10] published review and classification of different methods typically employed for entity resolution. These methods mostly pertain to the matching step, but many can also be used for the blocking stage. The most commonly used entity resolution methods are based on a series of deterministic rules for the comparison of the record attributes. This type of ER is referred to as deterministic entity resolution or rule-based entity resolution. In such methods, the attribute values of both records are compared. This can be binary, i.e. they match or not, or similarity based. Many different distance metrics can be used to compute the similarity between two attributes, such as Levenshtein distance for words in western languages, Jaro-Winkler distance for names, and Jaccard similarity for unstructured strings. In deterministic methods, rules and thresholds are constructed for deciding a match. There can be many rules depending on the number of attributes and the complexity of the domain. In practice, these rules are often hand-crafted specifically by domain experts, but they can also be learned from the data. Deterministic methods are very suitable for records with structured attributes without much noise in them. When the data is more noisy, probabilistic methods might have better performance. In this category, a continuous probability function is used in place of rules. The similarity of the records determines the probability of them being a match. Decision boundaries are used to label each pair a match, no match, or possible match. The objective then is to optimise the boundaries and minimise type-I and type-II errors. Probabilistic ER methods have been used for many decades, and the approach has since evolved into machine learning and deep learning techniques. Binette and Steorts group all methods applying a clustering technique into the third category. In these methods, many records can be matched together as opposed to traditional pair-wise comparison.

Entity resolution is a common task in anti-money laundering. AML platform developer Quantexa explains how ER is used to connect data points spread over the different systems within one financial institution [52]. This allows institutions to harmonise their data and build a structured profile of each of their customers. It is also used to determine if two legally different customers are actually the same entity. For example, a bank may have a person named John Smyth as a customer and also a company which has its owner listed as someone named John Smith. Banks must resolve if these two customers refer to the same entity in order to properly monitor for money laundering. Facctum, another company providing AML services, explains how entity resolution is important in both the know-your-customer and transaction monitoring steps [24]. For the KYC procedure, entity resolution helps a bank to establish a more accurate picture of the customer by linking information with existing records and watch lists, and by identifying relations with subsidiaries and intermediaries.

One example of how entity-resolution is specifically used for AML is presented in the work by Fior et al. (2022) [28]. They introduce a method to link the names given in bank transactions to known organisations. Here the names relate to organisations that are not customers of the bank in question, meaning there is no other data available other than the name and IBAN number listed on the transaction. The problem described by Fior et al. is complex, because many large organisations operate many bank accounts and by nature the names listed on transactions are noisy and ambiguous, e.g. a transaction to 'Bank of America' might only be named in the transaction data as 'BoA NYC'. Resolving these names to entities provides very valuable information for the transaction monitoring process. The work introduces a data processing pipeline where the names are first cleaned by transforming the name to uppercase, removing any punctuation, and removing any standardised stop words. Then, the common ER processing steps are applied, such as distance computation with the Hamming, Jaro-Winkler and Demerau-Levenshtein distances, matching and clustering.

3.5. Privacy Preserving Entity Resolution

Privacy preserving entity resolution (PPER) techniques are a subset of regular entity resolution that focus on linking records in datasets owned by different parties without revealing one's actual records to others [30]. Since PPER is always about linkage and never about de-duplication, it is also frequently referred to as privacy preserving record linkage (PPRL). PPER is used in applications where the linking of multiple datasets is very useful, but the data are also very privacy-sensitive, e.g. social sciences, crime detection, and healthcare patient research. Records in such datasets contain many attributes with both personal identifiable information (PII) and very sensitive information, such as health- and crime records. This makes that such data cannot be shared with other parties. Even if the data custodian trusts the other party completely, privacy regulations forbid disclosure of such information.

Vatsalan et al. (2017) [65] present a literature review to describe different facets of PPER and the most commonly used techniques. The authors first underline that PPER is always a trade-off between scalability, linkage quality and privacy. Different methods have been developed for different applications, and they typically offer good performance for two of these aspects in favour of decreased performance for the third. To put it in other words, introducing privacy to ER requires one to sacrifice some scalability or quality when compared to non-private ER techniques. PPER methods usually contain the core steps of normal ER (blocking and matching) but add extra steps. For the blocking, this step involves converting the records to a format which hides the actual information from the other party while keeping the linking utility. A lot of Works in PPER use a variety of anonymisation or masking techniques. Here, the contents of the records are obscured by adding noise, using pseudo-random functions or using a distance preserving encoding. For the matching stage, more secure privacy preserving techniques are used, such as secure hash-encodings and encryption schemes. Vatsalan et al. also give an overview of the possible attacks on PPER methods. These include dictionary attacks on deterministic methods, and cryptanalysis attacks.

Gkoulalas-Divanis et al. (2021) [30] published a chronological overview of works on PPER. The first generation of PPER techniques (1998-2004) is defined as mostly focussing on the exact matching of attributes. For exact matching, simple secure multi-party computation (SMC) techniques are used for strong privacy. However, these techniques were seen as slow and not accurate. In the second generation (2004-2009), many methods were created that were capable of private fuzzy matching of attributes, i.e. using the distance between them. This saw the introduction of many anonymisation and encoding techniques, notably the use of the Bloom filter. The third generation of PPER (2009-2014) mainly focusses on scalability of methods to deal with the problem of datasets becoming increasingly larger. Particularly, many works introduce more efficient private blocking methods. In this generation, new methods are introduced for two-party PPER (previous works typically used a (semi-) trusted third party). Most attention in the review goes to methods in the fourth generation (2014-). Here, most methods deal with the problem of many modern datasets being more variable and containing much noise. New techniques focus on linking records based on numerical and sequence data, and on more accurate linking of large textual attributes. There is also increased attention for the security of the methods. Previous generations relied on only anonymisation for privacy, even though this is not cryptographically secure. Modern methods find new ways to use SMC methods like homomorphic encryption and secret sharing, trading off some efficiency for PPER that is more secure against attacks.

While not mentioned by Gkoulalas-Divanis, some recent publications suggest the emergence of a fifth generation, namely a generation of methods utilising private deep-learning. Ranbaduge et al. (2023) [53] introduced the first PPER protocol that uses a deep-learning model. Their method can perform entity resolution on databases with limited training data available and uses differential privacy for provable security. Similarly, the CampER framework by Guo et al. (2023) [31] allows parties to use deep learning with provable security. Parties first individually create an encoder based on their own data. Then, they collaboratively fine-tune and align their encoders. Finally, they use partially homomorphic encryption to securely compute the similarity between encoded records.

One of the most influential works in the domain of privacy preserving entity resolution is that of Schnell et al. (2009) [58]. Stemming from the second generation of PPER, the authors sought to develop a new method that allowed for private fuzzy matching of attributes. Where previous works mainly focussed on phonetic encodings and embedding spaces, Schnell et al. aimed to develop an anonymisation technique that is efficient and versatile. In their method, they use Bloom filters for anonymisation. A Bloom filter is a probabilistic data structure that is used to represent membership of elements in a set. If an element is present, then some corresponding positions in the filter are always set (see Section 2.6 for details). As a consequence, Bloom filters can be used to approximate the similarity between two sets: if the sets have many elements in common, then the Bloom filters will have great overlap in the positions that are set to 1. Conversely, if the Bloom filters have high overlap, the sets are likely similar. To use Bloom filters for PPER, the attribute strings are split into bigrams. The bigrams then form a set that can be encoded as a Bloom filter. When creating the bigrams, the strings are first padded on both ends with the stop character '_'. This makes that the string *smith* becomes the set $\{\_s, sm, mi, it, th, h\_\}$. This ensures that a one character difference always results in a difference of two bigrams. Creating the Bloom filters requires using k independent hash functions. To easily obtain an arbitrary number of such functions, the authors use the method by Kirsch and Mitzenmacher (2008). Here k independent hash functions are created by linearly combining two independent hash functions h_1 and h_2 as follows:

$$g_i(x) = h_1(x) + ih_2(x) \mod l \tag{3.1}$$

for i in range 0 to $k-1$, where l is the length of the Bloom filter. For h_1 and h_2, Schnell et al. suggest using SHA1 and MD5. After creating the Bloom filters, parties send them to a third party, who does the matching by computing the similarity. Many similarity or distance metrics can be used on Bloom filters. Here, the Dice coefficient is used. The accuracy of the approximation of the similarity depends on the false positive rate of the Bloom filter encoding, which in turn depends on the filter length l, the number of hash functions k and the size of the set $|X|$. To find optimal values for l and k, Schnell et al. evaluate their method by comparing the precision of their Bloom filter method with varying parameters to the precision of exact comparison and of other PPER methods. They do this with both synthetic data and real records of German names. They find that their method can achieve near equal precision to an exact comparison approach, and that a filter length of 500 bits with 15 hash functions is optimal.

While the Bloom filter encoding method provides anonymisation of the input string, it does not provide any cryptographic confidentiality. That is, there exists no proof that the encoding satisfies any formal notion of security. Moreover, Kuzu et al. (2011) [40] have shown that the method is vulnerable to a cryptanalysis attack. They use a frequency based attack to demonstrate that the original string values can be learned from the corresponding Bloom filters. The attack is computationally feasible when the parameters are optimised for linking performance, like is the case for the method by Schnell et al. and other applications of the method. Kuzu et al. show that the attack can be made more difficult by changing the parameters, such as using 4-grams in place of bigrams and using more hash functions for a set filter size. These changes, however, degrade the precision of the Bloom filter PPER method. A similar conclusion is reached by Vos et al. (2024) [66]. They show that probabilistic nature of the filters causes information about the set to be leaked. Similarly, they suggest that changes to the encoding method and its parameters can make it more secure, but that this comes at the cost of reduced performance.

To address the security flaws of the Bloom filter based PPER method without compromising the linkage quality, Karapiperis and Verykios (2015) [36] introduced a protocol that provides a cryptographically secure matching technique. Their work, which also introduces a new blocking method, presents a three party protocol with an optional extension that can be used if the data custodians do not fully trust the third party. The method uses additively homomorphic encryption to compute a distance metric between two encrypted Bloom filters. This works as follows: both parties create Bloom filters for each of their records' attributes with the technique described by Schnell et al. [58]. Karapiperis and Verykios concur that using a filter length of 500 bits with 15 independent hash functions is optimal for short strings, such as first- and last names, addresses, and place names. Each of the two parties needs to encrypt half of the records that need to be matched and send these to the other party. Which records need to be matched and which party does which (the work is divided in two) is determined by the third party after the blocking phase has been completed. The encryption happens with the Pallier encryption scheme. Given a Bloom filter $BF =< b_1, b_2, ..., b_l >$, it is encrypted as:

$$\text{Enc}_{pk}(BF) =< \text{Enc}_{pk}(b_1), \text{Enc}_{pk}(b_2), ..., \text{Enc}_{pk}(b_l) > \tag{3.2}$$

Where pk is a public encryption key generated by the third party. To encrypt a record, every attribute Bloom filter is encrypted this way. After the encrypted records are exchanged, for each record pair that it needs to match, each party now has its own unencrypted record and the other party's encrypted record. For each attribute in the record pair, a party computes three intermediate variables a, b and c, or rather encryptions thereof. Say, Alice is the party that created an encryption of her Bloom filter and sent it to Bob, who has his own plain filter and will find the intermediate variables. a is the number of positions in the filters where both Alice and Bob have 1, which Bob can find by summing all the positions in Alice's filter where he has 1. b is the number of positions set to 1 in Alice's filter, but 0 in Bob's filter, which he finds by summing all the positions in Alice's filter where he has 0. Finally, c is the number of positions with 0 for Alice but 1 for Bob, which is the total number of positions where Bob has 1 minus the number of these positions which are also 1 for Alice, the latter being a. All these calculations include summations and can thus be computed homomorphically on the encrypted filters:

$$\text{Enc}_{pk}(a) = \text{Enc}_{pk}(\sum_{\psi \in \Psi} BF_A[\psi]) = \prod_{\psi \in \Psi} \text{Enc}_{pk}(BF_A)[\psi] \mod n \tag{3.3}$$

$$\text{Enc}_{pk}(b) = \text{Enc}_{pk}(\sum_{\omega \in \Omega} BF_A[\omega]) = \prod_{\omega \in \Omega} \text{Enc}_{pk}(BF_A)[\omega] \mod n \tag{3.4}$$

$$\text{Enc}_{pk}(c) = \text{Enc}_{pk}(\sum_{\psi \in \Psi} BF_B[\psi] - a) = \text{Enc}_{pk}(\sum_{\psi \in \Psi} BF_B[\psi]) \cdot \text{Enc}_{pk}(a)^{-1} \mod n \tag{3.5}$$

Here, Ψ are the indices where Bob has 1 in his Bloom filter and Ω the indices where he has 0. Note that $\text{Enc}_{pk}(\sum_{\psi \in \Psi} BF_B[\psi])$ is the number of positions set to 1 in Bob's own filter, which he can compute and encrypt himself. Once the parties have computed these three encrypted intermediary values for every attribute in every record, they send them to the third party, who decrypts them. Then, for every attribute, the third party computes a distance metric using a, b and c.

Karapiperis and Verykios list three common distance metrics that can be computed this way: The Jaccard distance (d_J), the Hamming distance (d_H) and the squared Eulidean distance (d_E^2). These can be computed as:

$$d_J = 1 - \frac{a}{a+b+c} \tag{3.6}$$

$$d_H = b + c \tag{3.7}$$

$$d_E^2 = a + b + c \tag{3.8}$$

After finding the distances between the attributes for a record, the third party decides if there is a match using a deterministic matching strategy.

The protocol overcomes the security issues associated with the Bloom filters, because the filters themselves are never shared. Only encryptions of the filters are shared between the parties, which are provably secure. The Bloom filters are only used as an efficient encoding of the bigrams. The encryption and homomorphic operations make the process more expensive. Therefore, this technique can be seen as a trade-off between efficiently and privacy.

4

Protocol for Entity Cycle Detection

The chapter introduces the protocol for detecting entity cycles in decentralised transaction networks. First, Section 4.1 explains the objective of the protocol and discusses the problem that it solves. Then, Section 4.2 gives a concise overview of the workings of the protocol. Finally, in Section 4.3, the exact steps of the protocol are presented in detail.

4.1. Protocol Objective and Problem Definition

The goal of the protocol is to detect short cycles of entities in a transaction network. Such a cycle is present when money is moved between accounts and entities, but ultimately ends up with the same entity it started with. Since in a transaction network the vertices are not entities but bank accounts, the goal is to detect if the money ends up in a bank account that is owned by the same entity as the first bank account. So, in a transaction network, an entity cycle is a path between two accounts with the same owner. Such a path is only an indication of money laundering if it is short: 3 to 6 transactions [32].

For the protocol to work, it must recognise that two accounts are owned by the same entity. This can be achieved by looking at the customer information associated with the account, which is here referred to as the identity of the account. Every bank account should have a basic set of attributes as an identity, because banks are required to collect such information in the AML KYC procedure. Figure 4.1 and Table 4.1 show an example of a short entity cycle. The figure shows a transaction graph were no cycle between accounts is present. The table shows the identity of the entities owning each of the accounts in the graph. Both account A and account E are owned by the same person. So, if the money is moved from account A to account E, it starts and ends with the same entity: an entity cycle.

The protocol must detect that these two accounts are owned by the same entity, i.e. that the identities match. In a case where the person registered the two bank accounts in the same country and with the same passport, the matching is easy: the social security numbers (SSN) are likely the same. However, such a situation is unlikely for money launderers, as they often use accounts spread over different countries and the might use passports from two different countries. Another case is also easy: if the person registered with two banks using exactly the same personal information, the individual attributes can be exactly matched, e.g. name is exactly the same, email is exactly the same, etc. However, such a scenario is also not realistic for two reasons: 1) the account owner might have given the banks slightly different information, such as a different email address or phone number, and 2) the banks' systems might make some slight changes to the attributes when storing them, or the personal information might even be slightly different between passports of different countries. Therefore, for the protocol to be reliable, it can not simply do exact matching on the identity attributes and instead must determine if two identities are the same based on the similarity of the attributes.

