Securely Characterising Fraudulent Cycles in Financial Graphs

Rodrigue Vande Capelle[1]
Supervisor: Dr. Zekeriya Erkin[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Rodrigue Vande Capelle
Final project course: CSE3000 Research Project
Thesis committee: Dr. Zekeriya Erkin, Dr. Nezihe Merve Gürel

Abstract

With between 800 billion and 2 trillion US dollars laundered annually, which is up to 5% of the global GDP, money laundering undermines international financial systems. Crucially, these crimes are often intertwined with organised crime and violent illicit operations. Effectively identifying these illicit activities across transactional networks presents a critical dilemma: privacy regulations restrict data sharing, yet distributed data hinders fraud detection. Although graph-based strategies identify transaction cycles while respecting privacy regulations, existing methods cannot evaluate whether a cycle is benign or illicit before disclosing it. Many transactional cycles are benign, and revealing them raises privacy concerns. This paper addresses this gap by integrating the additive homomorphic Paillier cryptosystem into a pre-existing cycle detection propagation routine. Our protocol enables distributed institutions to sequentially aggregate per-node risk metrics without deciphering intermediate states. This allows nodes to evaluate collective cycle legitimacy homomorphically before revealing full cycles. We provide a theoretical evaluation of the framework's privacy guarantees and space-time complexity under an honest-but-curious adversary model. Finally, we benchmark the implementation with a parallelised C++ implementation evaluated against a synthetic scale-free graph. While the protocol successfully achieves on-demand structural risk characterisation, the integration of 2048-bit Paillier operations introduces a substantial computational overhead of up to two magnitudes more than the baseline.

1 Introduction

Financial crimes, which encompass theft, fraud, money laundering, and corruption, are often tied to violence. They undermine global financial systems and can cause major financial losses to businesses and individuals alike [1], [2]. The United Nations Office on Drugs and Crime (UNODC) estimates that between 800 billion and 2 trillion US dollars are laundered annually, which is equivalent to 2 to 5% of the global GDP [3].

Detecting these illicit activities is crucial. However, they frequently involve multiple financial institutions [4], [5]. This presents a significant obstacle, as effective detection and prevention often require data sharing among multiple parties. Because financial data is highly sensitive and strictly regulated, organisations face a fundamental dilemma: sharing data introduces severe privacy and compliance risks, whereas keeping data secret hinders the collective ability to detect illicit activity [6], [7].

To solve this impasse, Privacy Enhancing Technologies (PETs) have been developed to enable collaborative data processing without revealing the underlying raw data [8]. In this context, privacy-preserving financial crime detection implies that multi-party detection mechanisms ensure that participants learn nothing about the global financial network beyond what can be inferred from their own inputs and the final output of the computation.

According to the UNODC, money laundering typically progresses through three stages: placement (moving illicit funds from direct association with the crime), layering (obscuring the origin of the money through complex transactions), and integration (reintroducing the laundered funds into the legitimate economy) [3]. Traditional anti-money laundering (AML) methods often rely on rule based approaches. These are simplistic, need to be kept up-to-date, and are prone to high false-positive rates [9]. To overcome these limitations, recent research leverages graph-based approaches, as laundering activities naturally generate specific structural topologies. Common patterns include nodes with high out or in traffic, cliques or dense subgraphs, and cycles [10]. Cycle patterns are particularly indicative of fraud when the amount transferred over them is nearly constant [10].

How can we characterise detected cycles in privacy-preserving financial crime detection?

While existing privacy-preserving methodologies offer frameworks for identifying cycles within distributed financial graphs [11], [12], [13], they lack the ability to distinguish between fraudulent and ordinary cycles without revealing the cycle.

This paper addresses this gap by extending and modifying existing privacy-preserving algorithms. Specifically, we build upon the framework introduced by Jense [11]. This paper introduces a framework which utilises homomorphic encryption to securely pass a risk value along a graph cycle. While individual nodes compute local risk using transactional data, a standardised, effective risk value requires proprietary financial analysis; our protocol enables the collective calculation of cycle legitimacy while preventing intermediate nodes from deciphering this risk value. Based on this approach, our paper makes the following contributions:

Algorithmic Framework: We propose a new protocol extension that adds characterisation capabilities to topological cycle detection, allowing institutions to evaluate the financial risk before deciding to expose the cycle itself.

Privacy-preserving Integration: We integrate the Paillier cryptosystem into the baseline protocol's message echo propagation. By adding homomorphically encrypted risk values onto the existing echo routine, we enable secure risk aggregation.

Theoretical and Empirical Performance Analysis: We provide a theoretical analysis of privacy and space-time complexity, as well as a C++ implementation to evaluate the trade-offs introduced by the 2048-bit Paillier operations.

The remainder of this paper is structured as follows. Section 2 provides the theoretical background relevant to the paper. Section 3 reviews related work on transactional network characterisation and private cycle detection. Section 4 details our algorithm and its assumptions, followed by a theoretical analysis and experimental results in Section 5. Section 6 discusses the security and practical implications of our findings, while Section 7 addresses the ethical considerations. Finally, Section 8 concludes the paper and outlines future work. A summary of the notation used throughout the paper is provided in Appendix A; the symbols therein apply unless stated otherwise.

2 Background

2.1 Graph Theory and Transactional Cycle

Financial data can be considered a graph since money flows from account to account. In this context each account can be considered a node and each transaction an edge in this graph. We model this financial network as a simple weighted directed graph, $G = (V, E, W)$, where V represents the set of vertices (entities such as bank accounts), E the set of directed edges such that $(v_i, v_j) \in E$ if a transaction from v_i to v_j exists, and $W : E \to \mathbb{R}^+$ maps an edge to a positive real number representing the transaction amount. The term *simple* imposes a strict constraint against self-loops; an entity cannot have a transaction with itself such that if $v_i = v_j$ then $(v_i, v_j) \notin E$ and prevents more than one edges with the same source and target node. The neighbours of a vertex v_i are given by $N(v_i) \subset V$, and are the set of vertices that share an edge with v_i. $N^+(v_i)$ is used to denote the neighbours to which v_i has an outgoing edge and $N^-(v_i)$ denotes those with an incoming edge.

We can now define a *transaction cycle* of length k (where $k \geq 2$) as a sequence of distinct edges $(e_1, e_2, ..., e_k)$ connecting a sequence of vertices $(v_1, v_2, ..., v_k, v_1)$ that forms a simple, directed closed loop. This requires that transactions flow sequentially in a uniform direction where $(v_i, v_{i+1}) \in E$ for all $1 \leq i \leq k$. A *simple* cycle also requires that no vertex or edge is visited more than once, such that $v_i \neq v_j$ for all distinct indices $i, j \leq k$, except for the first and last vertex. By formally including edge weights $W(e)$ in this topological definition, privacy preserving cycle detection algorithms can be extended to securely evaluate financial risk metrics throughout the loop.

2.2 Homomorphic Encryption

Homomorphic encryption (HE) is a cryptographic technique that allows computations to be performed directly on encrypted data without revealing the underlying plaintext [14]. We use this in the context of this paper to characterise cycles and modify risk values while preserving confidentiality. HE schemes are commonly divided into three categories: Partially Homomorphic Encryption (PHE), which supports a single operation (addition or multiplication) an unlimited number of times; Somewhat Homomorphic Encryption (SHE), which supports both operations a limited number of times; and Fully Homomorphic Encryption (FHE), which supports both operations without such limitations [15]. However, FHE is computationally more expensive than PHE [16].