ID	SSN	First name	Last name		Phone number
A	6464231364	Álice	de Vrij		
B	6831327211	Bob	de Jong		
C	7683333455	Charlie	Janssen		
D	3833353666	David	Smit		
E	8654556644	Alice	Devry		

Table 4.1: Identities for accounts in Figure 4.1. Even though accounts A and E reference the same entity, no single attribute matches exactly.

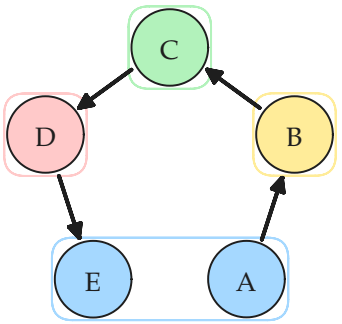


Figure 4.1: Transaction network with an entity cycle. Accounts A and E are owned by the same person.

Table 4.1 shows what such variations in identities can look like. Accounts A and E are owned by the same person, but not a single attribute of their identities match. Firstly, the SSN values are different, indicating that a different passport was used. Furthermore, The name of the person is 'Álice de Vrij'. This has been registered correctly by bank account A. However, somewhere in the process of opening account E some nuances have been lost: the 'Á' is replaced with a simple 'A', the different parts of the last name are concatenated, and the Dutch combination 'ij' is misspelt as 'y'. This could be the result of either the bank's systems not handling the name correctly, or the name simply being defined differently between the two passports. Furthermore, neither the email address nor phone numbers match exactly, even though they are semantically the same.

Furthermore, it is important to note that while this work uses bank accounts owned by individuals as examples, the protocol also works for accounts owned by companies or organisations. These have different kinds of identities compared to persons, but can be matched with each other all the same. Moreover, personal accounts and organisation accounts can still be matched if the personal identity of the legal owner of the organisation is used.

Finally, the protocol must preserve the privacy of the involved parties. This means banks cannot share any of their data with others in plain form. After the protocol is run, banks cannot have learned any new information about bank accounts that they do not manage or transactions that do not involve the bank. The only exception to this is learning information about accounts involved in money laundering: the protocol should allow banks to learn that one of their accounts is part of a short entity cycle. Additionally, they can learn which accounts that they do not manage are involved in a particular entity cycle, so they can work with other banks to investigate further, or report the findings directly to the national FIU.

4.2. Protocol Summary

The protocol detects entity cycles by finding bank accounts that have the same identity using privacy preserving entity resolution. The method that is used for this is based on the matching step introduced by Karapiperis and Verykios (2015) [36]. The protocol operates in a fully decentralised setting. That is, each bank account (vertex in the transaction network) is assumed to be its own agent. This simplifies the protocol, because it removes the need to make a distinction between intra-bank and inter-bank transactions. Like the methods of Porsius Martins (2023) [49] and Jense (2024) [34], the protocol is run by the vertices exchanging messages with their adjacent neighbours.

However, unlike these existing works, the messages include information about the vertices' identities, and the protocol hence employs multiple secure multi-party computation techniques to securely communicate and resolve this data, including homomorphic encryption, blinding and linked values. Additionally, there is a semi-trusted third party (STTP) which is required to do the matching of identities of the accounts.

To set up the protocol, each vertex encodes its identity, i.e. all its personal attributes, as Bloom filters. Also, the STTP generates a secure keypair for the Paillier cryptosystem and informs all the vertices of the public key (pk_{sttp}). An instance of the protocol is stated by one account that is the initiating vertex (v_0). Using a variant of pk_{sttp}, v_0 encrypts its Bloom filters, which are collectively referred to as its encrypted identity. This encrypted identity is sent to all its out-neighbours, who forward it to all their out-neighbours. This forwarding is limited to ℓ steps by a time-to-live (TTL) value included in the message. Each vertex receiving the message computes the distances between the encrypted identity of v_0 and its own identity. For each pair of Bloom filters, the distance is computed homomorphically using the matching method introduced by Karapiperis and Verykios (2015) [36]. Unlike the original method, the match is not immediately resolved, but the distances are communicated back to v_0 in order to prevent dictionary attacks by other vertices. Each vertex includes this set of distances in the response to the message, which is propagated back to the initiating vertex. v_0 will then send its set of distances to the STTP, who decrypts them and decides if there is match or not, informing v_0 of the result. Additionally, all vertices include an encryption of their ID (IBAN) in the response. These are also forwarded to the STTP, who decrypts them and sends them to v_0 if there is a match. This way, the initiating vertex immediately learns the vertex that is matched with and all vertices in the path to it.

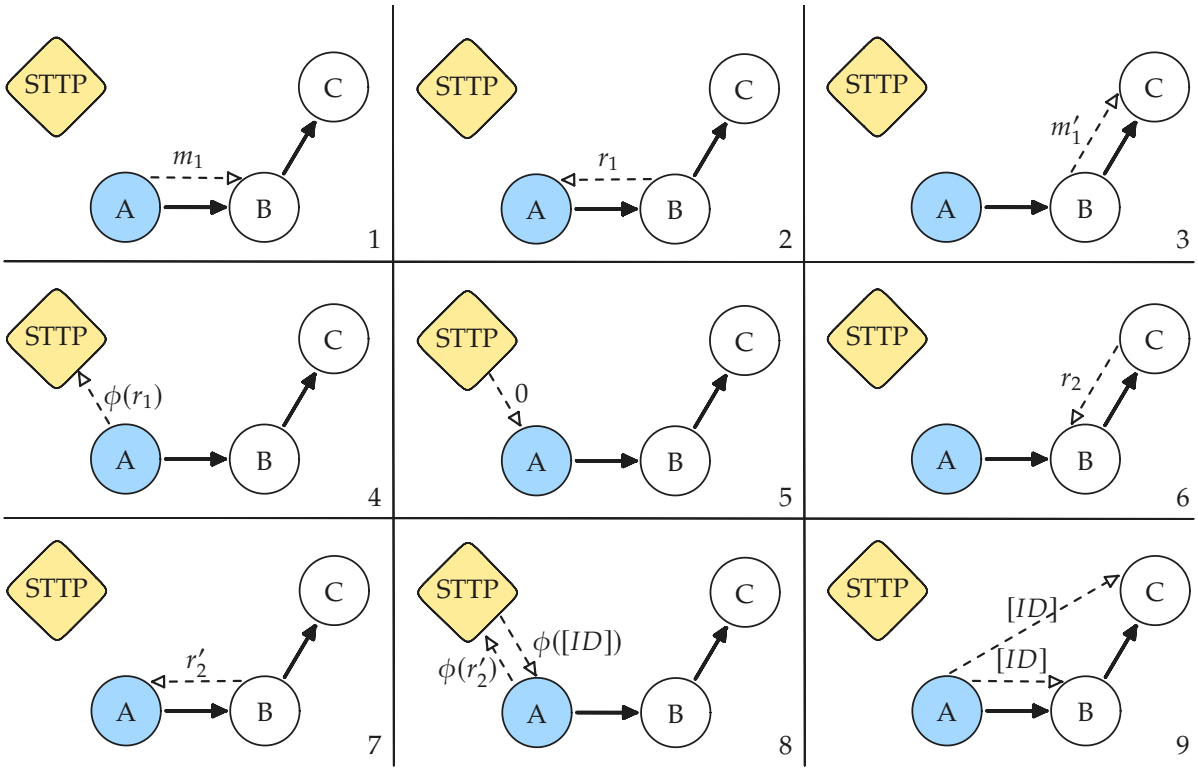


Figure 4.2: Overview of interactions in a protocol instance. 1: After having encrypted its identity and the value 1, A sends these as a message (m_1) to all its neighbours. 2: B computes the distance between the encrypted identity in the message and his own identity, and includes both this and his encrypted ID in the response (r_1) to A. 3: B re-randomises and propagates the message to all his out-neighbours (m_1'). 4: Having received r_1, A blinds the ID component and forwards the response to the STTP ($\phi(r_1)$). 5: The STTP decrypts the distances and determines if there is a match. In this case, it informs A that there is no match. 6: C also replies (r_2) to the message. 7: B re-randomises and propagates the response, while also appending its own encrypted ID to it (r_2'). 8: A again blinds the IDs the in response ($\phi(r_2')$) and sends it to the STTP. The STTP determines if there is a match. Since in this case there is a match, the STTP decrypts the blinded IDs and sends them back to A ($\phi([ID])$). 9: A unblinds the list of IDs and informs the vertices that appear in the list that they are part of an entity cycle.

Before a message or response is propagated, the randomness in the encrypted values is refreshed. Likewise, the messages and responses are tracked by each vertex using constantly changing nonces, similar to the method used by Jense (2024) [34]. This ensures that linking messages is infeasible, preventing accounts, and, by extension, banks, from learning anything new about the topology of the transaction network. Additionally, the protocol includes blinding and uses variants of keys for each vertex. Both these techniques help prevent a number of attacks by malicious accounts attempting to learn about the identity of v_0. Figure 4.2 shows the different steps and the messages exchanged in a basic instance of the protocol.

4.3. Protocol Steps

In this section, each step of the protocol is presented and explained in some detail. It is again important to highlight that the protocol operates in a fully decentralised transaction network. Thus, in the protocol, each bank account or vertex is an individual agent, assumed to know nothing about the other vertices. In reality, each vertex is controlled by a bank, and each bank controls many vertices.

The protocol uses the following parameters: the length of all Bloom filters l, the number k of independent hash functions used to create the filters, the length in bits bl of the modulus for the Paillier cryptosystem, the minimum length ℓ_{min} of an entity cycle to be detected, and the maximum length ℓ of an entity cycle to be detected.

4.3.1. Setup

Before the protocol can be run, both the semi-trusted third party (STTP) and the vertices go through some preliminary steps to set up the protocol. First, the STTP is initiated by passing it the protocol parameters k, l and bl. Algorithm 1 shows this procedure. After storing k and l, the STTP generates a Paillier keypair of secure bit-length bl. The subroutine PAILLIER_KEYGEN is an implementation of the Paillier key generation process shown in Equations 2.7 to 2.12. Since it such an implementation is not novel, it is not shown here. Once the STTP has obtained a keypair, it publishes the public key such that all vertices can access it.

Algorithm 1 Setup of the STTP.

procedure SETUP_STTP(k, l, bl)
  store k, l
  $keys_{sttp} = \langle pk_{sttp}, sk_{sttp} \rangle = \langle \langle n, g \rangle, \langle \lambda, \mu \rangle \rangle \leftarrow$ PAILLIER_KEYGEN(bl)
  store $keys_{sttp}$
  publish pk_{sttp}
end procedure

Algorithm 2 Setup of a vertex.

procedure SETUP_VERTEX($k, l, \ell, \ell_{min}, pk_{sttp}, I$)
  store $k, l, \ell, \ell_{min}, pk_{sttp}$
  for *attribute* in I **do**
    initialise BF as a sequence of 0s of length l
    append and prepend '_' to *attribute*
    bigrams ← split *attribute* into bigrams
    for *bigram* in *bigrams* **do**
      for i from 0 to k **do**
        $h(x) \leftarrow \text{SHA1}(x) + i \cdot \text{MD5}(x) \mod l$
        $BF[h(bigram)] \leftarrow 1$
      end for
    end for
    store BF in FILTERS
  end for
end procedure

Secondly, the vertices are initialised. This is shown in Algorithm 2. The vertex is given protocol parameters k, l, ℓ and ℓ_{min}, as well as the previously published STTP public key pk_{sttp}. Input for the initialisation is also the bank account's identity I. This is a set of string attributes such as first name, last name, etc. The main objective in the initialisation process is to transform the identity from strings into Bloom filters. For each attribute, this is done by first creating an empty filter of length l, which is a sequence (or bitset) of zeroes. Then, the string is split into bigrams. It is important that before it is split, a start-stop character '_' is placed at the start and end of the string. This ensures that a change of one character in the string always affects two bigrams. To populate the filter, each bigram is hashed with k hash functions, which are created using the method shown in Equation 3.1. Each hash function outputs an index of the filter, which is then set to 1. The Bloom filter for each attribute is stored in a variable called FILTERS.

4.3.2. Initialisation of an Instance

An instance of the protocol is created when one vertex (v_0) initialises it. The goal of this instance is for v_0 to learn if any other vertices reachable in at most ℓ hops have an identity that matches with the identity of v_0. Initialisation by a vertex starts with the vertex requesting a custom public key from the STTP. The procedure run by the STTP to create this key is shown in Algorithm 4. The new keypair is based on keys previously generated by the STTP for itself ($keys_{sttp}$). To create a new keypair, the same modulus n is used but a different generator g_{v_0} is created. In the Paillier scheme, there are many possible generators for a given n, the only condition being that the generator divides the order of n. Since n remains the same, λ also remains the same. Since μ depends on g and λ, every secret key can be derived using one secret key and the other public key. This is not a problem in this protocol because no other party than the STTP ever knows the secret keys. Each initiating vertex thus only obtains the public key of its new keypair. Such a new key is required to prevent an attack from other vertices, which is further explained in Section 5.1.3. The reason for the new key using the same n as pk_{sttp} is that it allows all other vertices to re-randomise ciphertexts and do homomorphic operations (for which one only requires n), without allowing them to create encryptions themselves (which also requires g).

Algorithm 3 v_0 initialises an instance.

1: **procedure** INITIATE_INSTANCE
2: **store** $pk_{v_0} \leftarrow$ STTP_KEY_REQUEST()
3: **for** BF in FILTERS **do**
4: $E_{v_0}(BF) \leftarrow$ encrypt every bit in BF with pk_{v_0}
5: **end for**
6: let $E_{v_0}(I_{BF})$ be the list of all encrypted Bloom filter attributes $E_{v_0}(BF)$
7: $E_{v_0}(1) \leftarrow$ encrypt 1 with pk_{v_0}
8: **for** out-neighbour v_j in $N^+(v_0)$ **do**
9: $o_j \xleftarrow{R} \{0,1\}^{100}$
10: **add** $\langle id_j, o_j \rangle$ **to** INITIATED
11: **send** $message \leftarrow \langle o_j,\ E_{v_0}(I_{BF}),\ E_{v_0}(1),\ \ell - 1 \rangle$ **to** v_j
12: **end for**
13: **end procedure**

After v_0 has obtained its public key pk_{v_0} (Algorithm 3), it will encrypt its Bloom filter identity I_{BF}. Encrypting I_{BF} means that every Bloom filter will be encrypted individually, which means that every bit in the filter will be encrypted. $E_{v_0}(I_{BF})$ denotes such an encrypted identity. All encryption operations in the protocol are encryptions using the Pallier scheme as shown in Equations 2.13 to 2.15. Next, an encryption for the value 1 is created with the same key (line 7). Finally, messages are sent to all the out-neighbours of v_0. The vertex keeps track of the messages by including a random nonce o_j in each message. This nonce is stored in a set called INITIATED such that v_0 can later identify that a response it receives belongs to an instance it initiated itself. The message further contains the encrypted identity and the encrypted 1, which are needed to compute the distance between identities, and a time-to-live (ttl) counter, which is set to the maximum number of steps ℓ minus the one step that will be taken now.

Algorithm 4 STTP receives a key request from v_0.