This paper utilises the Paillier cryptosystem, an additive PHE scheme [17]. For our algorithm, this is the most practical choice, as it allows an entity to aggregate calculated and received values an arbitrary number of times. Furthermore, despite not being FHE, it supports the addition of plaintexts and multiplication of a plaintext by a constant through operations on ciphertexts. The homomorphic properties that are used in this work are presented below.

$$\begin{aligned} \text{Enc}_{PK}(m_1) \cdot \text{Enc}_{PK}(m_2) &= (g^{m_1} r_1^n \bmod n^2)(g^{m_2} r_2^n \bmod n^2) \\ &= g^{m_1+m_2}(r_1 r_2)^n \bmod n^2 \\ &= \text{Enc}_{PK}(m_1 + m_2), \end{aligned} \tag{1}$$

where $\text{Enc}_{PK}(m)$ denotes the encryption of the plaintext message m with the public key PK. In this subsection, n is the RSA modulus (the product of two large primes p and q), g is a public base integer chosen from $\mathbb{Z}^*_{n^2}$ and $r_1, r_2 \in \mathbb{Z}^*_n$ are random blinding factors ensuring the probabilistic nature of the encryption. This is important as it gives Paillier the property of Indistinguishability Under Chosen Plaintext Attack (IND-CPA) [17], meaning that an attacker, given two Paillier ciphertexts, cannot distinguish which plaintext was encrypted with probability significantly better than random guessing [18].

Although not explicitly used in this paper, another useful feature of Paillier is that it supports the multiplication of an encrypted plaintext by a known constant through ciphertext exponentiation:

$$\text{Enc}_{PK}(m)^k \bmod n^2 = \text{Enc}_{PK}(mk \bmod n), \tag{2}$$

where $k \in \mathbb{Z}_n$ represents the scalar constant factor applied to the message.

These homomorphic properties make the Paillier cryptosystem suited for privacy-preserving computations involving additions and scalar multiplications on encrypted data [15].

2.3 Privacy Preserving Cycle Detection

The goal of this paper is to build upon existing privacy preserving cycle detection methods and make them more informative. As mentioned, this paper builds upon Jense's paper [11]. This section will explain how Jense's algorithm functions and define some important symbols from their work. The algorithm relies on the Decisional Diffie-Hellman (DDH) assumption, a computational hardness assumption that ensures intermediate nodes cannot distinguish routing keys from random data, thereby preserving privacy during the cycle detection [19], [20]. Specifically, a node starts a key exchange with itself through its neighbours; if a cycle exists, the node receives its own message back and completes the key agreement.

For the algorithm, let $G = (V, E)$ represent the network graph. We assume that all nodes agree upon a group $\mathbb{G} \subseteq \mathbb{Z}_p^*$ of order q where the DDH problem is hard, and that the generator g of $\mathbb{G}$ is publicly known. Following the paper, for arithmetic in $\mathbb{G}$, the explicit modular notation is omitted for simplicity. The algorithm uses κ_p, κ_q, and κ_r as security parameters, corresponding to the bit-lengths of the primes p and q, as well as the nonces generated in the protocol.

In more detail: the solution works in four stages: initiating the protocol, propagating forward messages, echoing responses, and tracing cycles. This subsection will explain the basic idea behind the protocol.

When a node, $v_i \in V$ wants to initiate the cycle detection protocol, it executes `initiate`, which takes ℓ as input to determine the maximum length of cycles to detect. It sends a propagating message to all its outgoing neighbours with ℓ - 1. For each of its outgoing neighbours $v_j \in N^+(v_i)$, it picks a private key x_j and nonce r_j. x_j is used to compute the partial public key g^{x_j}. The message sent to v_j is $(r_j, g^{x_j}, \ell - 1)$.

The next algorithm, `propagate`, is initiated when a node v_j receives a propagating message, $(r_j, g^x, \ell - 1)$. Firstly, the node sends an echo message back to the node by picking a private key y from $\mathbb{Z}_q^*$, then calculating g^y and g^{xy}. The node stores the key g^{xy} in the data structure `keys`, and sends an echo back to v_i with the message $(r_{i,j}, g^y)$. This process eventually helps the initiating node detect whether a cycle has occurred. Afterwards, if ℓ has not reached 0, then the node sends a propagating message to its outgoing neighbours with ℓ - 1. This propagating message for v_j is constructed in a similar manner to that of the initiating node. The node picks two random values for each node $v_n \in N^+(v_j)$. A nonce $r_{j,n} \in \{0,1\}_{\kappa_r}$, and an intermediate secret key $k_{j,n} \in \mathbb{Z}_q^*$ are randomly chosen. The data structure `routes` stores a tuple for each v_n. The tuple contains v_i, its associated nonce, v_n, $r_{j,n}$, and the intermediate secret key. This allows the node to track where messages came from and should go. The node constructs the message $(r_{j,n}, g^{xk_{j,n}}, \ell - 1)$ for each v_n and sends it to each v_n.

Upon receiving an echo message, a node performs the `echo` routine, which does one of two things. The first case is that the node is the initiating node and it is expecting an echo from this node. In this case, the node verifies whether a cycle has been detected by checking if the key exchange has been performed. If it has, then it starts the `trace` routine. In the other case, where it is not the initiating node or did not perform a key exchange with itself, it continues to forward the echo backward by checking its routes.

The final algorithm is the `trace` function. This algorithm sends a message along the cycle to discover which nodes are participating in the cycle.

To summarise the algorithms put together: the initiate and propagating aspects of the protocol help explore the graph, the echoes help the initiating node determine whether a cycle has been detected, and finally, the trace aspect helps reconstitute the nodes part of the cycle. Appendix B contains an example figure of how the algorithm behaves.

3 Related Work

3.1 More Privacy-Preserving Cycle Detection Methods

The cycle detection algorithm this paper is based on is not the only available method. One privacy-preserving cycle detection method uses one-time pads (OTPs) to detect cycles by causing them to cancel out after going around a cycle twice. Once the cancellation occurs, the initiating entity can determine a cycle has been detected. However, the method has been shown to reveal information about the cycle structure to intermediate nodes, and we therefore do not discuss it further [11].

Another framework for privacy-preserving cycle detection is Oryx [12], which is tailored to federated graphs (graphs split across multiple institutions). To protect privacy, Oryx relies on two or more honest-but-curious, non-colluding servers and uses secure multi-party computation (MPC), a cryptographic framework allowing multiple parties to jointly compute a function over private inputs without revealing data beyond the output [21].

Oryx operates in three stages. It begins with an initialisation stage where data-holding parties distribute secret shares of their local subgraphs to the computing servers, ensuring that intermediate graph states remain hidden. In the second stage, the servers iteratively execute an oblivious path extension protocol in rounds; to detect cycles of length k, they extend the secret-shared paths of length $k-1$ using the outgoing neighbour nodes of their terminal nodes. In the third stage, the servers filter out invalid paths (such as those containing repeating or dummy nodes), detect and reveal completed cycles, and format the valid paths to serve as the baseline for length $k+1$ path extensions in the next round.

Given v as the total number of vertices, n the average number of neighbours, and d the maximum degree, when detecting a cycle of length k, the total number of subgraphs to process is $T = vn^{k-1}$. The computational complexity for that round is $O(kT(d + \log(T)))$. Oryx is capable of detecting all cycles up to length 6 in under 5 hours for a financial graph with tens of millions of nodes and edges.

The authors note that their techniques significantly reduce computational complexity in comparison to generic MPC and prior works, but the protocol still requires substantial communication between the computation servers. While Oryx makes private cycle detection on large federated graphs practical, it follows a client-server MPC model in which dedicated computation servers perform the cycle detection over secret-shared graph data. As a result, the protocol is designed to detect cycles across the entire distributed graph rather than to support on-demand investigations initiated by a single institution for a specific suspicious node.

3.2 Flow Detection Problems

To the best of our knowledge, there are no existing works that combine privacy-preserving cycle detection with algorithmic enhancements to characterise these cycles. Instead, current literature treats these objectives separately. For instance, there has been work focused on reducing the false positives and false negatives of cycle detection algorithms operating directly on plaintext data [22]. In this paper, they reduce false negatives by eliminating constraints on both the maximum cycle length and the duration of the transaction time-window. Unlike generic cycle detection methods, their approach is tailored to financial fraud analysis by focusing strictly on simple temporal cycles, ensuring that bank accounts are not repeated within a transaction chain and that transaction timestamps strictly increase chronologically from one edge to the next.