1: **procedure** STTP_KEY_REQUEST
2: $n \leftarrow pk_{sttp}.n$
3: $\lambda \leftarrow pk_{sttp}.\lambda$
4: take random g_{v_0} from $\mathbb{Z}^*_{n^2}$ such that $\gcd(g_{v_0}, n) = 1$
5: $\mu_{v_0} \leftarrow (L(g^{\lambda}_{v_0} \mod n^2))^{-1} \mod n$
6: $keys_{v_0} = \langle pk_{v_0}, sk_{v_0} \rangle \leftarrow \langle \langle n, g_{v_0} \rangle, \langle \lambda, \mu_{v_0} \rangle \rangle$
7: **put** $\langle id_{v_0}, keys_{v_0} \rangle$ **in** KEYSTORE
8: **send** pk_{v_0} **to** v_0
9: **end procedure**

4.3.3. Handling a Message

The central step in the protocol is how an arbitrary vertex v_j handles a message it receives from one of its in-neighbours. This neighbour can be the vertex that initiated the protocol instance, or a vertex that simply propagated the message. Algorithm 5 shows the procedure executed by v_j once it receives a message. There are two tasks: creating a response to the message (lines 2-11) and propagating the message (lines 12-18).

Algorithm 5 v_j receives a message from v_i.

1: **procedure** HANDLE_MESSAGE($o_{i,j}$, $E_{v_0}(I_{BF})$, $\mathrm{E}_{v_0}(1)$, ttl)
2: **if** $ttl \leq \ell - \ell_{min}$ **then**
3: $E_{v_0}(D) \leftarrow$ COMPUTE_DISTANCES(FILTERS, $E_{v_0}(I_{BF})$, $E_{v_0}(1)$)
4: $E_{sttp}(id_{v_j}) \leftarrow$ ENCRYPT(id_{v_j}, pk_{sttp})
5: $\beta_j \xleftarrow{R} \mathbb{Z}_n \cap \mathbb{Z}^*_{n^2}$
6: link $E_{v_0}(D)$ and $E_{sttp}(id_{v_j})$ by multiplicatively blinding both with β_j
7: $E_{sttp}(\beta_j) \leftarrow$ ENCRYPT(β_j, pk_{sttp})
8: let *ids* be a list of encrypted IDs and add $E_{sttp}(id_{v_j})$ to it
9: let *blinds* be a list of encrypted blinding factors and add $E_{sttp}(\beta_j)$ to it
10: **send** $response = \langle o_{i,j}, E_{v_0}(d), ids, blinds \rangle$ **to** v_i
11: **end if**
12: **if** $ttl > 0$ **then**
13: $E'_{v_0}(I), E'_{v_0}(1) \leftarrow$ refresh the randomness on $E_{v_0}(I_{BF})$ and $E_{v_0}(1)$
14: **for** out-neighbour v_k in $N^+(v_j)$ **do**
15: $o_{j,k} \xleftarrow{R} \{0,1\}^{100}$
16: **put** $\langle \langle id_k, o_{j,k} \rangle, \langle id_i, o_{i,j} \rangle \rangle$ **in** ORIGIN
17: **send** $message' = \langle o_{j,k}, E'_{v_0}(I_{BF}), E'_{v_0}(1), ttl - 1 \rangle$ **to** v_k
18: **end for**
19: **end if**
20: **end procedure**

If v_j is at least ℓ_{min} steps removed from the initiating vertex, it will respond to the message. The most important part of the response is the list of encrypted distances $E_{v_0}(D)$ between the identity attributes of v_0 and the attributes of v_j. Algorithm 6 shows how the distances are computed homomorphically from the encrypted Bloom filters of v_0 and the plain filters of v_j. This algorithm is an implementation of the method described in Equations 3.3 to 3.8. In the algorithm, encryptions of the three intermediate variables a, b and c are created by homomorphically adding the values in the encrypted Bloom filter. See Section 3.5 for details. The functions PAILLIER_ADD and PAILLIER_SUBSTRACT are trivial implementations of homomorphic addition in the Paillier cryptosystem as described in Equation 2.18. The distance metric that is used in the protocol is the Hamming distance (line 16), because it can be computed homomorphically. One could also use the Jaccard distance, but since this requires a division, all three intermediate variables would have to be included in the response such that the STTP can compute the distance after decryption.

Algorithm 6 Computation of distances between an encrypted and unencrypted Bloom filter identity.

1: **procedure** COMPUTE_DISTANCES($I_{BF}, E_{v_0}(I_{BF}), E_{v_0}(1)$)
2: let $E_{v_0}(D)$ be the list of encrypted distances
3: **for** index i in I_{BF} **do**
4: $E_{v_0}(a) \leftarrow$ PAILLIER_SUBSTRACT($E_{v_0}(1), E_{v_0}(1)$) ▷ Initialise a as 0
5: $E_{v_0}(b) \leftarrow E_{v_0}(a)$ ▷ Initialise b as 0
6: $E_{v_0}(c) \leftarrow E_{v_0}(a)$ ▷ Initialise c as 0
7: **for** index j in $I_{BF}[i]$ **do**
8: **if** $I_{BF}[i] == 1$ **then**
9: $E_{v_0}(a) \leftarrow$ PAILLIER_ADD($E_{v_0}(a), E_{v_0}(I_{BF})[i][j]$)
10: $E_{v_0}(c) \leftarrow$ PAILLIER_ADD($E_{v_0}(c), E_{v_0}(1)$)
11: **else**
12: $E_{v_0}(b) \leftarrow$ PAILLIER_ADD($E_{v_0}(b), E_{v_0}(I_{BF})[i][j]$)
13: **end if**
14: **end for**
15: $E_{v_0}(c) \leftarrow$ PAILLIER_SUBSTRACT($E_{v_0}(c), E_{v_0}(a)$)
16: $E_{v_0}(D)[i] \leftarrow$ PAILLIER_ADD($E_{v_0}(b), E_{v_0}(c)$) ▷ Hamming distance
17: **end for**
18: **return** $E_{v_0}(D)$
19: **end procedure**

Continuing in Algorithm 5, after the distances are computed, the vertex encrypts its ID with the public key of the STTP (line 4). Then, this encrypted ID is linked to the encrypted distances by blinding both with the same blinding factor (lines 5-6), which involves multiplying the encrypted ID with a blinding factor and multiplying every encrypted distance in $E_{v_0}(D)$ with the same factor. This is done as to prevent an attack by v_0 on the encrypted ID, which is further explained in Section 5.1.3. The blinding factor β_j is picked randomly from the intersection of the plaintext space and the ciphertext space, such that it can both be applied to ciphertexts and can be encrypted itself. When a ciphertext is multiplied with β_j it is no longer valid. Only when the factor is removed can it be decrypted again. For this reason, v_j encrypts β_j with the public key of the STTP and includes it in the response. Finally, the response is sent to the neighbour that sent the message. It consists of the encrypted distances, a list with (for now) only the encrypted ID of v_j, and a list of only the encrypted blinding factor. Also, it includes the same nonce as was included in the message.

The second step of handling a message is to propagate it if the time-to-live has not yet reached 0. Lines 12-18 in Algorithm 5, show how this is done. Most importantly, the randomness on the encryptions in the message is refreshed. This is done by multiplying them with a new random number, as explained in Equations 2.20 and 2.21. Note that no full public key is required, only the modulus n, which is shared among all keys. This makes it possible for v_j to re-randomise encryptions made with pk_{v_0}, despite it not knowing this key. Finally, the message is propagated to every out-neighbour of V_j (lines 13-17). A new random nonce $o_{j,k}$ is used for every neighbour. Each nonce is coupled to the nonce in the incoming message ($o_{i,j}$) and saved in a dictionary called ORIGIN.

4.3.4. Propagating a Response

When a vertex v_j receives a response from one of its out-neighbours, two things can happen: if the response belongs to an instance initiated by this vertex, then v_j must process it, otherwise it must be propagated. Algorithm 7 shows the procedure for handling a response. The vertex has a set of nonces used in messages it initiated called INITIATED. If the nonce $o_{j,k}$ included in the response is in this set, v_j knows it must process the response and calls PROCESS_RESPONSE, which is explained in the next section.

If the nonce is not in the set INITIATED, then the response is propagated. The response consists of three core parts: $E_{v_0}(D)$ is the list of encrypted distances blinded by both the vertex that created it and all the vertices that have forwarded it. Secondly, *ids* is the list of encrypted identifiers of all the vertices that have handled the response, each blinded by a blinding factor created by each vertex. Similarly, *blinds* is the list of all these encrypted blinding factors.

Algorithm 7 v_j receives a response from v_k.

1: **procedure** HANDLE_RESPONSE($o_{j,k}$, $E_{v_0}(D)$, *ids*, *blinds*)
2: **if** INITIATED contains $\langle id_{v_k}, o_{j.k} \rangle$ **then**
3: **call** PROCESS_RESPONSE($E_{v_0}(D)$, *ids*, *blinds*)
4: **else**
5: $E'_{v_0}(D), ids', blinds' \leftarrow$ refresh the randomness of $E_{v_0}(D)$, *ids* and *blinds*
6: $E_{sttp}(id_j) \leftarrow$ ENCRYPT(id_j, pk_{sttp})
7: $\beta_j \xleftarrow{R} \mathbb{Z}_n \cap \mathbb{Z}^*_{n^2}$
8: link $E_{v_0}(D)$ and $E_{sttp}(id_{v_j})$ by multiplicatively blinding both with β_j
9: $E_{sttp}(\beta_j) \leftarrow$ ENCRYPT(β_j, pk_{sttp})
10: **append** $E_{sttp}(id_j)$ **to** *ids'*
11: **append** $E_{sttp}(\beta_j)$ **to** *blinds'*
12: **fetch** $\langle\langle id_k, o_{j,k} \rangle, \langle id_i, o_{i,j} \rangle\rangle$ **from** ORIGIN
13: **send** $\langle o_{i,j}, E'_{v_0}(D), ids', blinds' \rangle$ **to** v_i
14: **end if**
15: **end procedure**

For the response to be propagated, first, the randomness on each encryption in the response must be refreshed. It does not matter that some of the encryptions are blinded: re-randomising an invalid ciphertext will still yield a valid ciphertext once the blinding factor is removed. Then, v_j encrypts its own ID and links it to the encrypted distances by adding a new blinding-factor to the distances and to the encrypted ID (lines 6-8). These are the same steps as executed by the vertex that created the response. Next, the new blinding factor β_j is encrypted with the public key of the STTP (9). The vertex adds its encrypted ID to *ids'* and its encrypted blinding factor to *blinds'* (10-11). Then, the vertex must find out which neighbour to forward the response to. This must be the neighbour that it previously received the message from, which it then forwarded to the neighbour which is now sending the response. v_j has remembered this information in the dictionary ORIGIN. It can find the sender of the original message and the nonce it used by looking up the entry with the key consisting of v_k and the nonce $o_{j,k}$ belonging to this instance (12). Finally, the vertex propagates the messages to v_k, which now includes the re-randomised and re-linked encrypted distances, the list of blinded encrypted IDs, extended with its own blinded ID, and the list of encrypted blinding factors, extended with its own blinding factor.

4.3.5. Processing a Response

When a vertex (v_0) found that has received a response to a message that it initiated, it starts the process of finding out whether there is a match or not. This involves sending the response to the STTP for decryption. However, to prevent the STTP from learning the list of IDs, each ID is first again blinded by v_0. Algorithm 8 shows the steps taken to process such a response.

Algorithm 8 v_0 processes a response

1: **procedure** PROCESS_RESPONSE($E_{v_0}(D)$, *ids*, *blinds*)
2: let ids_γ be the list of blinded IDs
3: let *inner-blinds* be the list of inner-blinding-factors
4: **for** $E_{sttp}(id)$ in *ids* **do**
5: $\gamma \xleftarrow{R} \mathbb{Z}_n$
6: $E_{sttp}(id_\gamma) \leftarrow$ PAILLIER_ADD($E_{sttp}(id), \gamma$)
7: **append** $E_{sttp}(id_\gamma)$ **to** ids_γ
8: **append** γ **to** *inner-blinds*
9: **end for**
10: $o_{sttp} \xleftarrow{R} \{0,1\}^{100}$
11: **put** $\langle o_{sttp}, inner\text{-}blinds \rangle$ **in** STTP_BLINDS
12: **send** $\langle o_{sttp}, E_{v_0}(D), ids_\gamma, blinds \rangle$ **to** *STTP*
13: **end procedure**

First, each ID in the list is blinded by adding a value γ (lines 4-8). While each ID is already blinded by β_i, this factor is multiplied directly with the encryption and sits 'outside' the encrypted value. The new factor γ is added homomorphically to the plaintext ID, so it is placed inside the encryption and remains after the ID has been decrypted. Next, v_0 takes a random nonce which is used as a key to store the list of blinds *inner-blinds* in a dictionary called STTP_BLINDS (11-12). Finally, the encrypted distances, encrypted and dually blinded IDs, and the encrypted blinds β are sent to the STTP for decryption.

When the STTP receives a response to decrypt (Algorithm 9), it will first decrypt all outer-blinds β and compute their inverses in $\mathbb{Z}^*_{n^2}$. Then, it will look up the keypair for v_0 that it previously stored in KEYSTORE. Next, for every attribute in $E_{v_0}(D)$, it will first remove all the outer-blinds β from the ciphertext by multiplying with the inverted blinds β^{-1}. This makes $E_{v_0}(d)$ a valid ciphertext again. It will then decrypt each ciphertext with the key of v_0 to obtain a list of plaintext distances between the attributes *dists*.

Algorithm 9 STTP decrypts a response received from v_0

1: **procedure** DECRYPT_RESPONSE($o_{sttp}, E_{v_0}(D), ids_\gamma, blinds$)
2: *inverse-blinds* ← decrypt all blinding-factors in *blinds* with sk_{sttp} and invert them in $\mathbb{Z}^*_{n^2}$
3: **fetch** $\langle id_0, keys_{v_0} \rangle$ **from** KEYSTORE
4: let $dists$ be the list of all decrypted distances
5: **for** blinded encrypted distance $E_{v_0}(d)$ in $E_{v_0}(D)$ **do**
6: unblind $E_{v_0}(d)$ by multiplying it with every β^{-1} in *inverse-blinds*
7: $d \leftarrow$ PAILLIER_DECRYPT($E_{v_0}(d), keys_{v_0}$)
8: **append** d to *dists*
9: **end for**
10: *match* ← DECIDE_MATCH(*dists*)
11: **if** *match* **then**
12: **for** id_γ in ids_γ **do**
13: remove the outer-blind β by multiplying with the corresponding β^{-1} from *inverse-blinds*
14: decrypt with sk_{sttp}
15: **end for**
16: **send** $\langle o_{sttp}, True, ids_\gamma \rangle$ **to** v_0
17: **else**
18: **send** $\langle o_{sttp}, False, \emptyset \rangle$ **to** v_0
19: **end if**
20: **end procedure**

Using the distances, the STTP will decide if there is match between the identity of v_0 and the identity of the vertex that created the response (10). The subroutine DECIDE_MATCH should output True or False based on the distances using a rule-based deterministic entity resolution matching method. The exact rules that are used are highly dependent on the attributes that are included in the identity, and the domain, context and manner in which such attributes obtained, stored and processed. Suitable thresholds for individual attribute distances can only be found once the details of the identity have been clearly specified and the nuances of the attributes have been studied empirically. For this reason, an example of an implementation for DECIDE_MATCH is omitted here.

Finally, when the STTP has found that there is a match, it will continue to decrypt the IDs. First, it will remove the outer-blinds β from the ciphertexts (13). Note that the whole process of removing blinds is only successful if both the distances and the IDs were blinded with the same blinding factors, i.e. linked. After that, each ID is decrypted. The STTP cannot learn anything about the actual IDs because they are still blinded by γ. Then, the STTP informs v_0 that there is a match and sends the list of IDs (16). If there is no match, the STTP simply informs v_0 that the matching is negative (18).

Algorithm 10 v_0 receives a result

```
1:  procedure PROCESS_RESULT(o_sttp, match, ids_γ)
2:      if match then
3:          fetch ⟨o_sttp, inner-blinds⟩ from STTP_BLINDS
4:          ids ← []
5:          for index i in ids_γ do
6:              ids[i] ← ids_γ[i] – inner-blinds[i]
7:          end for
8:          inform vertex with the ID in ids[0] that it has a matching identity
9:          inform the other vertices in ids that they are part of an entity cycle
10:     end if
11: end procedure
```

Lastly, Algorithm 10 shows how v_0 processes a result. If there is no match, v_0 simply does nothing. However, if there is a result, v_0 needs to learn the list of IDs. The first ID in this list is the vertex with whom the identity of v_0 matches. The other IDs are the vertices that are in the path to this matched vertex. v_0 first retrieves the blinds it stored in STTP_BLINDS (line 3). Then, for each ID in the list, it subtracts the corresponding blinding factor to obtain the actual ID (6). Finally, it can inform all the vertices that they are potentially involved in money laundering, so the banks can proceed with further investigation (8-9). Since v_0 knows the actual IDs of these vertices, they can be contacted directly, without having to traverse the transaction graph.

5 Analyses

This chapter contains two analyses of the protocol. Section 5.1 contains a security analysis and Section 5.2 contains a complexity analysis.

5.1. Security Analysis

In order to make claims about the security of the protocol, a leakage analysis is performed on the three main ways an adversary can try to learn more information about the other parties than it should. Firstly, it is shown that the messages used in the protocol are unlinkable, making it infeasible for a semi-honest adversary to learn about the topology of the network. Second, it is explained that the protocol does not allow colluding vertices to learn anything more than they collectively already know. Lastly, it is shown that the protocol is secure against some attacks on the identities of other vertices by a covert adversary.

In all cases, it is assumed that secure communication channels are in place so that eavesdropping is not possible. Likewise, it is assumed internal computations, memory and storage of honest parties are hidden. Moreover, all adversaries are polynomially bounded, making it infeasible for them to learn anything by solving the factorisation or discrete logarithm problems. For this analysis, only the contents of the protocol itself are considered and analysis side-channel attacks is omitted.

5.1.1. Semi-Honest Adversaries

The main threat to the security of the protocol is the presence of semi-honest adversaries. It must be assumed that such adversaries are widespread among the participants in the protocol, because there is no formally established structure of trust between them. The participants in the protocol are bank accounts, which outside of the decentralised model are each controlled by a bank. The banks are private and commercial organisations that not only have no reason to trust each other when it comes to respecting the confidentiality of their data, but in fact have significant reason to distrust each other because they are competitors in a commercial setting. This means that there is financial incentive for a bank to learn about the others' data.

Let $\mathcal{A}_S$ be a polynomially bounded semi-honest adversary that participates in the protocol as a bank account (vertex). The goal of this adversary is to learn more about the other bank accounts' data and transactions than it knows at the start of the protocol. For the protocol not to leak any data, the adversary must not learn anything else than what is revealed to it by the inputs and outputs in correct operation of the protocol.

The first approach $\mathcal{A}_S$ can use to learn about the other vertices data is to infer information from the inputs it receives and the outputs in computes. A message contains the identity of the initiating vertex v_0, or more precisely Bloom filters of each of the attributes of the identity. Since Bloom filters themselves provide an inadequate level of confidentiality [40] [66], v_0 has encrypted the filters with the Paillier encryption scheme using the public key pk_{v_0}. This involves treating each bit in the filter as a plaintext to be encrypted into a ciphertext. Thus, the encrypted identity can be seen as a collection of individually encrypted ones and zeros.

Since the Paillier scheme is IND-CPA secure, an adversary has only a negligible advantage compared to random guessing in distinguishing between two ciphertexts when it has access to the encryption key. This means that is $\mathcal{A}_S$ cannot do better than randomly guess if an encrypted bit in a filter is a one or a zero, meaning it cannot infer any information about the original data. The same holds for all other values that are computed by $\mathcal{A}_S$ or that it receives in a response: each distance d, each vertex ID id_j and each blinding factor β_j is encrypted using the same encryption scheme with either pk_{sttp} or pk_{v_0}. Because the Paillier scheme is IND-CPA secure, $\mathcal{A}_S$ cannot infer anything significant about the content from the ciphertexts. This all, of course, is with the assumption that the modulus n used for the Paillier keys is at least 2048 bits long, making it infeasible for $\mathcal{A}_S$ to find the factors p and q [9].

The second approach $\mathcal{A}_S$ can take is to attempt to infer information about the topology of the transaction network. That is, to determine the existence of transactions between other bank accounts. This becomes possible when messages in the protocol are linkable to the same instance. If an adversarial vertex receives from different neighbours two messages that are linkable, it can infer with high probability the existence of edges to and from those neighbours. Figure 5.1 shows two examples of how linkability makes its possible for $\mathcal{A}_S$ to infer the existence of edges. In both examples, $\mathcal{A}_S$ receives two messages m_a and m_b. If it can link the messages, i.e. to conclude they belong to the same instance, it can with high probability guess the existence of edges to v_i and v_j. In the left example, the messages have the same $ttl = 3$ and ℓ is 5. Then, $\mathcal{A}_S$ is certain that either v_0 has edges to both v_i and v_j (as pictured), or v_i and v_j have an edge between each other (though it might be unknown who v_0 is). In the right example, the messages have $ttl_a = 4$ and $ttl_b = 3$. Given that ℓ is 5, $\mathcal{A}_S$ now learns with certainty that edge (v_0, v_i) exists.

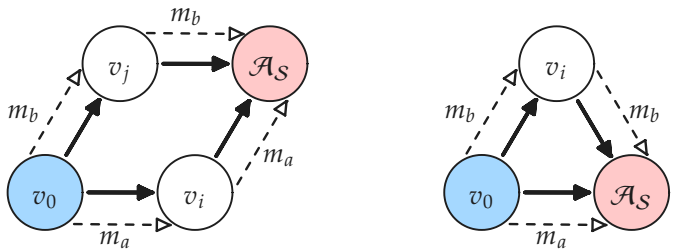


Figure 5.1: Two examples of how linkability exposes network topology to an adversary. Left: if m_a and m_b are linkable, $\mathcal{A}_S$ learns the existence of a path from v_0 to both v_i and v_j. Right: if m_a and m_b are linkable, $\mathcal{A}_S$ learns the existence of a path from v_0 to v_i. Using the messages' ttl value and the ℓ value, $\mathcal{A}_S$ might also be able to determine the existence of specific edges.

For the protocol to not leak any information, it must be infeasible for $\mathcal{A}_S$ to learn about the topology by linking messages. Messages are unlinkable if the adversary does not have a significant advantage in guessing whether two messages belong to the same instance. Messages and responses in the protocol consist of three types of data: values encrypted with the Paillier cryptosystem, nonces, and a TTL value. When a message (or a response) is forwarded by a vertex, it will refresh the randomness on every encryption. Since Paillier is IND-CPA secure, $\mathcal{A}_S$ does not gain a significant advantage in guessing a link between messages by comparing such encrypted values. Similarly, the nonce in the message is replaced with a new random value every time a message is forwarded by a vertex and a different nonce is used for every neighbour. This means that if $\mathcal{A}_S$ receives a message belonging to the same instance from two different neighbours, the nonces will be different, making it impossible to for the adversary to use the nonces to infer information about the instance.

While $\mathcal{A}_S$ cannot use the encryptions or the nonces for linking messages, two other aspects of the messages and responses do reveal some information about the instance. Every message contains a ttl attribute that serves as a counter for the number of vertices that the message has passed through. Similarly, each response contains two lists containing the encrypted IDs and the encrypted blinds. The number of items in these lists (which is always equal), reveals essentially the same thing as the ttl in the messages: it serves as a counter for the number of vertices the response has traversed. $\mathcal{A}_S$ can use this information to distinguish between some messages. For example, say it receives two messages from one neighbour, both with $ttl = \ell - 2$, then it can be certain that the messages belong to different instances, since the neighbour is the only vertex between $\mathcal{A}_S$ and v_0, and a v_0 does not send the same message to the same vertex twice, there must be two different v_0 and different instances.

Inferring that two messages do belong the same instance based on *ttl* or length of a list is significantly more difficult. This is because the number of possible values is extremely low, typically only six [32]. Given that many instances of the protocol are running at the same time, $\mathcal{A}_S$ will receive many messages from different instances with the same or close *ttl* values. This makes it infeasible for $\mathcal{A}_S$ to use this value to effectively link messages.

Lastly, there is the case of the third party being an adversary. In fact, since the third party is completely independent of any of the other parties in the protocol, it cannot be trusted to be honest, and it must always be assumed that it is at least a semi-honest adversary. For this reason, the third party is always referred to as the 'semi-honest third party' (STTP). In the protocol, the STTP communicates only with v_0 and receives only one kind of message: a matching request. This contains the encrypted and blinded distances between v_0 and another vertex's identity, the blinded and encrypted IDs and the encrypted blinding factors. The STTP can decrypt all of these. It removes the blinding factors from the ciphertexts of the distances and the IDs. However, it learns no useful information from these values. While it does find the distances, it cannot use them to infer any information about either of the vertices' identities. The only thing it learns from this is whether there is a match or not. However, since there are many matching requests and the STTP does not know who the neighbours of v_0 are, nothing about the identities or the network topology can be learned from this. Secondly, the STTP can decrypt the list of IDs. However, it does not learn the IDs, because v_0 has homomorphically added a different blinding factor to each ID.

There is one aspect of the protocol which does allow the STTP to learn significant information. The STTP can count the number of requests it receives from each v_0, and how many of these are matches. This allows it to infer information about the number of neighbours of v_0 and the average number of neighbours in the network. Specifically, it reveals the number of neighbours reachable from v_0 in ℓ_{min} to ℓ hops. However, this type of leakage can be easily prevented by v_0 by randomly sending some artificial matching requests to the STTP. This will prevent it from obtaining an accurate count of the number of vertices reachable from v_0.

In summary, nothing in the protocol's messages or responses gives $\mathcal{A}_S$ a significant advantage in linking messages to learn about the network's topology, and $\mathcal{A}_S$ can infer nothing significant about the content of the messages. The STTP cannot learn anything significant from a single matching request. While information is leaked to the STTP by the number of matching requests made, initiating vertices can easily prevent this by sending random artificial matching requests. Therefore, the protocol does not leak any significant information in the semi-honest security model.

5.1.2. Collusion

Collusion is the cooperation of multiple adversaries to collectively achieve more than only a single adversary would be able to achieve. In the setting of collaborative AML, collusion between adversarial vertices is expected, because such a vertex is controlled by a bank, which likely controls more vertices. Outside of the fully decentralised model, the adversaries are not individual bank accounts, but rather entire banks, which control many accounts. So, for the protocol not to leak information in the semi-honest model, it must not leak anything significant in the presence of colluding semi-honest adversaries.

In the case where multiple vertices collude, they can uncover no more information than they would already be able to uncover outside of the protocol. Let $\mathcal{A}_{S1}$ and $\mathcal{A}_{S2}$ be two arbitrary semi-honest adversarial vertices. When they are colluding, everything that is known by $\mathcal{A}_{S1}$ is also known by $\mathcal{A}_{S2}$ and vice versa. Say $\mathcal{A}_{S1}$ receives a message, then $\mathcal{A}_{S2}$ also obtains this exact message. Firstly, the adversaries gain no extra advantage in inferring information about the content of the message. Since all the values are encrypted with the Paillier cryptosystem, the confidentiality of the content is protected and this protection is independent of how many parties learn the ciphertext: only the STTP can decrypt. Secondly, they are not able to learn anything new about the network that they could also learn outside of the protocol. Say, for example, that $\mathcal{A}_{S1}$ receives message m_1 from neighbour v_i and $\mathcal{A}_{S2}$ receives message m_2 belonging to the same instance from v_i. $\mathcal{A}_{S2}$ also knows m_1 because of the collusion, and can to link m_1 and m_2 because the ciphertexts are the same (v_i only re-randomises encryptions once for forwarding a message to all neighbours). This allows $\mathcal{A}_{S2}$ to learn about the existence of the edge $(v_m, \mathcal{A}_{S1})$. However, this is not new information to $\mathcal{A}_{S2}$, because both adversaries already know each other's neighbours. Since collusion between $\mathcal{A}_{S1}$ and $\mathcal{A}_{S2}$ does not allow them to learn anything new, the protocol does not leak any information in the presence of such colluding vertices.

However, there is a case where the protocol can leak some information. If the STTP colludes with one or more vertices, both parties can learn information that should not be revealed to them. Firstly, say $\mathcal{A}_{\mathcal{S}_{sttp}}$ is an adversarial STTP that colludes with an adversarial vertex $\mathcal{A}_{\mathcal{S}0}$, which initiated an instance of the protocol (i.e. $\mathcal{A}_{\mathcal{S}0}$ is v_0). Normally, the IDs that are sent to the STTP are blinded by v_0. However, in a scenario with collusion, $\mathcal{A}_{\mathcal{S}_{sttp}}$ knows the blinding factors created by $\mathcal{A}_{\mathcal{S}0}$. $\mathcal{A}_{\mathcal{S}_{sttp}}$ thus learns the list of IDs and thereby learns about the topology of the network. Since the parties are colluding, $\mathcal{A}_{\mathcal{S}0}$ learns this too. Furthermore, $\mathcal{A}_{\mathcal{S}0}$ knows the distances decrypted by $\mathcal{A}_{\mathcal{S}_{sttp}}$. This allows $\mathcal{A}_{\mathcal{S}0}$ to infer a lot of information about the identities of the other vertices by comparing the distances to its own identity. Furthermore, if $\mathcal{A}_{\mathcal{S}_{sttp}}$ is colluding with an arbitrary vertex $\mathcal{A}_{\mathcal{S}_i}$ that is involved in an instance initiated by v_0, then $\mathcal{A}_{\mathcal{S}_i}$ learns the distance between its own identity and the identity of v_0. $\mathcal{A}_{\mathcal{S}_i}$ can compare the distance to its own identity to infer a lot about the identity of v_0.

In conclusion, colluding vertices can discover nothing more than they already collectively know. However, in a scenario where the STTP colludes with a vertex, both adversaries are able to learn about the identity of other vertices and about the topology of the network. Therefore, the protocol leaks no significant information in the semi-honest model, with the condition that the STTP does not collude.

5.1.3. Attacks by Covert Adversaries

While it is important for any application to not leak information in the semi-honest model, it often does not reflect true security, because in many practical settings the adversaries are not only semi-honest but also covert. Where semi-honest adversaries follow the protocol honestly and only try to learn from the data they receive, covert adversaries go a step further and also engage in malicious interactions, as long as such malicious activities are guaranteed to go unnoticed by the other parties. For the privacy preserving entity cycle detection protocol to not leak information in a practical setting, it must be able to withstand a number of attacks by such adversaries that are likely to occur. The attacks covered here are expected in practical setting, because they are easy for the adversaries to execute, undetectable by other parties, and the adversaries can gain very valuable information if they succeed.

In the protocol, the STTP serves as a decryption oracle: it decrypts the distances, informs v_0 of a match, and decrypts the list of encrypted IDs if a match is found. In correct operation, the STTP should only reveal such information to the vertex that initiated the protocol instance (v_0). However, because the STTP does not keep any state about the instances of the protocol, all vertices can make an arbitrary number of calls to it. Covert adversaries might try to exploit this construction to trick the STTP into revealing information that such adversaries should not be privy to. Two different cases can be identified.