Their algorithm extends the Johnson algorithm for simple directed cycle enumeration [22], [23]. To overcome the performance limitations of standard depth-first search (DFS) methods, Zheng et al. introduced a block-time mechanism to prune redundant or unsuccessful traversal paths. This mechanism tracks the timestamp attributes of visited nodes and uses this data to prune subsequent exploration paths that cannot complete a valid temporal cycle. The integration of this pruning technique with concurrent, multi-threaded architecture runs 15 to 20 times faster on average than the baseline approach by Hajdu et al [24]. The baseline uses a DFS strategy that verifies every transaction edge individually and yields non-simple cycles.

The protocol of Zheng et al. has an initial setup phase where transactions with small amounts are filtered out based on an expert defined threshold to eliminate regular commercial noise. The algorithm then constructs a directed transaction graph. As a performance improvement, it separates the graph into strongly connected components (SCCs) by removing transaction edges that cannot form cycles using Tarjan's algorithm [25]. This separation permits the use of multiple threads to perform localised cycle detection on each of the components individually.

To separate structured suspicious activity from normal behaviour, the algorithm imposes a transaction amount constraint during enumeration. This rule requires that amounts from all subsequent transaction edges in a cycle remain tightly bounded relative to the amount of the very first transaction edge. While the authors report that varying the initial minimum amount filter can alter the volume of detected cycles by up to a factor of 10, the paper does not provide an explicit experimental or statistical evaluation of how effectively either of these constraints correlates with true false-positive or false-negative fraud rates.

3.3 Secure Value Passing

This paper was in part inspired by another AML method that propagates a risk value through transaction graphs in a privacy-preserving way [26]. Each account is initially assigned a risk score. This risk score can be modified based on need, but for their experiments, each account was assigned a risk score based on the ratio of cash added to their account. Risk scores are propagated through the network based on the source of money being sent to each account. It uses multiple iterations of propagation to mimic the depth of the layering phase, such that illicit money flow can be followed. To propagate these values in a privacy-preserving manner, additive homomorphic encryption was used, specifically Paillier.

While the approach by van Egmond et al. efficiently tracks the fluid movement of risk across distributed banking networks, it does not exploit explicit topological structures. Linear risk propagation can propagate scores through a cycle, but it does not inherently isolate the shape. We adopt their core concept of utilising additive homomorphic encryption to pass and accumulate risk metrics across distributed financial graphs. In our algorithm, we adapt this mechanism to securely evaluate the legitimacy of explicit, bounded-length topological cycles.

4 Characterising Cycles with a Value

4.1 Security Model and Assumptions

Our algorithm is designed under certain assumptions regarding potential threats, requirements, and system availability. We assume an honest-but-curious adversary model. All

participating entities strictly follow the prescribed protocol, provide correct inputs, adhere to the computation process, and do not collude. However, participants may try to learn information by analysing the data they receive during the execution of the algorithm. We also assume a trusted third party responsible for public key distribution and decryption with the private key.

In terms of system availability, we assume there is a reliable bidirectional communication channel between all participants. The system is designed to work under standard network conditions and assumes that all nodes remain available throughout the execution of the protocol to ensure the completion of the computation.

We also assume that the underlying cryptographic primitives and algorithms we use, such as the cycle detection algorithm and homomorphic encryption, are computationally secure against a polynomially bounded adversary.

4.2 Protocol Architecture

For the algorithm presented in this section, let $G = (V, E, W)$ as defined in 2.1. We also keep the same notation Jense used in his paper, which we define in 2.3.

The new parameters we define are as follows: $\text{Enc}_{PK}(m)$, which corresponds to the generic homomorphic encryption function using the public key PK, $\text{Dec}_{SK}(c)$, which is the decryption function of the ciphertext c using the secret key SK. For this algorithm, we assume a trusted third party that has SK and distributes PK securely to all parties involved beforehand. We introduce `risk`, a function which calculates a node's risk score based on the transaction data available to that node. Accurate risk assessment requires domain-specific models and extensive proprietary analysis to minimise false positives. Additionally, we have the function `Aggregate` which combines the received risk value, c, with the risk value of the local node v_i, $\texttt{risk}(v_i)$. When using addition in Paillier for aggregation, this is equivalent to performing the homomorphic addition of $\text{Enc}_{PK}(\texttt{risk}(v_i))$ and c as per Equation 1. Finally, the function `Rerandomise` is responsible for ensuring that c and the aggregated risk value are not linkable. For Paillier this is not necessary so $\texttt{Rerandomise}(c) = c$. We still include this in the pseudocode to take into consideration when using other HE schemes.

The idea of the algorithm is to have the echo messages forward a risk value backward, allowing the intermediary nodes to aggregate the value together such that the final value is $\sum_{v_i \in V_c} \text{risk}(v)$, where risk represents the function to calculate the risk value and V_c are the nodes of a particular cycle. To prevent intermediary nodes from being able to read the value and revealing more information than required, we use homomorphic encryption to aggregate the values and only decrypt the value once a cycle has been detected. This is done by modifying the `propagate` and `echo` routines, while the `initiate` and `trace` routines remain as described in Section 2.3.

4.3 Modifying Propagate

We start by modifying the `propagate` routine, as shown in Algorithm 1, by adding a homomorphically encrypted 0, $\text{Enc}_{PK}(0)$, that will be sent along with the echo message. This modification alters the echo message for a node v_j sent to a node v_i into $m' = (r_{i,j}, g^y, c_{local})$. The initiating vertex encrypts its initial contribution with 0 because it only possesses knowledge of its immediate outgoing edge; the subsequent path validation relies on adjacent nodes sequentially accounting for the linking edges within the loop.

Algorithm 1 v_j receives forwarded message from v_i

propagate$(m = (r_{i,j}, g^x, \ell))$

1: $y \in_R \mathbb{Z}_q^*$
2: **append** g^{xy} **to** keys
3: $c_{local} = \text{Enc}_{PK}(0)$
4: **send** $m' = (r_{i,j}, g^y, c_{local})$ **to** v_i ▷ echo message
5: **if** $\ell == 0$ **then return**
6: **for** $v_n \in N^+(v_j)$ **do**
7: $r_{j,n} \in_R \{0,1\}^{\kappa_r}$
8: $k_{j,n} \in_R \mathbb{Z}_q^*$
9: **append** $(v_i, r_{i,j}, v_n, r_{j,n}, k_{j,n})$ **to** routes
10: **send** $m'' = (r_{j,n}, g^{x \cdot k_{j,n}}, \ell - 1)$ **to** v_n ▷ forward message

4.4 Modifying Echo

The `echo` routine has been modified as per Algorithm 2. When a node, v_i, receives an echo message, $(r_{i,j}, g^y, c)$, from v_j the routine first checks if `pending` contains an entry with the nonce $r_{i,j}$. This match occurs only if it is the initiating node and one of its outgoing neighbours echoes back a message related to the cycle detection instance it started. In this scenario, the node retrieves $(x, r_{i,j})$ from `pending`. It then checks whether a cycle has been detected by verifying if g^{xy} is in `keys`. If it is, a cycle has been detected and the node decrypts the value, compares it to a threshold value, and only performs the `trace` routine if the value is above the threshold. In the case where it is not an initiating node or `pending` does not contain $(\_, r_{i,j})$, the vertex calculates its own risk value and homomorphically aggregates it with the received echo value. This value, `risk`(v_i), is subsequently aggregated with c using the appropriate operation and randomised to prevent linkability between two messages. This aggregated value is then forwarded back to the next node v_n based on the route fetched using $r_{i,j}$. The final result of this pipeline is that all intermediate nodes calculate the aggregate value based on all the edges comprising the loop.