Firstly, let $\mathcal{A}_{\mathcal{C}i}$ be a polynomially bounded covert adversary that is not the initiating vertex. During normal operation of the protocol, $\mathcal{A}_{\mathcal{C}i}$ homomorphically computes the distances between its own identity and v_0's identity. Even though $\mathcal{A}_{\mathcal{C}i}$ is not the initiating vertex, it might still try to request the STTP to reveal if the distances correspond to a match or not. While it would not be a problem for $\mathcal{A}_{\mathcal{C}i}$ to learn if it matches with v_0 or not, such a thing being possible would open the door for an attack to infer the identity of v_0. Namely, $\mathcal{A}_{\mathcal{C}i}$ can also take an arbitrary identity, compute the distances, and ask the STTP if this identity matches. Since any vertex can make an arbitrary number of such calls to the STTP, $\mathcal{A}_{\mathcal{C}i}$ can repeat this process for many identities and that way execute a dictionary attack.

In the protocol, such an attack is prevented by making it impossible for a vertex to ask the STTP if distances match or not if that vertex is not the initiating vertex. This is achieved by using different encryption keys for every instance of the protocol. When v_0 starts an instance, it requests a new public key from the STTP, who generates a keypair and saves the secret key. v_0 encrypts its identity with this new public key. The other vertices cannot know this key, as it would serve as a tool to uniquely identify v_0 and the instance, compromising the privacy of v_0 and the network. The distances in the responses will also be encryptions of v_0's unique key. When the initiating vertex then asks the STTP to decrypt, it will do so using the secret key corresponding to v_0. This prevents the previously described attack, because if $\mathcal{A}_{\mathcal{C}i}$ asks the STTP to decide a match involving an instance that was started by v_0, it will decrypt the distances with the secret key of $\mathcal{A}_{\mathcal{C}i}$ and not the correct key of v_0. This would yield a useless corrupted plaintext and the STTP would discover the adversary's malicious action.

The only caveat in this setup is that for the other vertices to be able to homomorphically compute the distances and refresh the randomness on encryptions, they need the public key used to create the ciphertexts. While strictly speaking this is true, there is a way to get around it. Firstly, to compute the distances, each vertex needs to be able to encrypt with v_0's key. However, the value that needs to be encrypted is the number of bits in their Bloom filter which is set to 1. This will always be at most the length of the filter l. Therefore, creating such an encryption can also be cheaply achieved homomorphically by repeated addition of an encrypted 1. This is the reason that an encrypted 1 is included in the message created by v_0. Secondly, all vertices need to be able to do homomorphic operations on the ciphertexts. This does not require the full public key, but only the modulus n. The protocol solves this by using the same modulus for every public key. This is possible in the Paillier scheme, because a public key consists of n and a generator g. Given one n, there are many possible generators that can be chosen, with the only condition being that n divides the order of g. Thus, when creating a new keypair for v_0, the STTP reuses the modulus from its own keypair and simply picks a new random generator. In other applications, this would not be secure, because the secret key λ stays the same (dependent on only n). However, in the protocol, v_0 is not given the secret key. Only the STTP knows it. Thus, by using the modulus from the STTP's key, all vertices can do homomorphic operations on all encryptions, without knowing the full encryption key.

The second attack that is likely to occur in the covert model is an adversary trying to learn about the network's topology by exploiting the STTP. Let $\mathcal{A}_{C0}$ be a covert adversary that is the initiator of an instance. When $\mathcal{A}_{C0}$ receives a response, it can ask the STTP whether the response is a match. If it is, the STTP decrypts the list of IDs and reveals them to the vertex. If there is no match, the STTP does not reveal the IDs, because parties may only learn these if there is evidence of money laundering (i.e. a match). $\mathcal{A}_{C0}$ can try to trick the STTP into revealing any list of IDs by faking a match: instead of sending the real encrypted distances (that were included in the response), it can easily create fake encrypted distances that are guaranteed to be a match, e.g. by directly generating such distances and encrypting them. Then, when $\mathcal{A}_{C0}$ receives a response, it simply swaps out the real distances in the response for the fake distances and sends the response to the STTP, who will always conclude that there is a match and reveal the list of IDs. This attack would allow $\mathcal{A}_{C0}$ to learn the list of IDs for all responses it receives, which are all the paths to all the vertices in the neighbourhood. So, if this attack would be possible, any covert adversary can learn a lot about the network's topology.

This attack is prevented in the protocol by the linking of the IDs to the distances by the other vertices. Before each vertex adds their encrypted ID to the list, they multiply it with a blinding-factor β. Then, they multiply the distances with β also. Since this is not a homomorphic operation, it makes the ciphertexts invalid (although randomness can still be correctly refreshed on them). The process is repeated by every vertex that adds an ID to the list: at the end, each distance is multiplied with all the factors β. These ciphertexts of the distances and the IDs can only be made valid again if these factors are removed. Therefore, each vertex includes its factor β in the response, encrypted by pk_{sttp}. This allows the STTP to remove the factors by decrypting them, inverting them (in $\mathbb{Z}^*_{n^2}$) and multiplying these inverses with the ciphertexts. If the factors are correctly removed, the ciphertexts become valid again and can be decrypted. This mechanism prevents $\mathcal{A}_{C0}$ from swapping the distances for its fake distances in the response. The reason for this is that the STTP removes the same factors from the distances as from the IDs. If the distances are swapped, they will not have the same factors in them any more, leading to corrupted ciphertexts when the STTP multiplies with the inverse factors. $\mathcal{A}_{C0}$ cannot apply these factors itself because it does not know them; only encryptions of the factors are included in the response, and they cannot be added homomorphically. Thus, if the distances are swapped, their decryption will fail and the STTP knows something is wrong.

In conclusion, the protocol is secure against information leakage in the semi-honest model, given that the STTP does not collude, and is resistant against two attacks that would otherwise be likely to be attempted by a covert adversary.

5.2. Complexity Analysis

The worst-case complexity of the protocol can be determined by analysing every step. The greatest complexity out of all steps is the total worst-case complexity of the entire protocol. There are three different aspects of which the complexity can be determined: the computational (runtime) complexity O_R, the communication complexity O_C, and the storage complexity O_S. Below, V denotes the number of vertices in the network, N_{max} denotes the maximum number of out-neighbours a vertex can have, and ℓ denotes the maximum length of a path from v_0 to another vertex, i.e. the number of hops after which the *ttl* value in the protocol is 0.

The first step, the setup (Algorithms 1 and 2) are only run once and therefore have limited practical impact on the runtime. The main process in setting up the STTP is the generation of a Paillier key, wherefor the complexity depends on the required bit-length of the key *bl*. However, since in any practical setting *bl* remains fixed at a secure length, all Paillier operations are treated as constant time operations K in this analysis. This makes the runtime complexity for setting up the STTP $O_R(K)$. Since one key has to be stored, the storage complexity is $O_S(1)$.

Setting up each vertex (Algorithm 2) is a little more complex. Each vertex has to generate k independent hash functions, and use these to compute Bloom filters of length l for $|I|$ attributes of length $|I_i|$. So, while strictly speaking the complexity depends on l, k, $|I|$, and $|I_i|$, in practice all the values are fixed, so again the complexity can be denoted as constant K. However, the step needs to be reaped for every vertex, making the computational complexity $O_R(V)$. Storage is also constant per vertex in this step: $O_S(V)$ in total. Finally, since each vertex has to retrieve pk_{sttp} from the STTP, total communication is $O_C(V)$.

Furthermore, the initialisation of an instance by v_0 (Algorithm 3) consists of two steps. First, the STTP generates a keypair, which is a process with fixed parameters, so the operations are constant. Secondly, v_0 encrypts every bit in every Bloom filter. Since the number of filters and their length is constant throughout the protocol, these operations take roughly a constant amount of time. Then, it has to generate a nonce for every neighbour and store it. This makes the runtime complexity $O_R(N_{max})$ and the storage complexity $O_S(N_{max})$. Then, each neighbour has to be sent a message. Since the message is constant in size, the communication complexity is also $O_C(N_{max})$.

The next step is the handling of a message (Algorithm 5). This consists of two parts: computing and sending a response, and forwarding the message. Firstly, computing the response again depends on the number of Bloom filters and their length, which, since this is fixed, results in constant time operations, i.e. $O_T(K)$. Also, the response is of constant length and only has to be sent to one neighbour, making the communication $O_C(K)$. Next, propagating the response requires picking a nonce for all out-neighbours, storing it, and sending them a constant length message. This gives the same complexity in all three aspects: $O_T(N_{max})$, $O_S(N_{max})$, and $O_C(N_{max})$. However, note that in the worst case, these two steps are executed by all neighbours of v_0, of which there are N_{max}. Furthermore, all neighbours of their neighbours will also execute it, etc. This goes on until the *ttl* on the message is 0. The first step (computing and sending response) is always executed, meaning that it will be executed by $N_{max} + N_{max}^2 + N_{max}^3 + ... + N_{max}^\ell$. Since this first step was constant, the total complexity for the entire instance is $O(N_{max}^\ell \cdot K) = O(N_{max}^\ell)$. The second step is executed by all neighbours, except the last layer (when $ttl = 0$). This means that $N_{max} + N_{max}^2 + N_{max}^3 + ... + N_{max}^{\ell-1}$ will execute it. Since the complexity for this step is $O(N_{max})$, the total complexity for the entire instance is $O(N_{max}^{\ell-1} \cdot N_{max}) = O(N_{max}^\ell)$.

The next phase of the protocol is the propagation of a response (Algorithm 7). The operations in this step are constant, but many are repeated multiple times depending on the length of the lists of IDs and blinds. In the worst case, $\ell - 1$ vertices have already added one item to each of these lists. This makes the computational complexity of this step $O_R(\ell - 1)$. Similarly, the response has to be sent only once, but its length is the number of previous vertices plus one, making the communication complexity $O_C(\ell)$. Like the previous step, this step is repeated by all vertices in every layer of neighbours, except the last layer, so it is executed by more than $N_{max}^{\ell-1}$ vertices. This makes the complexities for the entire instance $O_R((\ell - 1) \cdot N_{max}^{\ell-1})$ and $O_C(\ell \cdot N_{max}^{\ell-1})$.

	Initiate	**Handle Message**		**Prop. Res.**	**Process Response**			**Protocol**
	v_0	$ttl = 0$	$ttl > 0$	v_i	$v_{0,A}$	STTP	$v_{0,B}$	*instance*
Runtime	$O(N_{max})$	$O(K)$	$O(N_{max})$	$O(\ell - 1)$	$O(\ell)$	$O(\ell)$	$O(\ell)$	-
Comm.	$O(N_{max})$	$O(K)$	$O(N_{max})$	$O(\ell)$	$O(\ell)$	$O(\ell)$	$O(\ell^2)$	-
Storage	$O(N_{max})$	-	$O(N_{max})$	-	$O(\ell)$	-	-	-
# per inst.	1	N_{max}^{ℓ}	$\sum_{i=1}^{\ell-1} N_{max}^{i}$	$\sum_{i=1}^{\ell-1} N_{max}^{i}$	$\sum_{i=1}^{\ell} N_{max}^{i}$	$\sum_{i=1}^{\ell} N_{max}^{i}$	$\sum_{i=1}^{\ell} N_{max}^{i}$	-
Inst. runtime	$O(N_{max})$	$O(N_{max}^{\ell})$	$O(N_{max}^{\ell})$	$O(\ell N_{max}^{\ell-1})$	$O(\ell N_{max}^{\ell})$	$O(N_{max}^{\ell})$	$O(\ell N_{max}^{\ell})$	$\boldsymbol{O(\ell N_{max}^{\ell})}$
Inst. comm.	$O(N_{max})$	$O(N_{max}^{\ell})$	$O(N_{max}^{\ell})$	$O(\ell N_{max}^{\ell-1})$	$O(\ell N_{max}^{\ell})$	$O(N_{max}^{\ell})$	$O(\ell^2 N_{max}^{\ell})$	$\boldsymbol{O(\ell^2 N_{max}^{\ell})}$
Inst. storage	$O(N_{max})$	-	$O(N_{max}^{\ell})$	-	$O(\ell N_{max}^{\ell})$	-	-	$\boldsymbol{O(\ell N_{max}^{\ell})}$

Table 5.1: Runtime, communication, and storage complexities of each step in the protocol and of an entire instance.

Next, when v_0 receives are response (Algorithm 8), it will do constant operations that are repeated for every ID in the list of IDs. Similarly, it will store one factor for every ID. Since there are at most ℓ vertices who added their ID, the runtime complexity for one execution of this step is $O_R(\ell)$ and the storage complexity is $O_S(\ell)$. Furthermore, it will send the response to the STTP, with the size depending on the length of the lists of IDs and blinds. This gives a communicational cost of $O_C(\ell)$. The storage required here is constant. This step will be repeated for every response that comes in. Since every neighbour in all layers of neighbours will send a response, the step must be repeated N_{max}^{ℓ} times. Brining all three complexities to $O(\ell \cdot N_{max}^{\ell})$.

When the STTP receives a response from v_0 (Algorithm 9), it will do similar operations as v_0 did: constant time operations repeated for every ID in the list, of which there are at worst ℓ. This gives a runtime complexity of $O_R(\ell)$. In the worst case, a response is always a match, meaning that the STTP always has to send the list of IDs to v_0, making communication $O_C(\ell)$ as well. Again, the STTP executes this step for every response in an instance. So, like before, the complexities for this step should be multiplied by N_{max}^{ℓ}, making them $O(\ell \cdot N_{max}^{\ell})$.

Finally, when v_0 receives the result of a match (Algorithm 10), it will again do constant operations for every ID in the list: $O_R(\ell)$. It will inform every vertex in the list with the list of IDs. This gives ℓ messages of size ℓ, bringing the computational complexity to $O(\ell^2)$. As the step is repeated for every response, the runtime complexity is $O_R(\ell \cdot N_{max}^{\ell})$ for the entire instance, and the communication complexity is $O_C(\ell^2 \cdot N_{max}^{\ell})$

Having derived the three complexities for all steps, they are summarised in Table 5.1. The table shows the three types of complexity for each of the steps in the protocol that are executed in each instance. The complexity for each individual step is shown, as well as the number of times each step is executed in each instance. This gives the total complexities per instance at the bottom of the table. By taking the worst complexity in every step, the total complexity of one instance of the protocol is determined.

In conclusion, the protocol has ha runtime complexity of $O(\ell N_{max}^{\ell})$, a communication complexity of $O(\ell^2 N_{max}^{\ell})$, and a storage complexity of $O(\ell N_{max}^{\ell})$. Additionally, it should be mentioned that these are the worst case complexities wherein it is assumed that all vertices have the same maximum number of out-neighbours. This is not realistic, because in a transaction network the number of neighbours varies greatly between bank accounts, with only a very small number of accounts having many neighbours. Therefore, a more realistic expression of the complexities can be given by using the average number of out-neighbours N_{avg} instead of the maximum. Also, in a realistic setting a match is very unlikely, making the practical communication for the STTP when processing the response constant, and making the last step where v_0 handles the result $v_{0,B}$ negligible in computation and communication. Thus, the protocol has a realistic runtime complexity of $O(\ell N_{avg}^{\ell})$, a realistic communication complexity of $O(\ell N_{avg}^{\ell})$, and a realistic storage complexity of $O(\ell N_{avg}^{\ell})$.

6 Evaluation

This chapter introduces two evaluations of the protocol. First, an evaluation of the accuracy in presented which shows the protocol is able to match entities, even if variations and errors are present in the attributes. Secondly, the runtime of the protocol is evaluated, which supports the theoretical runtime complexity and demonstrates that the protocol is highly parallelisable.

For both evaluations, the protocol is implemented in Java 21 and run on synthetic data. The source code of this implementation and data generation can be found on Github [44].

6.1. Accuracy Evaluation

In the protocol, identities are encoded as Bloom filters. This allows for the cryptographically secure computation of similarity between them. However, since Bloom filters are a probabilistic data structure, the distances between them are only an approximation of the actual distances between the attribute strings. While it is inevitable that some accuracy is lost, Schnell et al. (2009) [58] have already shown that with the right parameters the effect is minimal, and that near perfect entity resolution on real data is possible. Since the cryptographic methods used in the protocol do not affect the correctness of the distance computation between the filters, similar performance should be expected.

It is, however, difficult to evaluate the protocol in a realistic AML setting. There are two reasons for this. Firstly, no real data representing a realistic transaction network is publicly available. While individual banks are already extremely reserved when it comes to publishing (anonymised) transaction data, the data that is available only contains intra-bank transactions, e.g. [57]. Such data creates a limited and very skewed view of a real transaction network and would therefore not yield representative results. Secondly, realistic entity resolution on bank customer identities involves many attributes and a complex set of deterministic matching rules. As mentioned in Section 4.3, such a rule set can only be constructed if the attributes are known and standardised, and if appropriate thresholds are determined by empirical evaluation of the domain. Therefore, only a basic evaluation of the protocol's accuracy is performed here using a simple transaction graph and basic identities.

The goal of this evaluation is twofold: to demonstrate correct operation of the protocol and to show the effect of using different matching rules. The only attributes used are first name and last name. The transaction graph used is shown in Figure 6.1. Each account has an identity as shown in Table 6.1. In the network, there are five accounts that owned by the same entity. Identities A and B match exactly, and E, G and L are variants that might occur when names are corrupted when entering the system of a bank or national passport. Account I is owned by a different person with a similar name.

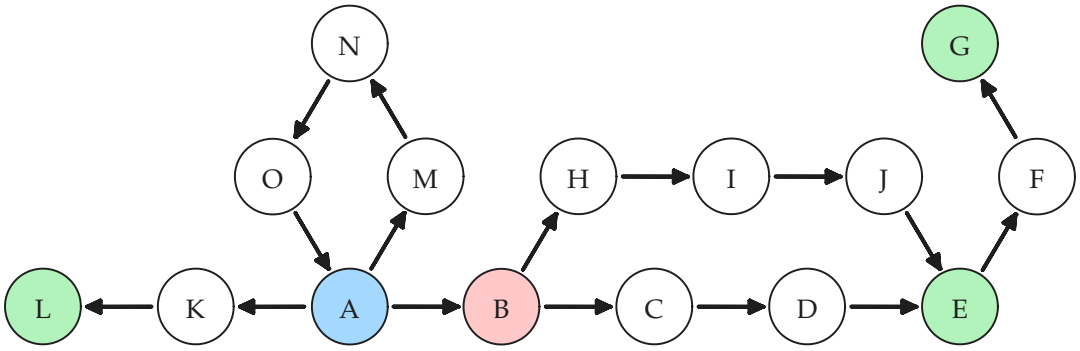


Figure 6.1: Simple transaction network. The initialising vertex is A. The green vertices have a matching identity. The red vertex B has a matching identity, but the path to B is smaller than ℓ_{min}.

id	First name	Last name
A	**Álice**	**de Vrij**
B	**Álice**	**de Vrij**
C	Charlie	Foo
D	David	Bar
E	**Alice**	**Devry**
F	Frank	Baz
G	**Alice**	**de Vrij**
H	Hendrick	Bazz
I	Alin	Devon
J	John	Smith
K	Karen	Bazzzz
L	**Alice**	**Vrij**
M	Maria	Bazzzzz
N	Nick	Bazzzzzz
O	Olivia	Bazzzzzzz

Table 6.1: Identities for accounts in Figure 6.1. Rows highlighted in bold are variations of the same identity.

In this experiment (run on a machine with a Intel Core i7-7700 CPU and 24 GB of RAM), the protocol is run multiple times on the network with different matching rules. For simplicity, the structure of the rules is kept the same, but the thresholds are varied. The matching rules are structured as follows:

$$\frac{d_H(\text{firstname})}{2k} < t_1 \quad \wedge \quad \frac{d_H(\text{lastname})}{2k} < t_2 \tag{6.1}$$

Here, k is the number of independent hash functions (constant), and t_1 and t_2 are the thresholds. The following parameters are used: minimal path length $\ell_{min} = 2$, maximum path length $\ell = 6$, Bloom filter length $l = 500$, number of hash functions $k = 15$, and number of bits for the Paillier modulus $bl = 2048$. Each time, account A serves as the initiating vertex.

Both t_1 and t_2 are varied from 1 to 4, resulting in 16 different matching functions tested. Table 6.2 shows which matching vertices are found by the protocol for each matching rule. Table 6.3 shows for each matching vertex the discovered paths, which are the same for every run in which a match with the vertex is found.

t_1	t_2	Matching vertices
1	1	A
1	2	A
1	3	A
1	4	A
2	1	A, G
2	2	A, G, L
2	3	A, G, L
2	4	A, G, L, E
3	1	A, G
3	2	A, G, L
3	3	A, G, L
3	4	A, G, L, E
4	1	A, G
4	2	A, G, L
4	3	A, G, L
4	4	A, G, L, E, I

Table 6.2: Matches discovered by running the protocol on network 6.1 with different matching rules. More matches are found with larger thresholds.

Matching vertex	List of ids (path)
E	[B,C,D,E]
E	[B,H,I,J,E]
G	[B,C,D,E,F,G]
L	[K,L]
A (*cycle*)	[M,N,O,A]

Table 6.3: Paths found to matching vertices. The protocol correctly reveals two paths to vertex E, and that there exists a cycle.

The results show as the threshold sizes increase, so does the number of matched vertices. For every matching rule, a match with vertex A is found. This is the trivial case of the cycle [A,M,N,O,A]: vertex A is the initiating vertex and the protocol has matched it with itself. Furthermore, as the thresholds increase, the first additional match that is found is with vertex G. This is because its identity is the most similar to A. If t_1 is larger than 1 and t_2 is set to 4, matches with vertices L and E are also found. However, note that if both t_1 and t_2 go to 4, the protocol also lists a match with vertex I. This is a false positive because vertex I has a different identity than vertex A and is not just a variation of 'Álice de Vrij'. This highlights the importance of finding optimal matching rules and thresholds: if the thresholds are too low, not all matching vertices are discovered (false negatives); if the thresholds are too high, non-matching vertices are labelled as matching (false positives). In any implementation of the protocol, the matching rules and thresholds must be carefully chosen to find an optimal balance between false positives and false negatives. In this experiment, where there are only two attributes and the total number of values is limited, optimal thresholds can be found: $2 \leq t_1 \leq 3$ and $t_2 = 4$ give no false positives nor false negatives.

Moreover, note that the protocol correctly detects no match with vertex B, even though its identity is exactly the same as A's identity. This is because B is only one hop removed from A and the minimum path length parameter ℓ_{min} is set to 2. Finally, Table, 6.3 lists vertex E twice. This is because the protocol detects two different matches with this vertex via two different paths. This is important, because it shows the protocol does not only reveal to which account money is being laundered, but also all the accounts and routes involved.

6.2. Runtime Evaluation

The goal of this second evaluation is to find how the runtime of one instance of the protocol scales with varying transaction network sizes and entity cycle lengths. The protocol is implemented monolithically, but uses threading to simulate the fully decentralised model where each vertex is an individual computational agent. To evaluate the performance, it is run on a computational node with two Intel XEON E5-6448Y CPUs with a combined total of 64 cores and 256 GB of memory [19].

For each run of the protocol, the following parameters are fixed: minimum path length $\ell_{min} = 1$, Bloom filter length $l = 500$, number of hash functions $k = 15$, and number of bits for the Paillier modulus $bl = 2048$. Also, identities consist of the same two attributes each time, first name and last name. These are generated randomly for each vertex in all networks, so the probability of finding a match is extremely low. Furthermore, to better analyse how the runtime scales for different transaction networks, the constant time ($\sim 12s$) taken for set up and key generation is excluded in the measurements.

The networks are synthetically generated in order to control the average number of out-neighbours. To omit having to run the protocol many times with different initiating vertices in each network, the out-degree for every vertex is set exactly to the average. The protocol is run on networks ranging in size from $N_{avg} = 1$ up to $N_{avg} = 18$. The experiment is repeated with three different parameters for the maximum path length ℓ. The results are shown in Figure 6.2.

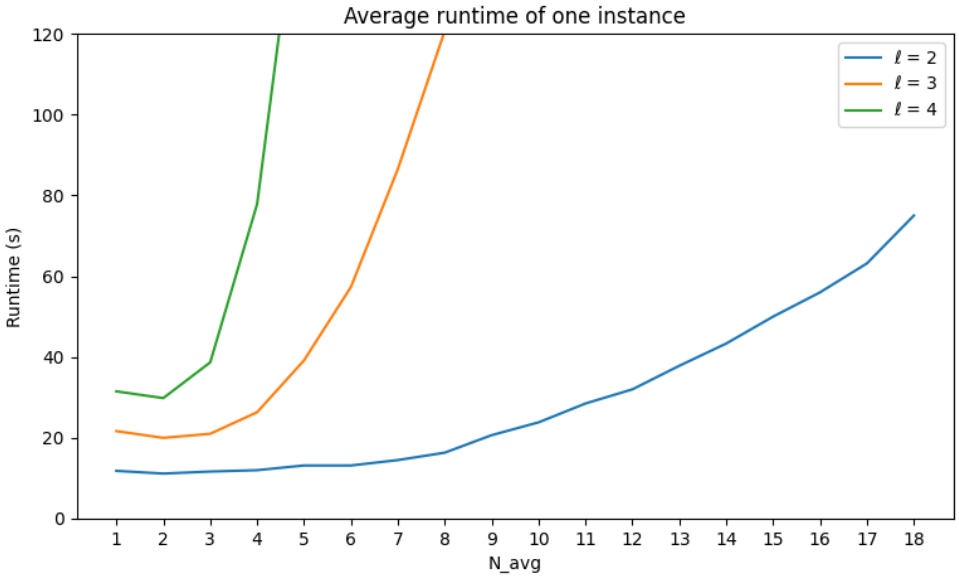


Figure 6.2: Effect of the average number of out-neighbours per vertex (N_{avg}) on the average runtime for maximum path lengths $\ell = 2$, $\ell = 3$, and $\ell = 4$. The runtime scales polynomially on N_{avg} with ℓ as the degree.

The results show that the runtime scales polynomially with the average out-degree of the network N_{avg}. The figure demonstrates that the degree of this polynomial is the maximum path length ℓ. These results support the finding from the analysis that the practical runtime complexity is $O(N_{avg}^{\ell})$.

Furthermore, by looking more closely at Figure 6.2 it can be seen that the relation between N_{avg} and the runtime is not a perfect curve. In fact, for the smaller networks, the runtime remains roughly constant and only starts growing exponentially later. This is best visible in the line for $\ell = 2$, where the time stays close to 12 seconds for the first 7 networks and only starts growing quadratically from $N_{avg} = 8$. The same trend is visible for $\ell = 3$, where the runtime does not grow at all for the first three networks. Similarly, for $\ell = 4$, the two smallest networks that roughly an equal amount of time.

This trend can be analysed more closely by plotting the results in a different way. Figure 6.3 shows how the runtime is affected by the number of vertices in the last layer of each network. The red vertical line in the plot ($N = 64$) indicates the number of available threads on the machine.

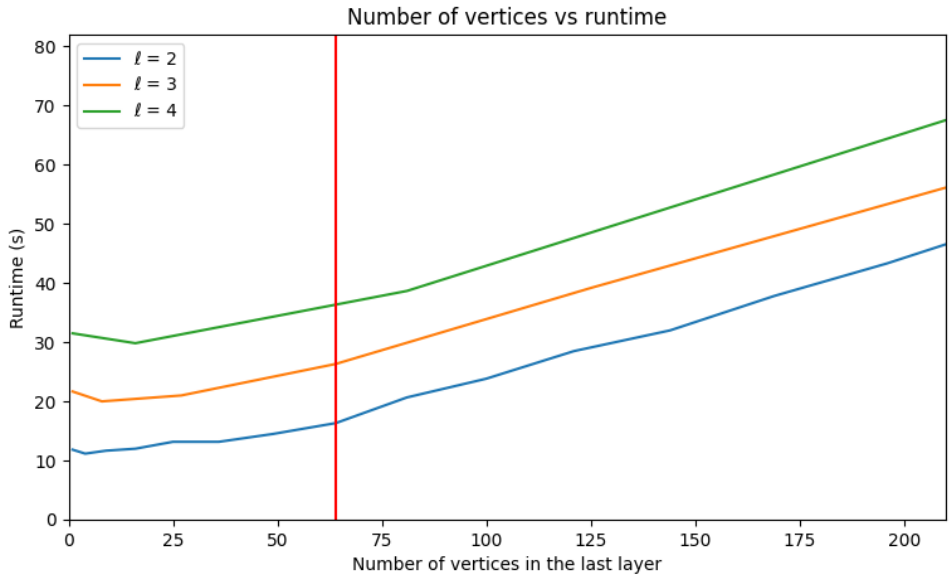


Figure 6.3: Effect of the number of vertices in the last layer (V_{last}) on the average runtime for maximum path lengths $\ell = 2$, $\ell = 3$, and $\ell = 4$. The vertical line indicates the maximum number of available threads. The runtime only grows minimally if free threads are still available ($V_{last} < 64$), but grows significantly faster if all threads are in use ($V_{last} \geq 64$).

This plot shows that the runtime grows linearly with the number of vertices in the final layer. Here, it is more clearly visible that there is indeed a change in how fast the increases at a certain point. For $\ell = 2$, the runtime increases with only 4.8 seconds between $V_{last} = 0$ and $V_{last} = 64$, averaging 0.075 seconds per extra vertex. After $V_{last} = 64$, the growth is constant at roughly 0.2 seconds per extra vertex. This change is almost identical for $\ell = 3$ and $\ell = 4$.

It is no coincidence that the point where the runtime starts increasing significantly faster is at 64 vertices. This being exactly the number of available threads on the machine running the program, it suggests that the work required for each extra vertex is executed in parallel to the others. In the implementation, a new thread is spawned for the handling of each message and response. The computational bottleneck in the protocol is the forwarding of the messages, which requires refreshing the randomness on each encryption once. This step takes significantly more time than any other operation in the protocol. Hence, it is this phase which keeps the threads occupied. When a layer has more vertices than there are cores available, the randomness refreshing operations can no longer be fully parallelised, resulting in a steep increase in runtime. Conversely, when there are free threads available, the runtime stays roughly constant. There is only a slight increase in runtime when V_{last} gets close to the limit of 64, which can be explained by the fact that the other steps (i.e. forwarding + processing results) are taking up some small amount of time on the cores as well.

7

Discussion

In this chapter, the research and the resulting protocol are discussed in some detail. Primarily, Section 7.1 explains the main practical limitations that impacted the research. Next, Section 7.2 discusses the protocol and some possible extensions to it. After that, Section 7.3 discusses potential performance of the protocol in a real-world setting. Finally, Section 7.4 contains a discussion on the practical considerations regarding the protocol's accuracy.

7.1. Research Limitations

This research project is affected by some practical limitations that impact the design of the protocol and its evaluation. The main limitation is the lack of access to real-world financial data and to insider knowledge of the operation of banks. The protocol presented in this work operates on data representing the global financial transaction network and bank customer information. While the basic structure of this data is simple and publicly available, examples of actual representative datasets are not accessible. This is because such real data represent the actual banking information of real people, which, logically, cannot be shared with the public. The availability of anonymised transaction data is also very limited.

To the best of the author's knowledge, the most realistic example of bank transaction data is the publication by Saxena et al. (2022), which entails a transaction network containing 1.6 million accounts of a Dutch bank [57]. While this dataset provides some insights into the real-world structure of a transaction network, it cannot be used as a representative example of the global transaction network because it only contains intra-bank transactions. All transactions to accounts of other banks have been left out, making the provided graph statistics highly skewed. Also, the publication only provides a simple graph of the network, and does not include any other relevant information, such as data about account ownership.