Algorithm 2 v_i receives an echoed message from v_j

echo$(m = (r_{i,j}, g^y, c))$

1: **if** pending **contains** $(-, r_{i,j})$ **then** ▷ Ignore x when checking `pending` for $r_{i,j}$
2: **fetch** $(x, r_{i,j})$ **from** pending
3: **if** keys **contains** g^{xy} **then**
4: Value $= \text{Dec}_{SK}(c)$ ▷ c is sent to trusted party for decryption
5: **if** Value $\geq$ Threshold **then**
6: **send** $m' = (r_{i,j}, g^y, [v_i])$ **to** v_j ▷ trace message
7: **else**
8: **append** g^{xy} **to** keys
9: **else**
10: **fetch** $(v_n, r_n, v_j, r_{i,j}, k_{i,j})$ **from** routes
11: $c_{updated} = \text{Aggregate}(c, \text{risk}(v_i))$ ▷ Homomorphic accumulation
12: $c_{final} = \text{Rerandomise}(c_{updated}, PK)$
13: **append** $(r_n, v_j, r_{i,j}, k_{i,j}, g^y)$ **to** retrace
14: **send** $m'' = (r_n, g^{y \cdot k_{i,j}}, c_{final})$ **to** v_n ▷ echo message

5 Analysis

5.1 Security Analysis

In this subsection, we provide insight into the security of the amended algorithm presented in this paper under certain constraints. We build upon the work of the original paper [11], addressing only privacy changes in relation to the addition of homomorphism. We reuse a setup similar to that of the original paper. We assume that the group $\mathbb{G}$ is publicly known, that network traffic is stable and cannot be tampered with, that communication channels are asynchronous, and that they can be tapped. Internal processes and memory of honest nodes are hidden. We also assume that the nodes are honest-but-curious adversaries, as defined in Section 4.1.

An important consideration in this algorithm is a node's ability to determine whether two messages are part of the same instance. Messages are part of the same instance if they share the same initiating node. Two messages are considered unlinkable if an adversary does not have a significant advantage over randomly guessing whether two messages are part of the same instance, except where the link between two messages is implied. The link between two messages is implied for input messages and the corresponding output messages, as well as for forwarded messages and their corresponding echo messages. Furthermore, a node should not have a significant advantage over randomly guessing which value it receives from a node and what value it echoes back (except when decrypting the final risk value, as specified by the protocol).

In the original paper [11], it is shown that two messages $m_1 = (r_1, g^{x_1})$ and $m_2 = (r_2, g^{x_2})$, are unlinkable through their nonces r_1 and r_2 as they are selected uniformly at random. The authors also prove that linking the messages using g^{x_1} and g^{x_2} is not possible. We now show that the homomorphically encrypted risk values introduced by our protocol preserve this unlinkability.

The probabilistic nature of Paillier encryption, described in Section 2.2, implies that two freshly encrypted ciphertexts cannot be distinguished, making arbitrary messages containing them unlinkable. To show unlinkability in the presence of messages observed during protocol execution, let c_{h1} and c_{h2} be two homomorphically encrypted values, and let c_{agg} denote their homomorphic sum. Consider two messages, one containing c_{h1} and the other c_{agg}. A node cannot determine whether c_{h1} contributed to c_{agg}; thus the messages are unlinkable. Since c_{h2} is encrypted probabilistically and homomorphic addition in Paillier is performed via ciphertext multiplication, no operation applied to c_{h1} and c_{agg} reveals useful information about their relationship. This follows from the IND-CPA security property of Paillier [27].

Although collusion and protocol deviation are excluded from the security model, it is still important to consider their potential impact. In particular, since a trusted party holds the private decryption key, a malicious node could deviate from the protocol and request the decryption of intermediate homomorphic values. Given that each node maintains a nonce for every propagated message, such a node could attempt to reconstruct additional edge values in the transactional network and infer whether certain values are within x hops of its position. In the simplest case, the node could iteratively query the decryption of aggregates (e.g., obtaining a sequence of partial sums starting from zero). However, this attack is mitigated by the fact that each message propagation induces multiple subsequent propagations from neighbouring nodes, thereby generating overlapping aggregates. These overlapping responses, combined with the reuse of nonces across propagation paths, prevent the adversary from cleanly attributing decrypted values to a single propagation chain.

5.2 Complexity Analysis

To evaluate the complexity of the algorithm, we need to take into consideration the space, time, and communication complexities. We calculate the complexity as per the pseudocode presented in Section 4. We will also assume that a κ_h-bit Paillier cryptosystem is used and that aggregation performed given a message $m = (r_{i,j}, g^y, c)$ and a locally calculated risk value, $\texttt{risk}(v_i)$, is the homomorphic addition of $\text{Enc}_{PK}(\texttt{risk}(v_i))$ and the received value c. The total number of nodes is represented as n. When using Paillier, the function `Rerandomise` is not necessary since we are always performing operations on ciphertexts and encryption with Paillier is non-deterministic. A summary of the complexity analysis for one round can be found in Appendix C. This complexity analysis for `trace` is omitted as it is the same as for the original algorithm and is insignificant compared to the routines like `echo`, and `propagate`. When calculating to run the algorithm for all nodes, multiply the provided values by n.

Space Complexity

The space complexity of the algorithm presented in this paper does not change compared to the original paper. The result of the homomorphic encryption is never stored by any node. Once the value has been used to determine whether the cycle is considered suspicious, it is no longer used, and is not saved. Therefore, the storage complexity depends on the security parameters κ_p, κ_q, and κ_r which control the bit length of p, q, and the nonces, respectively.

The worst case scenario for space complexity is when the graph is fully connected. In this case, when calling the algorithm `initiate` for a single node, each node needs to store κ_q and κ_r for each of the $n-1$ neighbours which is $O(n(\kappa_q + \kappa_r)$. For the `propagate` routine, the algorithm adds values to the data structure `routes`. This is of length $O(\kappa_q + 2\kappa_r)$ bits for the key and two nonces that are stored. The bit-length of the nodes is insignificant in comparison. Since the algorithm is called $O(n^\ell)$ times, the space complexity for `propagate` alone is $O(n^\ell(\kappa_q + 2\kappa_r)$. For the `echo` routine, the data stored depends on whether it is the initiator or not. In the case that the node is the initiator, it stores the shared key, which is κ_p bits. When it is not the initiator, it stores $O(2\kappa_r + \kappa_q + \kappa_p)$ for the two nonces, the private key, and the public key. Since the `echo` routine is called for each `propagate` routine and further propagates until the initiating node (which is ℓ hops away), it is called $O(\ell n^\ell)$ times. This leads to a total space complexity of $O(\ell n^\ell(2\kappa_r + \kappa_q + \kappa_p)$, which is $O(\ell n^\ell(\kappa_r + \kappa_q + \kappa_p))$ when simplified.

Time Complexity

Time complexity follows a similar pattern. Each algorithm is run the same number of times as explained in the space complexity calculations. However, the important consideration here is the time complexity for the exponentiations per function call, the time taken for fetching tuples from data structures, and the number of homomorphic operations and their time complexity. The complexity of exponentiation per function call scales with the security parameters. For the group $\mathbb{Z}_p$, the complexity of exponentiation is $O(\log^3 \kappa_p)$ [20]. For a κ_h-bit Paillier, the time complexity for encryption and decryption is $O(\kappa_h^3)$, and for homomorphic addition it is $O(\kappa_h^2)$ [17]. For fetching tuples from the various data structures, we assume the data structures use hash maps and use the expected runtime complexity of $O(1)$. We also assume the complexity for sending and receiving a message to be $O(1)$.

The `initiate` function is unchanged and, therefore, remains of time complexity $O(n \log^3 \kappa_p)$ to run the algorithm to compute exponentiation for each neighbour.

The `propagate` function changes as it now contains the encryption of 0 using Paillier. In the propagate routine, it performs exponentiation once, which is $O(\log^3 \kappa_p)$, performs a Paillier encryption of $O(\kappa_h^3)$, and performs $O(\log^3 \kappa_p)$ operations for each of its outgoing neighbours, which is $O(n)$. The complexity for running this algorithm n^ℓ times is $O(n^\ell(n \log^3 \kappa_p + \kappa_h^3 + \log^3 \kappa_p))$, and simplified is $O(n^\ell(n \log^3 \kappa_p + \kappa_h^3))$

The `echo` routine changes in a similar fashion. This means that for one iteration of the `echo` routine, the runtime complexity is $O(\log^3 \kappa_p + \kappa_h^3)$ for exponentiation and decryption of a homomorphic value if the node is the initiating node. In the case where it is not the initiating node the runtime complexity is $O(\kappa_h^3 + \kappa_h^2 + \log^3 \kappa_p)$. As previously mentioned, the upper bound number of calls is $O(\ell n^\ell)$. In a single run of the cycle detection algorithm the number of `echo` calls for the initiating node is $O(n^\ell)$ as each call to `propagate` creates an echo chain that reaches the initiating node. Therefore, the amount of `echo` calls with a non-initiating node is $O((\ell - 1)n^\ell$. The combined complexity of running echo is $O((\ell - 1)n^\ell(\kappa_h^3 + \kappa_h^2 + \log^3 \kappa_p) + n^\ell(\log^3 \kappa_p + \kappa_h^3))$. When simplified, this is $O(\ell n^\ell(\kappa_h^3 + \log^3 \kappa_p))$.

Communication Complexity

The only change in communication complexity is within the `echo` routine. The number of echo messages communicated per initiation is $O(\ell n^\ell)$. With the addition of Paillier cryptography, the communication complexity now also depends on κ_h. Each echo message now carries a κ_h-bit value in addition to the nonce and public key. So the communication complexity for a single round is $O(\ell n^\ell(\kappa_h + \kappa_r + \kappa_p))$.

5.3 Experiments

To experimentally test the time it would take to run the algorithm with the addition of homomorphism with Paillier we compared it with the runtime of the original algorithm.

To run experiments using the new protocol, we amended the original code on which our algorithm is based [28], which is in C++ and was tested against synthetically generated graphs. This synthetically generated graph uses the Barabási-Albert model from [29]. The method of generation creates a scale-free distribution, which financial market graphs are believed to follow [30], [31]. We first made some performance improvements and used parallelisation to both improve the time it takes for the algorithm to run and to more closely represent multiple servers initialising cycle detection from different nodes. Instead of using lists for the variables `routes`, `keys`, `pending` we used hash maps, which have an $O(1)$ expected time complexity compared to $O(n)$ for lists. After adding homomorphism by taking a Paillier implementation by Google [32], we made additional performance improvements by pre-generating a list of encrypted zeroes. We also pre-encrypted powers of two, and add them to calculate the risk value that needs to be added to the message instead of generating a new encryption. This is $O(\kappa_h^2)$ instead of $O(\kappa_h^3)$ but arguably decreases privacy unless using a large pool of zeroes. Since adding powers of two from a pre-generated list is deterministic, we use a zero from the zero-pool to re-randomise the value. All experiments were run on a machine with a 13th Generation Intel Core i7-13700H (14 physical cores, 20 threads), and 16 GiB of Ram.

To include our characterisation algorithm and test its performance, we extend the synthetic model generation to randomly generate edge values and amend the C++ code to incorporate the pseudocode discussed in Section 4. While this does not help determine

whether it reduces false-positives, it gives us an idea of the performance of the algorithm. We re-use the same 'toy' values for the security variables as the original paper's experiment: $\kappa_p = 60, \kappa_q = \kappa_r = 20$. However, for Paillier we use $\kappa_h = 2048$. Current cryptographic standards recommend a minimum modulus length of 2048 bits, with 3072 bits recommended for long-term security [33].

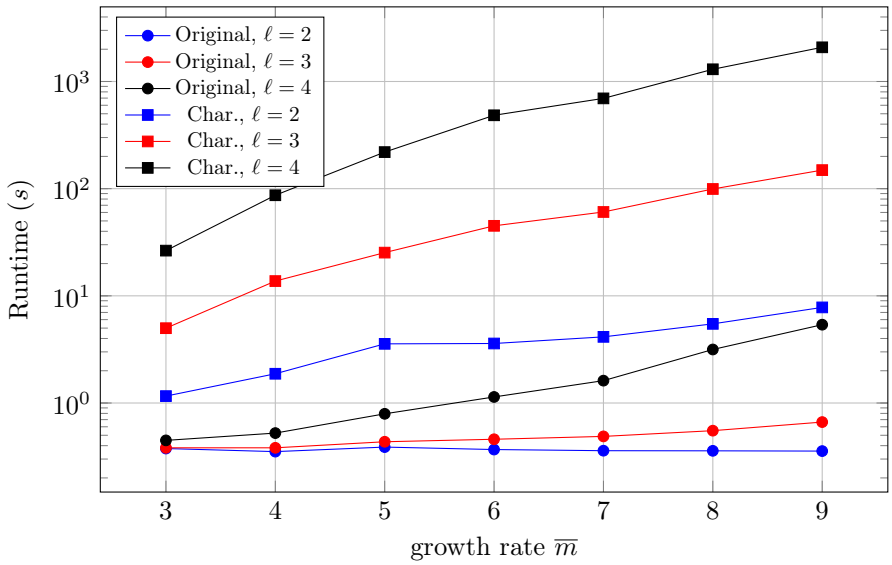


Figure 1: Relationship between the growth rate $\overline{m}$ (average node degree) and the runtime (in seconds), shown on a logarithmic scale based on the raw data available in Appendix D. Results are reported for the original implementation (with improvements) and for the characterisation approach with Paillier encryption. The experiment uses $n = 50$ and varies $\ell \in \{2, 3, 4\}$.

As shown in Figure 1, the average runtime significantly increases when value characterisation is applied. For a setup with $n = 50$ nodes, a growth rate of 9, and a maximum cycle length of $\ell = 4$, the runtime of the original algorithm is 5.4 seconds. In comparison, when homomorphic operations are introduced using 2048-bit Paillier, the runtime increases to 2084.6 seconds (34.74 minutes). This increase is substantial but expected, as Paillier encryption operations are computationally expensive, especially with a high number of bits. Given the $O(\ell n^\ell)$ number of echo messages, each requiring at least one homomorphic addition, a significant increase in runtime is anticipated. These results highlight the trade-off between early cycle characterisation and computational efficiency.

6 Discussion

6.1 Practical and Security Applicability

To the best of our knowledge, the protocol presented in this paper is the only privacy-preserving cycle detection with characterisation. While independent privacy-preserving detection and characterisation methods exist, this paper serves as a stepping stone towards combining them. The algorithm aims to provide a way to reduce false positives and avoid

publicly revealing cycles that may be non-fraudulent. Although the theoretical framework and prototype function correctly, several practical and security-related barriers must be addressed before real-world deployment.

One of the key challenges is the adversary model used. Our privacy guarantees rely on the assumption that participating nodes act as honest-but-curious adversaries. In a highly competitive or compromised financial ecosystem, this idealised scenario may not hold. The current protocol is not secure against malicious actors. The framework does not function correctly if fraudulent values are introduced during risk aggregations, if risk values are decrypted without following the protocol, or if collusion occurs among participating nodes. Furthermore, our analysis does not account for side-channel attacks. For example, network traffic analysis could reveal which nodes have connections between them.

6.2 Algorithmic Limitations

The computational complexity and empirical results in Sections 5.2 and 5.3 demonstrate that the protocol runtime scales exponentially with the target cycle length and the average node degree. The experimental results further show the computational overhead due to Paillier encryption operations. This exponential scaling presents a barrier for large-scale financial networks. Our tests were performed using 50 nodes and an node degree of up to nine, whereas real-world interbank networks regularly exceed both those values [34], [35]. While the protocol is theoretically designed for full parallelisation, our experimental evaluation was constrained by single-machine execution. Conversely, a fully distributed deployment would introduce communication latency, which were not modelled in our local experiments and require further testing.

Beyond computational constraints, the structural complexity of financial graphs and money laundering tactics presents an analytical limitation [36]. The objective of this paper to reduce false positives requires deeper investigation into exactly which data aggregations accurately signify illicit behaviour. Simply adding a risk score, such as the ratio of funds being passed sequentially along a cycle, fails to capture sophisticated laundering schemes. Refining these characterisation metrics requires collaboration with financial forensic experts.