Another facet of financial data that remains obscured is the exact nature of the customer data that banks collect during the KYC procedure, which can thus be used to resolve account ownership. Sources explaining the KYC data collection focus mostly on the goal of the customer identification and only give very basic examples of data collected [18] [59] [60]. It is mentioned that the KYC procedure involves obtaining data from a person's passport or ID-card (or a similar document in the case of an organisation), but sources are vague about the use of any other information. Additionally, the literature is unclear about to what extent the individuals controlling a company or organisations are identified during KYC.

These limitations impact this research in two ways. Firstly, it makes designing a realistic matching function for financial customer data infeasible. In deterministic entity resolution, the exact matching rules that are used greatly impact the performance of the method [10]. Such rules must be carefully crafted based on the attributes and the possible values of the data. Since such realistic bank customer data is not available, it is not possible to construct a realistic matching function.

This, however, is not problematic for the design of the protocol. The goal of this research is to find how entity cycles can be detected privately in a transaction network. While entity resolution is an important part of this, the exact matching rules used do not affect any other part of the protocol. The design provides a framework for private matching by the STTP using the distances between the attributes. The protocol can be implemented with any set of matching rules, which can be viewed as a parameter to be picked when implemented.

Secondly, the lack of real transaction data makes it difficult to find the protocol's performance in a practical setting. This is most hindering for the evaluation of the accuracy, which can only be tested on a very high level with synthetic data. Here, the problem is twofold: the lack of customer identity data results not only the omission of a matching function, but also the evaluation of the protocol on representative data. Furthermore, even if an approximate dummy matching function would be implemented and personal data from other domains were to be substituted for bank customer data, there still is no realistic transaction network available to run the protocol on. Therefore, this limitation makes that only basic, very synthetic evaluation of the protocol's accuracy can be performed. Nevertheless, this evaluation is still able to show the capabilities of the protocol. Also, it is less of an issue for the evaluation of the runtime, because generating synthetic data offers more flexibility in running the protocol on different networks.

7.2. Possible Extensions to the Protocol

The protocol as explained in Chapter 4 achieves the goal of detecting entity cycles in transaction networks while preserving privacy. The step that are presented, however, only describe a very basic version of the protocol that is only concerned with effectiveness and security. Whereas from the research perspective these are the two most important qualities, parties having to actually implement such a protocol also care very much about the efficiency of the design. Regarding this, it should be mentioned that certain extensions that improve the protocol efficiency are indeed possible and have been experimented with, but are omitted from this report for the reasons of brevity and simplicity. There are three main extensions or modifications to the protocol that can help speed up the money laundering detection process in practice.

Firstly, the main version of the protocol only propagates messages over the network along the direction of the edges. This means that an initiating vertex only receives responses from neighbours it has sent money to. If this vertex is on the receiving end of the money laundering, its activities can only be detected if the protocol is run starting in the matching vertex. In other words, the protocol needs to initiated and run from all accounts that send money to a given account (with a number of accounts in between) in order to detect if this account is involved in money laundering. These redundant steps can be easily avoided by a simple extension to the protocol. By including a simple flag in each message that indicates to the vertices the direction of the edges that should be followed, one instance of the protocol can detect matching vertices in all money flows the initiating vertex is involved in. When the flag is set to 0, all vertices propagate messages to their out-neighbours, and when it is set to 1 they propagate to their in-neighbours. This flag should not compromise the unlinkability of the messages because the flag only has two possible values, making it infeasible to link messages when many instances are running.

Moreover, the protocol scales on the average out-degree of the network. When taking into account all other accounts where an account has had a transaction with, this can be quite a large number. However, many transactions are irrelevant for detecting money laundering. Transactions that happened many years ago might not be useful for finding currently active launderers. Similarly, transactions of a small amount are unlikely to be part of a laundering operation. Consequently, it is not useful for the protocol to propagate messages over transactions that are not relevant in detecting money laundering. Therefore, a simple addition to the protocol is to have a selection of which neighbours messages should be propagated to, instead of blindly propagating to all neighbours. The values on the edges, i.e. timestamp and amount of the transaction, can be used to filter irrelevant neighbours or to construct a narrow subset of interesting neighbours. Such conditions can be used as a parameter of the protocol.

Lastly, the protocol models the transaction network as fully decentralised where each account is an independent agent. In reality, this is not the case. Rather, banks control the accounts and each bank controls many accounts. In the protocol, this leads to situations where an account propagates a message or response to a neighbour that is operated by the same bank. In such a case, it is not required to protect the privacy and unlinkability of the message, because the bank operating the neighbour already knows everything the sending account knows. Taking these cases into account allows for certain operations to be skipped during execution, leading to better runtimes. For example, a neighbour of the initiating vertex does not need to send a response if both belong to the same bank. The bank can resolve the entities internally without having to protect privacy. Similarly, replying to a message can be skipped is there exists a chain of accounts operated by the same bank back to the initiator. Furthermore, if all neighbours of a vertex are operated by the same bank as itself, then there is no need to refresh randomness.

7.3. Hypothetical Real-World Performance

It is difficult to investigate the performance of the protocol in a real-world setting because very little is known about the details of the global bank transaction network. As explained in Section 7.1, such representative data is not publicly available. For this reason, the evaluation in Chapter 6 is performed using synthetic networks. Despite this, it is possible to reason about the protocol's performance in a hypothetical real-world setting.

The runtime evaluation shows that for the messages in the last layer of the network to be fully processed in parallel (for an average instance), the number of cores available needs to be at least the number of vertices in the last layer, which is given as N_{avg}^{ℓ}. In the context of AML, the (entity) cycles that indicate money laundering are small, with a length of 6 typically used as a maximum [32]. Thus, in a practical setting, ℓ would be 6 at most. It is difficult to find a good approximation for the average number of out-neighbours N_{max} of the global bank transaction network. However, the data published by Saxena et al. (2022) can be used to find a very rough estimate [57]. This dataset includes all the intra-bank transactions for a large Dutch bank and reveals that such a network has an average out-degree of 2.36 neighbours per vertex. However, this figure is skewed because it only includes transactions made to accounts in the same bank. At the same time, this network includes transactions from a period of 11 years. When detecting money laundering, one is usually only interested in transactions that happened recently, e.g. half a year, and will therefore exclude any older transactions [32]. Similarly, many transactions can be excluded based on amount, as only significant sums of money transferred are relevant for money laundering detection (see also Section 7.2). Therefore, a very rough estimate would be that a typical transaction network would have an N_{avg} in the order of magnitude 2-20.

Using $N_{avg} = 15$ and $\ell = 6$ as an example, eleven million cores would be required for fully parallelised computation. Note that this computing power is distributed over the participating banks. When running the protocol on the global transaction network, the accounts involved in an instance are likely spread over thousands of banks, meaning an individual bank only responsible for a small fraction of the computation. Note also that this number of cores is the requirement for full parallelisability. Figure 6.3 shows that each layer of neighbours adds around 13 seconds to the runtime, making the runtime 78 seconds for $\ell = 6$ if enough cores are available. That parties can choose to which degree that they want to run the protocol in parallel. E.g. they can allow the protocol to take twice as long to run, which saves half of the required cores.

Furthermore, it is also important to highlight that the shape of the transaction network also plays a role. Payment networks are typically scale-free networks [42]. Saxena et al. found that their intra-bank transaction graph also exhibits many characteristics of a scale-free network [57]. In such a network, the degree distribution follows a power law, meaning that there are many vertices with a degree below the average and a few vertices with a degree far above the average. Qiu et al. (2018) refer to vertices with a large out-degree as 'hot-points' [51]. They highlight that the number of vertices encountered when visiting all neighbours grows rapidly when a hot-point is encountered. This phenomenon is also expected to occur when the protocol introduced in this work is run on a real transaction network. This means that great variance can occur in the expected runtime (or required cores): if a hot-point is encountered in the penultimate layer, then the last layer will be large. Conversely, most vertices visited will have and out-degree well below N_{avg}, making the number of vertices visited grow slowly.

7.4. Protocol Limitations

The protocol detects short entity cycles in a transaction network, which are paths between bank accounts that are owned by the same entity. The existence of an entity cycle is an indication of money laundering. The effectiveness of the protocol depends on the effectiveness of resolving two identities of bank accounts to the same entity. As shown in Section 6.1, the protocol is able to do this even when the attributes of the identities do not exactly match. The use of entity resolution allows effective matching of identities whose attributes are roughly similar but not the same due to natural variations in the values or errors.

However, the accuracy of this matching is affected by the matching rules that are used. These rules must be optimised for the data and the domain, which in this case is personally identifiable information of bank customers (people and organisations). Since distinguishing between identities that are variations of each other and identities that are actually different is difficult, and the protocol also uses Bloom filters, which are probabilistic data structures, false positives and false negatives will always be present. Thresholds for what is a match and what is not must be set such that an acceptable balance is struck between these errors. If banks are to implement such a protocol, they must be careful as were to set the boundaries, because both false positives and false negatives have impact on the real world. A false negative represents an undetected money flow between two accounts owned by the same entity. If there are many false negatives, the protocol is inefficient in achieving its goal. Conversely, a false positive represents a case where two accounts owned by different people are matched. Such a case would trigger an additional investigation into the accounts and their owners, violating the privacy of people who have done nothing wrong. Too many false positives would for a large part defeat the purpose of the protocol.

Furthermore, it should be noted that there is an additional kind of false positive. Even if two accounts are correctly matched, i.e. they are indeed owned by the same entity, it does not necessarily mean that money is being laundered. There are alternative cases imaginable where people operate different bank accounts and move money in between, without any criminal intent. A person may have multiple accounts for different purposes. This might even be more common for companies. For this reason it is important that the protocol is correctly used. Specifically, the ℓ_{min} parameter should be set to not detect matches between direct neighbours. Money laundering involves the practice of layering: hiding the origin of money by moving it through many accounts. The ℓ_{min} paraments should be tweaked to balance the number of false positives and false negatives.

Finally, regarding this second kind of false positive, it should again be stressed that the presence of an entity cycle is only an indicator of money laundering and cannot serve as direct evidence of money laundering. Detection of an entity cycle should always be followed up by addition investigation, as to 1) confirm that the accounts actually match, and 2) the entity involves is actually laundering money. Ideally, this should be done in stages where the privacy is preserved as much as possible.

8

Conclusion

Money laundering is a serious criminal enterprise that functions as the financial backbone of many criminal operations. Banks and other financial institutions are legally tasked with the detection and prevention of money laundering by regulations requiring them to set up anti-money laundering (AML) processes. With the recent increase in severity of organised crime in Europe, many (inter-)governmental organisations are introducing new legislation to boost the effectiveness of AML operations. Many such new regulations focus on strengthening AML collaboration by lowering the legal threshold for exchanging financial data between banks and government agencies. These developments, however, have a great impact on the privacy of innocent citizens. Such prevalent exchange of personal and financial information heavily infringes on people's right to keep sensitive details about their lives hidden from organisations and governments. In order to fight money laundering and organised crime, solutions are needed that achieve the same AML effectiveness without sacrificing the privacy rights of innocents.

Privacy preserving anti-money laundering (PPAML) methods are solutions that enable financial institutions to collaboratively detect money laundering without explicitly having to share sensitive data. Previous PPAML works have introduced methods for running algorithms in a decentralised transaction network and detecting transaction patterns that indicate money laundering. Notably, methods to detect cycles in bank transactions can be very effective in privately detecting money laundering activities. However, the current state-of-the-art methods only cover a basic case. These methods do not cover cases where money is laundered from a given entity and back while different source and destination accounts are used. This flaw serves as the research gap for this thesis, which leads to the following research question: *How can money laundering cycles in transactions be detected privately when criminals use multiple bank accounts at different banks?*

The research question is answered by the introduction of a new PPAML protocol that privately detects entity cycles in a decentralised transaction graph. The protocol works by having bank accounts exchange messages to their neighbours about their identity. The identity, consisting of a set of string attributes, is first encoded as a Bloom filter and then encrypted with the Paillier encryption scheme. Utilising the Paillier scheme's additively homomorphic nature, the protocol allows accounts to compute the distance between another account's identity and their own, all while preserving confidentiality. The messages and responses can be privately propagated through the network by refreshing the randomness on the encryptions before each hop. Additionally, a semi-trusted third party (STTP) is used to decrypt the distances and find if two identities match. When there is match, the accounts can be directly informed by direct contact, eliminating the need to traverse the graph again.

Analysis of the protocol shows that it is secure against information leakage in the semi-honest security model. The private propagation of messages throughout the network ensures that it is infeasible for adversarial vertices to learn about the existence of edges or paths, thereby keeping the topology of the transactions network private. Additionally, the security of the protocol is not affected by collusion between vertices. It is only important that the STTP does not collude.

Moreover, the protocol is designed to be secure against some attacks in the covert security model. The use of different public keys prevents adversaries from attempting to learn the value of received identities, and the use of blinding factors to link variables prevents adversaries from forging malicious decryption requests.

Furthermore, analysis of the complexities shows that the protocol scales based on the average number of neighbours in the network, and the desired cycle length to be detected. The formal worst-case runtime complexity is $O(\ell N_{max}^{\ell})$, where ℓ is the cycle length and N_{max} the maximum number of out-neighbours. However, since in practice only very few vertices will have many neighbours, the realistic runtime complexity is $O(\ell N_{avg}^{\ell})$, where N_{avg} is the average number of out-neighbours per vertex in the network. The same is the case for the realistic communication and storage complexities.

Evaluation of the protocol shows that it is highly parallelisable. Since the protocol is fully decentralised, operations by one vertex are completely independent of operations by another. Additionally, the steps are independent, meaning that one vertex can process many messages in parallel. The evaluation shows that the consequence of this is that the practical runtime of a protocol instance is determined by the number of cores available. When sufficient cores are free, the runtime of an instance stays roughly constant as N_{avg} increases. Only when the maximum number of available cores is reached does the runtime start to follow the polynomial theoretical bound.

The protocol introduced in this work does not include an exact matching function for bank customer identity records. Future works could seek collaboration with the financial sector to gain access to such data, and use it to investigate which exact attributes can be used in the matching function and what an optimal matching function would look like. Furthermore, the extensions which are only explained briefly here can be further implemented to yield a more efficient version of the protocol. Moreover, the techniques used in this and related works can be used as a framework for developing similar methods that detect other money laundering patterns in transaction networks. Finally, research should go into how technical solutions, such as this one, can be implemented in practice so that societal challenges involving big data can be resolved while also maintaining legislation that protects privacy.