Additionally, our framework is structurally restricted to a simple cycle pattern. Money laundering frequently relies on a variety of non-cyclic patterns, such as scatter-gather structures, high traffic, cliques, and more, that cannot be detected by performing cycle detection. The deployment of this algorithm would need to operate within a broader, multi-layered fraud detection system. Finally, the current design assumes a homogeneous currency system. Real-world international financial networks are fundamentally multi-currency, meaning future iterations must support secure, homomorphic cross-currency conversions.

7 Responsible Research

It is important to consider the broader implications of the research involving financial data due to its highly sensitive nature. In this context, revealing the mechanisms behind money laundering detection could have unintended consequences.

The point of this research is to enhance the money laundering detection while preserving data privacy. While this paper provides a foundational security analysis demonstrating the alignment of our protocol with IND-CPA properties of the Paillier cryptosystem, it lacks a mathematically formal, simulation-based security proof. Furthermore, since we adopt an honest-but-curious security model, collusion between nodes is not taken into account.

This remains a potential security threat that must be addressed in a real world scenario. Importantly, because the data used throughout this research is entirely synthetic, there is no risk of leaking genuine sensitive information.

In the detection of money laundering, revealing money laundering detection algorithms can inadvertently help criminals refine their evasion tactics. However, the research presented in this paper is strictly intended to improve existing privacy-preserving cycle detection techniques by reducing false positives. We believe that the information provided in this paper does not significantly help adversaries enhance their money laundering processes.

The reproducibility of research in computer science is important, particularly when introducing an algorithm. To ensure this, all code used for data processing and experimentation is publicly available [37]. The methods and configurations for generating the synthetic experimental data are also included within the same repository.

Another important consideration in ensuring responsible evaluation is addressing potential technical bias. The protocol proposed in this paper is evaluated exclusively against synthetically generated scale-free graphs. Because topological propagation algorithms can perform very differently depending on the density and structure of the underlying data, we include the generation code in the same repository as the algorithm. Providing this transparency ensures that future research and deployment can accurately anticipate variations in computational overhead.

8 Conclusions and Future Work

In this paper, we addressed the challenge of answering the research question, "How can we characterise detected cycles in privacy-preserving financial crime detection?". We proposed an initial framework to assign a risk value to financial cycles by modifying the privacy-preserving cycle detection algorithm introduced by Jense [11]. We showed that even though the addition of characterisation has the potential to reduce false positives, it comes with a performance trade-off. The algorithm presented in this paper utilises homomorphic encryption to allow operations such as addition to be performed directly on ciphertexts, thereby ensuring that the plaintext values remain confidential. With this method, the risk value can be securely propagated through the cycle without intermediary nodes being able to see the value. Only the node initiating the cycle detection can decrypt the value and compare it against a threshold to determine whether the cycle is likely to be fraudulent. This system allows non-fraudulent cycles to remain undisclosed, reducing the number of false-positives that need investigation.

While this framework establishes a strong foundation for risk-based, privacy-preserving graph analysis, several open challenges must be addressed before it can be deployed in real-world financial environments. We present these current limitations as strategic opportunities for future research.

First, regarding risk evaluation and data validation, an essential step is investigating which specific types of risk values can be propagated to efficiently and accurately reduce false-positives from detecting all cycles. Evaluating this will require using more accurate synthetic data such as AMLWorld [38], Tide [39], or, if accessible, real annotated transactional data.

Second, the cryptographic and security robustness must be expanded. The privacy and security of this system currently hold under an honest-but-curious adversary model; adversary nodes cannot determine the risk value of other nodes, nor can they determine if they are part of a cycle unless it is explicitly revealed. Broadening these assumptions to allow for malicious adversaries, collusion, or side channel attacks would significantly improve the

practical security of the algorithm. A specific improvement would be altering the protocol to eliminate the need for a trusted third-party for key distribution and decryption.

Finally, future work must tackle algorithmic scalability and performance. A significant limitation is both the theoretical runtime complexity and the resulting experimental runtime. The algorithm scales exponentially with respect to the maximum length of cycles, ℓ, we are trying to detect, and includes a factor that depends on the security variables κ_h and κ_p. Specifically, the runtime complexity is $O(\ell n^{\ell}(\kappa_h^3 + \log^3 \kappa_p))$ per instance. Work on either decreasing the algorithmic complexity or optimising the C++ implementation to reduce practical runtime would be immensely beneficial. For instance, a valuable optimisation would be developing a mechanism to prevent cycles shorter than $\frac{\ell}{2}$ from being detected multiple times.

References

[1] INTERPOL, *Financial crime*, Accessed: 2026-05-22, 2026. [Online]. Available: `https://www.interpol.int/en/Crimes/Financial-crime`.

[2] Europol, *Economic crime*, Accessed: 2026-06-19, 2026. [Online]. Available: `https://www.europol.europa.eu/crime-areas/economic-crime`.

[3] United Nations Office on Drugs and Crime, *Money laundering*, UNODC, 2024. Accessed: Apr. 24, 2026. [Online]. Available: `https://www.unodc.org/unodc/en/money-laundering/overview.html`.

[4] Z. Tian, Y. Ding, X. Yu, E. Gong, J. Liu, and K. Ren, "Towards collaborative anti-money laundering among financial institutions", in *Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025- 2 May 2025*, G. Long, M. Blumestein, Y. Chang, L. Lewin-Eytan, Z. H. Huang, and E. Yom-Tov, Eds., ACM, 2025, pp. 4722–4733. DOI: `10.1145/3696410.3714576`.

[5] M. Collin, "Illicit financial flows: Concepts, measurement, and evidence", *The World Bank Research Observer*, vol. 35, no. 1, pp. 44–86, Feb. 2020, ISSN: 0257-3032. DOI: `10.1093/wbro/lkz007`. eprint: `https://academic.oup.com/wbro/article-pdf/35/1/44/32448476/lkz007.pdf`.

[6] intersoft consulting services AG. "Personal data - General Data Protection Regulation (GDPR)", intersoft consulting services AG, Accessed: Apr. 24, 2026. [Online]. Available: `https://gdpr-info.eu/issues/personal-data/`.

[7] Financial Action Task Force, *Partnering in the fight against financial crime, data protection, technology and private sector information sharing*, FATF/OECD, Jul. 2022. Accessed: Apr. 24, 2026. [Online]. Available: `https://www.fatf-gafi.org/content/dam/fatf-gafi/brochures/Partnering-in-the-Fight-against-Financial-crime-handout.pdf`.

[8] N. Agrawal, R. Binns, M. V. Kleek, K. Laine, and N. Shadbolt, "Exploring design and governance challenges in the development of privacy-preserving computation", in *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama, Japan, May 8-13, 2021*, Y. Kitamura, A. Quigley, T. I. Katherine Isbister and, P. Bjørn, and S. M. Drucker, Eds., ACM, 2021, 68:1–68:13. DOI: `10.1145/3411764.3445677`.

[9] M. Jullum, A. Løland, R. B. Huseby, G. Ånonsen, and J. Lorentzen, "Detecting money laundering transactions with machine learning", *Journal of Money Laundering Control*, vol. 23, no. 1, pp. 173–186, Jan. 2020, ISSN: 1368-5201. DOI: 10.1108/JMLC-07-2019-0055. eprint: https://www.emerald.com/jmlc/article-pdf/23/1/173/1555833/jmlc-07-2019-0055.pdf.

[10] B. Dumitrescu, A. Baltoiu, and S. Budulan, "Anomaly detection in graphs of bank transactions for anti money laundering applications", *IEEE Access*, vol. 10, pp. 47 699–47 714, 2022. DOI: 10.1109/ACCESS.2022.3170467.

[11] J. Jense, "Finding bounded-length cycles in decentralised networks under privacy constraints: Consumer-friendly transaction monitoring", M.S. thesis, Delft University of Technology, Delft, The Netherlands, Jul. 2024. [Online]. Available: https://repository.tudelft.nl/record/uuid:a559439a-d5ee-4f1f-9823-2bab3905c0c9.

[12] K. Zhong and S. Angel, "Oryx: Private detection of cycles in federated graphs", *Proc. Priv. Enhancing Technol.*, vol. 2025, no. 2, pp. 527–542, 2025. DOI: 10.56553/POPETS-2025-0075.

[13] C. P. Martins, "Private cycle detection in financial transactions", M.S. thesis, Delft University of Technology, Delft, The Netherlands, Jan. 2023. [Online]. Available: https://repository.tudelft.nl/file/File_0c71b989-af88-44af-9a69-d569eae99395?preview=1.

[14] Massachusetts Institute of Technology and Tech Xplore, "Researchers develop innovative method for secure operations on encrypted data without decryption", *Tech Xplore*, Mar. 2025. [Online]. Available: https://techxplore.com/news/2025-03-method-encrypted-decryption.html.

[15] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation", *ACM Comput. Surv.*, vol. 51, no. 4, 79:1–79:35, 2018. DOI: 10.1145/3214303.

[16] S. I. Serengil and A. Özpinar, "Encrypted vector similarity computations using partially homomorphic encryption: Applications and performance analysis", *CoRR*, vol. abs/2503.05850, 2025. DOI: 10.48550/ARXIV.2503.05850. arXiv: 2503.05850.

[17] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes", in *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, J. Stern, Ed., ser. Lecture Notes in Computer Science, vol. 1592, Springer, 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16.

[18] W. Stallings and L. Brown, *Computer Security: Principles and Practice*, 4th. Pearson Education, Incorporated, 2018, ISBN: 9780134794105.

[19] D. Boneh, "The decision diffie-hellman problem", in *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, J. Buhler, Ed., ser. Lecture Notes in Computer Science, vol. 1423, Springer, 1998, pp. 48–63. DOI: 10.1007/BFB0054851.

[20] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996, ISBN: 0-8493-8523-7. DOI: 10.1201/9781439821916. [Online]. Available: http://cacr.uwaterloo.ca/hac/.

[21] C. Zhao et al., "Secure multi-party computation: Theory, practice and applications", *Inf. Sci.*, vol. 476, pp. 357–372, 2019. DOI: 10.1016/J.INS.2018.10.024.

[22] L. Zheng, J. Xiao, S. Ma, Z. Chen, and X. Luo, "Temporal cycle enumeration for detecting financial fraud", *Data Intelligence*, vol. 7, no. 3, pp. 567–590, 2025. DOI: https://doi.org/10.3724/2096-7004.di.2024.0058.

[23] D. B. Johnson, "Finding all the elementary circuits of a directed graph", *SIAM J. Comput.*, vol. 4, no. 1, pp. 77–84, 1975. DOI: 10.1137/0204007.

[24] L. Hajdu and M. Krész, "Temporal network analytics for fraud detection in the banking sector", in *ADBIS, TPDL and EDA 2020 Common Workshops and Doctoral Consortium - International Workshops: DOING, MADEISD, SKG, BBIGAP, SIMPDA, AIMinScience 2020 and Doctoral Consortium, Lyon, France, August 25-27, 2020, Proceedings*, L. Bellatreche et al., Eds., ser. Communications in Computer and Information Science, vol. 1260, Springer, 2020, pp. 145–157. DOI: 10.1007/978-3-030-55814-7_12.

[25] R. E. Tarjan, "Enumeration of the elementary circuits of a directed graph", *SIAM J. Comput.*, vol. 2, no. 3, pp. 211–216, 1973. DOI: 10.1137/0202017.

[26] M. B. van Egmond et al., "Privacy-preserving anti-money laundering using secure multi-party computation", in *Financial Cryptography and Data Security - 28th International Conference, FC 2024, Willemstad, Curaçao, March 4-8, 2024, Revised Selected Papers, Part II*, J. Clark and E. Shi, Eds., ser. Lecture Notes in Computer Science, vol. 14745, Springer, 2024, pp. 331–349. DOI: 10.1007/978-3-031-78679-2_18.

[27] K. Becher and T. Strufe, "Efficient cloud-based secret shuffling via homomorphic encryption", in *IEEE Symposium on Computers and Communications, ISCC 2020, Rennes, France, July 7-10, 2020*, IEEE, 2020, pp. 1–7. DOI: 10.1109/ISCC50000.2020.9219588.

[28] J. Jense, *Bounded-private-cycle-detection*, GitHub repository, 2025. Accessed: Jun. 21, 2026. [Online]. Available: https://github.com/junojense/bounded-private-cycle-detection.

[29] R. Albert and A. Barabási, "Statistical mechanics of complex networks", *CoRR*, vol. cond-mat/0106096, 2001. [Online]. Available: http://arxiv.org/abs/cond-mat/0106096.

[30] F. Lillo and R. Valdés, "Dynamics of financial markets and transaction costs: A graph-based study", *Research in International Business and Finance*, vol. 38, pp. 455–465, 2016, ISSN: 0275-5319. DOI: https://doi.org/10.1016/j.ribaf.2016.07.024.

[31] A. Saxena, Y. Pei, J. Veldsink, W. van Ipenburg, G. Fletcher, and M. Pechenizkiy, "The banking transactions dataset and its comparative analysis with scale-free networks", in *ASONAM '21: International Conference on Advances in Social Networks Analysis and Mining, Virtual Event, The Netherlands, November 8 - 11, 2021*, M. Coscia, A. Cuzzocrea, K. Shu, R. Klamma, S. O'Halloran, and J. G. Rokne, Eds., ACM, 2021, pp. 283–296. DOI: 10.1145/3487351.3488339.

[32] Google, *Private-join-and-compute*, GitHub repository, 2026. Accessed: Jun. 21, 2026. [Online]. Available: https://github.com/google/private-join-and-compute.

[33] E. Barker, "Recommendation for key management: Part 1 – general", National Institute of Standards and Technology, Tech. Rep. NIST SP 800-57 Part 1 Rev. 5, May 2020. DOI: 10.6028/NIST.SP.800-57pt1r5.

[34] K. Soramäki, M. L. Bech, J. Arnold, R. J. Glass, and W. E. Beyeler, "The topology of interbank payment flows", *Physica A: Statistical Mechanics and its Applications*, vol. 379, no. 1, pp. 317–333, 2007, ISSN: 0378-4371. DOI: https://doi.org/10.1016/j.physa.2006.11.093.

[35] European Banking Federation, "Banking in europe: EBF facts & figures 2025", European Banking Federation, Brussels, Belgium, Tech. Rep., Dec. 2025. [Online]. Available: https://www.ebf.eu/wp-content/uploads/2025/12/EBF-Fact-and-Figures-2025-18-December-2025.pdf.

[36] J. Lorenz, M. I. Silva, D. Aparício, J. T. Ascensão, and P. Bizarro, "Machine learning methods to detect money laundering in the bitcoin blockchain in the presence of label scarcity", in *ICAIF '20: The First ACM International Conference on AI in Finance, New York, NY, USA, October 15-16, 2020*, T. Balch, Ed., ACM, 2020, 23:1–23:8. DOI: 10.1145/3383455.3422549.

[37] R. Vande Capelle, *Privacy-preserving-characterisation-of-cycles*, 2026. Accessed: Jun. 21, 2026. [Online]. Available: https://github.com/Rodvdc/privacy-preserving-characterisation-of-cycles.

[38] E. R. Altman, J. Blanusa, L. von Niederhäusern, B. Egressy, A. Anghel, and K. Atasu, "Realistic synthetic financial transactions for anti-money laundering models", in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper%5C_files/paper/2023/hash/5f38404edff6f3f642d6fa5892479c42-Abstract-Datasets%5C_and%5C_Benchmarks.html.

[39] M. van den Beukel, J. M. Rozanec, and A. L. Varbanescu, "Tide: A customisable dataset generator for anti-money laundering research", *CoRR*, vol. abs/2603.01863, 2026. DOI: 10.48550/ARXIV.2603.01863.