References

[1] Hanaa Abbas and Roberto Di Pietro. 2025. *Blinding Techniques*. Springer, Cham, Zug, Switzerland, 275–277. doi:10.1007/978-3-030-71522-9_1764

[2] Erik R. Altman, Jovan Blanuša, Luc von Niederhäusern, Béni Egressy, Andreea Anghel, and Kubilay Atasu. 2023. Realistic Synthetic Financial Transactions for Anti-Money Laundering Models. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, Vol. 36. 29851–29874. https://proceedings.neurips.cc/paper_files/paper/2023/hash/5f38404edff6f3f642d6fa5892479c42-Abstract-Datasets_and_Benchmarks.html

[3] Yonatan Aumann and Yehuda Lindell. 2010. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. *Journal of Cryptology* 23 (2010), 281–343. doi:10.1007/s00145-009-9040-7

[4] Authority for Anti-Money Laundering and Countering the Financing of Terrorism . [n. d.]. *About AMLA*. Retrieved 2026-04-21 from https://www.amla.europa.eu/about-amla_en

[5] Autoriteit Persoonsgegevens. 2021. *AP adviseert Eerste Kamer: neem WGS niet aan*. Retrieved 2026-04-24 from https://www.autoriteitpersoonsgegevens.nl/actueel/ap-adviseert-eerste-kamer-neem-wgs-niet-aan

[6] Autoriteit Persoonsgegevens. 2022. *Nieuwe wet opent deur naar ongekende massasurveillance door banken*. Retrieved 2026-04-16 from https://www.autoriteitpersoonsgegevens.nl/actueel/nieuwe-wet-opent-deur-naar-ongekende-massasurveillance-door-banken

[7] Autoriteit Persoonsgegevens. 2025. *Wet gegevensverwerking door samenwerkingsverbanden*. Retrieved 2026-04-16 from https://www.autoriteitpersoonsgegevens.nl/themas/politie-en-justitie/bestrijding-van-ondermijnende-criminaliteit/wet-gegevensverwerking-door-samenwerkingsverbanden

[8] Chris Bagnall. 2024. *How Information Sharing Optimizes the Fight Against Financial Crime*. Quantexa. Retrieved 2026-04-20 from https://www.quantexa.com/blog/information-sharing/

[9] Elaine Barker and Allen Roginsky. 2019. *NIST Special Publication 800-131A Revision 2 - Transitioning the Use of Cryptographic Algorithms and Key Lengths*. Technical Report. National Institute of Standards and Technology. doi:10.6028/NIST.SP.800-131Ar2

[10] Olivier Binette and Rebecca C. Steorts. 2022. (Almost) all of entity resolution. *Science Advances* 8, 12 (2022). doi:10.1126/sciadv.abi8021

[11] Jovan Blanuša, Maximo Cravero Baraja, Andreea Anghel, Luc von Niederhäusern, Erik R. Altman, Haris Pozidis, and Kubilay Atasu. 2024. Graph Feature Preprocessor: Real-time Subgraph-based Feature Extraction for Financial Crime Detection. In *Proceedings of the 5th ACM International Conference on AI in Finance, ICAIF 2024*. 222–230. doi:10.1145/3677052.3698674

[12] Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* 13, 7 (1970), 422 – 426. doi:10.1145/362686.362692

[13] Michael Brand, Hamish Ivey-Law, and Tania Churchill. 2023. FinTracer: A privacy-preserving mechanism for tracing electronic money. Cryptology ePrint Archive, Paper 2023/649. https://eprint.iacr.org/2023/649

[14] Vincent Böhre. 2026. *Consultatiereactie - Internetconsultatie Besluit tot wijziging Besluit BRP i.v.m. derdenaanwijzing banken*. Stichting Privacy First. Retrieved 2026-04-24 from https://privacyfirst.nl/wp-content/uploads/PrivacyFirst_consultatie_BRP_23-3-2026.pdf

[15] Cambridge Dictionary. [n. d.]. *Privacy | English meaning*. Retrieved 2026-04-21 from https://dictionary.cambridge.org/dictionary/english/privacy

[16] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2020. An Overview of End-to-End Entity Resolution for Big Data. *ACM Computing Surveys* 53, 6 (2020), 1–42. doi:10.1145/3418896

[17] Council of the European Union. 2024. *Anti-money laundering: Council adopts package of rules*. Retrieved 2026-02-05 from https://www.consilium.europa.eu/en/press/press-releases/2024/05/30/anti-money-laundering-council-adopts-package-of-rules/

[18] Dennis Cox. 2014. *Handbook of Anti-Money Laundering* (1st ed.). John Wiley Sons Ltd, Chichester, West Sussex, United Kingdom. 752 pages. doi:10.1002/9780470685280

[19] Delft High Performance Computing Centre (DHPC). 2024. DelftBlue Supercomputer (Phase 2). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2.

[20] Bogdan Dumitrescu, Andra Băltoiu, and Ştefania Budulan. 2022. Anomaly Detection in Graphs of Bank Transactions for Anti Money Laundering Applications. *IEEE Access* 10 (2022), 47699–47714. doi:10.1109/ACCESS.2022.3170467

[21] European Commission. 2021. *Communication from the Commission to the European Parliament, the Council, the European Economic and Social Committee and the Committee of the Regions on the EU Strategy to tackle Organised Crime 2021-2025*. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A52021DC0170

[22] Europol. 2024. *Decoding the EU's Most Threatening Criminal Networks*. Technical Report. https://www.europol.europa.eu/cms/sites/default/files/documents/Europol%20report%20on%20Decoding%20the%20EU-s%20most%20threatening%20criminal%20networks.pdf

[23] Europol. 2025. *European Union Serious and Organised Crime Threat Assessment – The changing DNA of serious and organised crime*. Technical Report. https://www.europol.europa.eu/cms/sites/default/files/documents/EU-SOCTA-2025.pdf

[24] Facctum. [n. d.]. *What Is Entity Resolution In AML And Why Does It Matter?* Retrieved 2026-05-04 from https://www.facctum.com/terms/entity-resolution

[25] Financial Crime Academy. 2026. *Anti-Money Laundering: AML Important Definition*. Retrieved 2026-04-17 from https://financialcrimeacademy.org/anti-money-laundering-aml/

[26] Financial Crime Academy. 2026. *Role of the Financial Intelligence Unit*. Retrieved 2026-04-20 from https://financialcrimeacademy.org/role-of-the-financial-intelligence-unit/

[27] FinancialCrime.lu. 2025. *FATF ¦ R.29 Financial Intelligence Units*. Retrieved 2026-04-20 from https://financialcrime.lu/col_slider/2025-11-24-FATF-Recommendation-29/

[28] Jacopo Fior, Thomas Favale, Luca Cagliero, Danilo Giordano, Marco Mellia, and Elena Baralis. 2022. Legal Entity Disambiguation for Financial Crime Detection. In *2022 IEEE International Conference on Big Data*. 6639–6641. doi:10.1109/BigData55660.2022.10020700

[29] Financial Action Task Force. 2017. *Guidance on private sector information sharing*. Technical Report. https://www.fatf-gafi.org/en/publications/Fatfgeneral/Guidance-information-sharing.html

[30] Aris Gkoulalas-Divanis, Dinusha Vatsalan, Dimitrios Karapiperis, and Murat Kantarcioglu. 2021. Modern Privacy-Preserving Record Linkage Techniques: An Overview. *IEEE Transactions on Information Forensics and Security* 16 (2021), 4966 – 4987. doi:10.1109/TIFS.2021.3114026

[31] Yuxiang Guo, Lu Chen, Zhengjie Zhou, Baihua Zheng, Ziquan Fang, Zhikun Zhang, Yuren Mao, and Yunjun Gao. 2023. CampER: An Effective Framework for Privacy-Aware Deep Entity Resolution. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 626–637. doi:10.1145/3580305.3599266

[32] László Hajdu and Miklós Krész. 2020. Temporal Network Analytics for Fraud Detection in the Banking Sector. In *ADBIS, TPDL and EDA 2020 Common Workshops and Doctoral Consortium*, Vol. 1260. 145–157. doi:10.1007/978-3-030-55814-7_12

[33] Carmit Hazay and Yehuda Lindell. 2010. *Efficient Secure Two-Party Protocols* (1st ed.). Springer, Heidelberg, Baden-Württemberg, Germany. 263 pages. doi:10.1007/978-3-642-14303-8

[34] Juno Jense. 2024. *Finding Bounded-Length Cycles in Decentralised Networks under Privacy Constraints*. Master's thesis. Delft University of Technology, Delft, The Netherlands. Advisor(s) Zeki Erkin. Retrieved 2026-02-05 from https://resolver.tudelft.nl/uuid:a559439a-d5ee-4f1f-9823-2bab3905c0c9

[35] Donald B. Johnson. 1975. Finding All the Elementary Circuits of a Directed Graph. *SIAM Journal on Computing* 4, 1 (1975), 77–84. doi:10.1137/0204007

[36] Dimitrios Karapiperis and Vassilios S. Verykios. 2015. An LSH-Based Blocking Approach with a Homomorphic Matching Technique for Privacy-Preserving Record Linkage. *IEEE Transactions on Knowledge and Data Engineering* 27, 4 (2015), 909 – 921. doi:10.1109/TKDE.2014.2349916

[37] Jonathan Katz and Yehuda Lindell. 2021. *Introduction to Modern Cryptography* (3rd ed.). CRC Press, Abingdon-on-Thames, England, United Kingdom. 626 pages. doi:10.1201/9781003398134

[38] Cetin Kaya Koç, Funda Özdemir, and Zeynep Ödemiş Özger. 2021. *Partially Homomorphic Encryption* (1st ed.). Springer, Cham, Zug, Switzerland. 146 pages. doi:10.1007/978-3-030-87629-6

[39] Udo Kroon. 2013. Ma3tch: Privacy and knowledge: 'Dynamic networked collective intelligence'. In *2013 IEEE International Conference on Big Data*. 23–31. doi:10.1109/BigData.2013.6691683

[40] Mehmet Kuzu, Murat Kantarcioglu, Elizabeth Durham, and Bradley Malin. 2011. A Constraint Satisfaction Cryptanalysis of Bloom Filters in Private Record Linkage. In *Privacy Enhancing Technologies - 11th International Symposium, PETS*. 226–245. doi:10.1007/978-3-642-22263-4_13

[41] Mike Levi and Melvin Soudijn. 2020. Understanding the Laundering of Organized Crime Money. *Crime and Justice* 49, 11 (2020), 579–632. doi:10.1086/708047

[42] Felipe Lillo and Rodrigo Valdés. 2016. Dynamics of financial markets and transaction costs: A graph-based study. *Research in International Business and Finance* 38 (2016), 455–465. doi:10.1016/j.ribaf.2016.07.024

[43] Ministerie van Justitie en Veiligheid. 2024. *Wet van 19 juni 2024 tot regels omtrent gegevensverwerking door samenwerkingsverbanden (Wet gegevensverwerking door samenwerkingsverbanden)*. https://zoek.officielebekendmakingen.nl/stb-2024-198.html

[44] Mitchell-24. 2026. *Repository - Privacy Preserving Entity Cycle Detection for Decentralised Anti-Money Laundering*. Github. https://github.com/Mitchell-24/Privacy-Preserving-Entity-Cycle-Detection-for-Decentralised-Anti-Money-Laundering

[45] Michael Mitzenmacher. 2009. *Bloom Filters*. Springer, Boston, Massachusetts, United States, 252–255. doi:10.1007/978-0-387-39940-9_751

[46] National Institute of Standards and Technology. [n. d.]. *Compute Security Research Center glossary - personally identifiable information*. Retrieved 2026-04-17 from https://csrc.nist.gov/glossary/term/personally_identifiable_information

[47] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT '99*, Vol. 1592. 223–238. doi:10.1007/3-540-48910-X_16

[48] Pascal Paillier and David Pointcheval. 1999. Efficient Public-Key Cryptosystems Provably Secure Against Active Adversaries. In *ASIACRYPT'99*, Vol. 1716. 165–179. doi:10.1007/978-3-540-48000-6_14

[49] Célio Porsius Martins. 2023. *Private cycle detection in financial transactions*. Master's thesis. Delft University of Technology, Delft, The Netherlands. Advisor(s) Zeki Erkin. Retrieved 2026-02-05 from https://resolver.tudelft.nl/uuid:1ebcedc4-85cc-42e8-912e-2b3e8b9603cc

[50] Privacy Pillar. 2024. *Financial Data Privacy: A Guide for Business Owners and Privacy Professionals*. Retrieved 2026-04-21 from https://privacypillar.com/financial-data-privacy/

[51] Xiafei Qiu, Wubin Cen, Zhengping Qian, You Peng, Ying Zhang, Xuemin Lin, and Jingren Zhou. 2018. Real-time constrained cycle detection in large dynamic graphs. In *Proceedings of the VLDB Endowment*, Vol. 11. 1876 – 1888. doi:10.14778/3229863.3229874

[52] Quantexa. 2026. *What is Entity Resolution and How Does It Transform Data Into Value?* Retrieved 2026-05-04 from https://www.quantexa.com/resources/entity-resolution-guide/

[53] Thilina Ranbaduge, Dinusha Vatsalan, and Ming Ding. 2023. Privacy-Preserving Deep Learning Based Record Linkage. *IEEE Transactions on Knowledge and Data Engineering* 36, 11 (2023), 6839–6850. doi:10.1109/TKDE.2023.3342757

[54] Rijksoverheid. 2026. *Besluit om banken toegang te geven tot de Basisregistratie Personen in consultatie*. Retrieved 2026-04-16 from https://www.rijksoverheid.nl/actueel/nieuws/2026/01/26/besluit-om-banken-toegang-te-geven-tot-de-basisregistratie-personen-in-consultatie

[55] Ripjar. 2022. *The Importance of Data and Information Sharing to Enhance AML Measures*. Retrieved 2026-04-20 from https://ripjar.com/blog/the-importance-of-data-and-information-sharing-to-enhance-aml-measures/

[56] Alex Sangers, Maran van Heesch, Thomas Attema, Thijs Veugen, Mark Wiggerman, Jan Veldsink, Oscar Bloemen, and Daniël Worm. 2019. Secure Multiparty PageRank Algorithm for Collaborative Fraud Detection. In *Financial Cryptography and Data Security: 23rd International Conference*. 605–623. doi:10.1007/978-3-030-32101-7_35

[57] Akrati Saxena, Yulong Pei, Jan Veldsink, Werner van Ipenburg, George Fletcher, and Mykola Pechenizkiy. 2022. The banking transactions dataset and its comparative analysis with scale-free networks. In *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. 283–296. doi:10.1145/3487351.3488339

[58] Rainer Schnell, Tobias Bachteler, and Jörg Reiher. 2009. Privacy-preserving record linkage using Bloom filters. *BMC Medical Informatics and Decision Making* 9, 41 (2009). doi:10.1186/1472-6947-9-41

[59] Paul Allan Schott. 2003. *Reference Guide to Anti-Money Laundering and Combating the Financing of Terrorism*. Technical Report. The World Bank. https://documents.worldbank.org/en/publication/documents-reports/documentdetail/982514168340180508

[60] Tom Sullivan. 2025. *What KYC is and why it matters in financial services*. Plaid. Retrieved 2026-05-30 from https://plaid.com/resources/banking/what-is-kyc/

[61] Ishan Surana. 2024. *Paillier - Cryptosystems documentsation*. cryptosystems.readthedocs.io. Retrieved 2026-05-07 from https://cryptosystems.readthedocs.io/en/latest/asymmetric/Paillier.html

[62] Mikkel Thorup. 2025. *What Is Bank Secrecy, And Does It Still Exist?* Expat Money. Retrieved 2026-04-21 from https://expatmoney.com/blog/what-is-bank-secrecy-and-does-it-still-exist

[63] United Nations Office on Drugs and Crime. [n. d.]. *Money Laundering*. Retrieved 2026-02-05 from https://www.unodc.org/unodc/en/money-laundering/overview.html

[64] Marie Beth van Egmond, Vincent Dunning, Stefan van den Berg, Thomas Rooijakkers, Alex Sangers, Ton Poppe, and Jan Veldsink. 2025. Privacy-Preserving Anti-money Laundering Using Secure Multi-party Computation. In *Financial Cryptography and Data Security: 23rd International Conference*, Vol. 2. 331–349. doi:10.1007/978-3-031-78679-2_18

[65] Dinusha Vatsalan, Ziad Sehili, Peter Christen, and Erhard Rahml. 2017. *Privacy-Preserving Record Linkage for Big Data: Current Approaches and Research Challenges*. Springer, Cham, Zug, Switzerland, 851–895. doi:10.1007/978-3-319-49340-4_25

[66] Jelle Vos, Jorrit van Assen, Tjitske Koster, Evangelia Anna Markatou, and Zekeriya Erkin. 2024. On the Insecurity of Bloom Filter-Based Private Set Intersections. Cryptology ePrint Archive, Paper 2024/1901. https://eprint.iacr.org/2024/1901

[67] Mark A. Will and Ryan K.L. Ko. 2015. *A guide to homomorphic encryption*. Syngress, Waltham, Massachusetts, United States, 101–127. doi:10.1016/B978-0-12-801595-7.00005-7