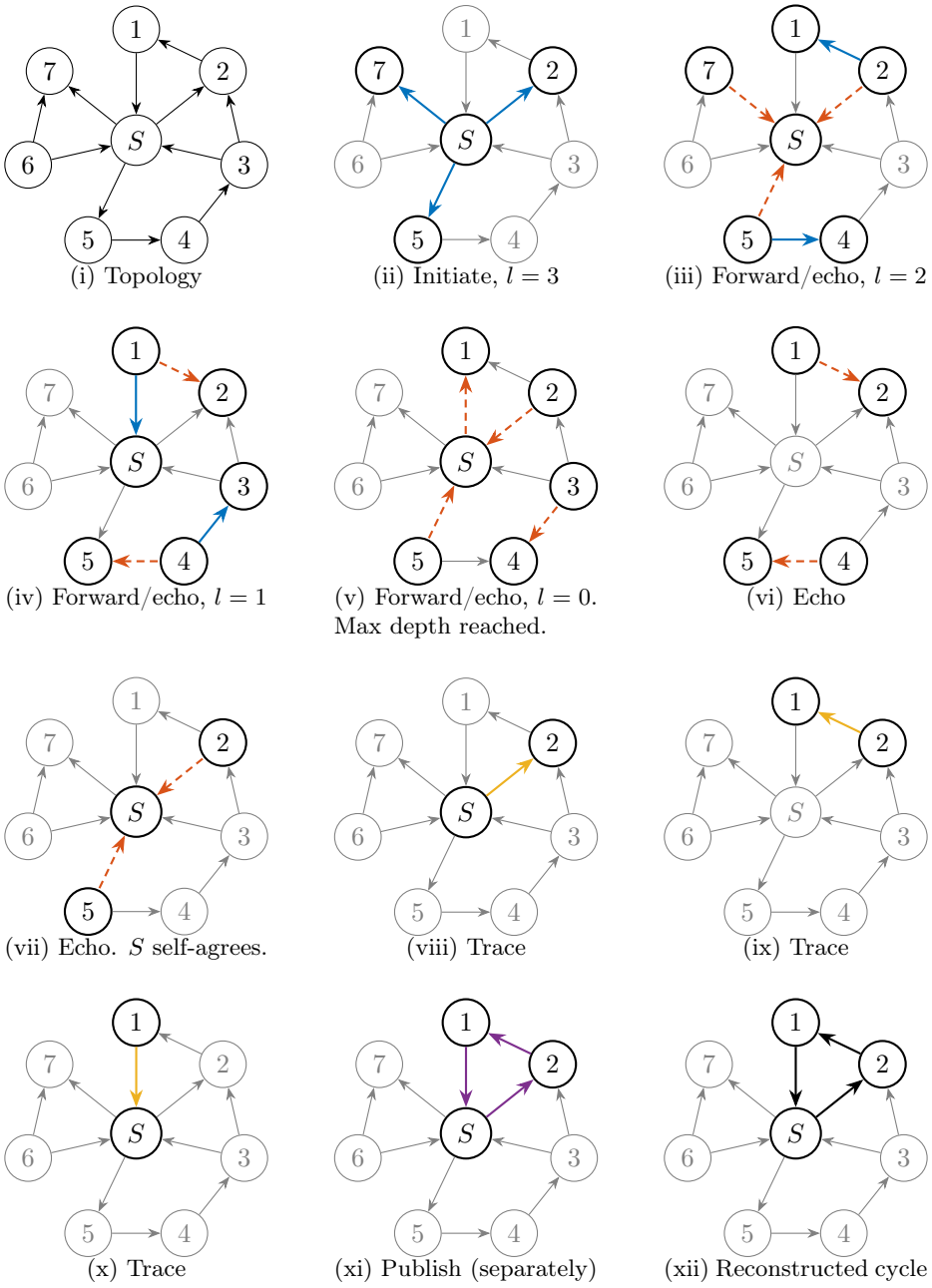
A Symbol Table

This appendix summarises the notation used throughout the paper. Unless a symbol is explicitly defined within a specific section, the definitions provided in Table 1 apply.

Table 1: Symbol Table with Description

Symbol	Description
Graph Theory	
G	Graph $G = (V, E, W)$
V	Vertex Set
E	Edge Set
W	Weight Set
v	Nodes, Vertices
(v_i, v_j)	Edge from v_i to v_j
$N(v)$	Neighbors of node v
$N^+(v)$	Out-neighbours of v
$N^-(v)$	In-neighbours of v
l	Hop distance
n	Total Nodes in G, $\|V\|$
d	Degree
Homomorphic Notation	
PK	Public-key for Homomorphic Encryption
SK	Secret-key for Homomorphic Decryption
$\text{Enc}_{PK}(m)$	Encryption using PK
$\text{Dec}_{SK}(c)$	Decryption using SK
m	Message
c	Ciphertexts
Value	Plaintext final risk value
Threshold	Plaintext Threshold
$\text{risk}(v)$	Risk value calculation function for node v
Privacy-Preserving Cycle Detection (Jense Baseline)	
$\mathbb{Z}_p^*$	Multiplicative Group of Integer Modulo p
$\mathbb{G}_q$	Cyclic Subgroup of $\mathbb{Z}_p^*$ with order q
g	Generator (Cycle Detection)
p, q	Primes Modulus (1025-3072 bits) and Order (160-256 bits) respectively
$x, x_i, x_{i,j}, y$	Secret Key for Jense's algorithm [11]
g^x	Public Key for Jense's algorithm [11]
$r_i, r_{i,j}$	Nonces / Randomness (Cycle Detection)
$\kappa_p, \kappa_q, \kappa_r$	Security Parameters for p, q, and Nonces
κ_h	Security parameter for Paillier encryption

B Sample Execution



(i) Topology (ii) Initiate, $l = 3$ (iii) Forward/echo, $l = 2$

(iv) Forward/echo, $l = 1$ (v) Forward/echo, $l = 0$. Max depth reached. (vi) Echo

(vii) Echo. S self-agrees. (viii) Trace (ix) Trace

(x) Trace (xi) Publish (separately) (xii) Reconstructed cycle

Figure 2: Visualisation of the network behaviour during an instance of the protocol, with initiator S [11]. Note the dotted lines, representing echoes being sent opposite the edge's normal direction. Also note that Figure 2(xi) actually shows three separate steps similar to the trace steps. Note that in our modified algorithm Figure 2(vii) would perform a verification against the received threshold value as shown in Algorithm 2, and based on that result would end the execution or continue to Figure 2(viii)

C Time Complexities

Table 2: Complexity Analysis of the Modified Protocol Running

Function	**Space**	**Time**	**Communication**
`initiate`	$O(n(\kappa_q + \kappa_r))$	$O(n \log^3 \kappa_p)$	$O(n(\kappa_p + \kappa_r))$
`propagate`	$O(n^\ell(\kappa_q + \kappa_r))$	$O(n^\ell(n \log^3 \kappa_p + \kappa_h^3))$	$O(n^\ell(\kappa_p + \kappa_r))$
`echo`	$O(\ell n^\ell(\kappa_r + \kappa_q + \kappa_p))$	$O(\ell n^\ell(\kappa_h^3 + \log^3 \kappa_p))$	$O(\ell n^\ell(\kappa_h + \kappa_r + \kappa_p))$
Overall (simplified)	$O(\ell n^\ell(\kappa_r + \kappa_q + \kappa_p))$	$O(\ell n^\ell(\kappa_h^3 + \log^3 \kappa_p))$	$O(\ell n^\ell(\kappa_h + \kappa_r + \kappa_p))$

D Experimental Results

Table 3: Raw runtime values (in miliseconds) underlying Figure 1.

			Original time (ms)			Characterisation time (ms)		
n	m	$\overline{m} = m/49$	$l = 2$	$l = 3$	$l = 4$	$l = 2$	$l = 3$	$l = 4$
50	147	3	376	382	448	1160	4991	26418
50	196	4	352	382	524	1875	13735	86824
50	245	5	388	435	793	3565	25308	219398
50	294	6	368	459	1139	3592	45004	483857
50	343	7	359	488	1617	4142	60599	695808
50	392	8	358	552	3158	5477	99311	1300098
50	441	9	356	664	5381	7802	149023	2084599