Secure computation of fan-in and fan-out degree of nodes using additive homomorphic encryption

Darius-Eduard Floroiu[1]
Supervisor(s): Dr. Zeki Erkin[1], **Dr. Kubilay Atasu**[1], **Lourens Touwen**[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 20, 2025

Name of the student: Darius-Eduard Floroiu
Final project course: CSE3000 Research Project
Thesis committee: Dr. Zeki Erkin, Dr. Kubilay Atasu, Dr. Megha Khosla

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

There is an increasing need for financial institutions to be able to detect illicit activities such as money laundering. While these institutions currently rely on graph-based analytics or machine learning algorithms for such detection, inter-bank collaboration is hindered by privacy concerns and regulations. In this paper, we introduce a new protocol for computing simple fundamental graph features (specifically fan-in and fan-out degrees) directly on encrypted transaction data using the advantages of additive homomorphic encryption schemes, especially the Paillier cryptosystem. Our algorithm allows a semi-trusted third party to perform computations without accessing plaintext data, enabling privacy-preserving collaboration between banks. Through the paper, we detail the protocol design, analyze its complexity, security and correctness, and demonstrate how it reduces the gap between utility and privacy. While the protocol currently supports only basic graph metrics and assumes a common normalized currency, it offers a scalable and practical foundation for future privacy-preserving financial crime analytics.

1 Introduction

Money laundering represents a critical threat to businesses worldwide, with far-reaching consequences that extend beyond immediate financial risks[1]. With an average cost of scams of $102m to financial institutions in 2022, there is an increasing need to develop a mechanism to combat financial crimes[2]. Although banks are able to analyze transactions within their own network, the current system lacks an efficient and privacy-preserving way to perform inter-bank transaction pattern analysis due to privacy concerns[3].

Trustworthy Financial Crime Analytics focuses on developing advanced algorithms for detecting such financial crimes using graph-based methods[4]. In a centralized setting, where all transaction data is aggregated and processed by a single trusted entity (e.g., a bank or regulator), these algorithms have shown considerable success in uncovering fraudulent patterns[5]. However, centralization raises concerns about data silos, single point of failure, and privacy, especially in scenarios of cross-institutional collaboration.

In recent years, graph-based machine learning has become a popular method for detecting financial crime. These techniques represent transactions as graphs, where accounts are nodes and transfers are edges. Features like *fan-in*, *fan-out*, and *scatter-gather* patterns help detect suspicious behavior such as money laundering[6, 5]. However, there is a major challenge: financial institutions often cannot share transaction data with each other due to privacy and legal restrictions.

To address these concerns and enable collaborative crime detection without exposing sensitive financial data, decentralized privacy-preserving versions leverage techniques such as Secure Multi-Party Computation (MPC) and Homomorphic Encryption (HE). MPC enables multiple parties to jointly compute a function over their private inputs without revealing them to one another[7], whereas HE allows one to evaluate circuits over encrypted data without the need to decrypt it first[8]. By integrating security and privacy, the goal is to enhance trust and compliance in financial crime analytics[4].

HE offers a promising way to address these concerns, by allowing computations to be performed directly on encrypted data without needing decryption, enabling privacy-preserving analysis. Recent work has explored privacy-preserving machine learning models using HE, such as encrypted inference with XGBoost[6] models enriched with graph features extracted via Graph Feature Preprocessors[5]. However, these approaches still require access to plaintext graph structures for feature extraction, and the HE component typically applies only

during the inference phase, not during graph analysis. By using HE, models can be trained on real data as well, instead of being trained on synthetic graphs (such as AMLworld[9]).

In this work we investigate how fan-in and fan-out degrees of nodes in financial transaction graphs can be computed using additive homomorphic encryption. Fan-in and fan-out degrees of nodes are the number of incoming/outgoing edges of a node, which represents an account. These degree measures are fundamental graph features used to detect anomalous patterns such as unusually high incoming or outgoing transaction volumes, which are often indicative of money laundering or fraud. Computing them directly over encrypted data would enable financial institutions to collaborate on identifying suspicious behavior without revealing the underlying transaction data. Compared to Somewhat and Fully Homomorphic Encryption (SWHE/FHE), which support arbitrary computations but remain impractically slow[10], AHE schemes offer a practical trade-off by supporting fast encrypted additions, which suffice for many graph metrics like degree computation.

Our main contributions are as follows:

1. We formalize the problem of encrypted fan-in and fan-out computation under AHE and propose a protocol that enables secure computation of these degree metrics in a decentralized setting

2. We analyze the correctness, efficiency, and privacy guarantees of the protocol, demonstrating its theoretical applicability to collaborative financial crime detection

The remainder of this paper is structured as follows. Section 2 provides background on homomorphic encryption and graph-based financial analytics. Section 3 describes the involved actors, the threat model and presents our proposed protocol in detail. Section 4 evaluates its performance and security. Section 5 reflects on the ethical aspects of our research and discuss the reproducibility of our methods. Finally, Section 6 discusses limitations and future directions, and Section 7 concludes the paper.

2 Background

2.1 Homomorphic Encryption

Homomorphic Encryption (HE)[11] allows for computations to be performed directly on encrypted data without the need of decryption. This powerful technique enables privacy-preserving data processing, where a third party can compute functions over ciphertexts and return the encrypted results to the data owner, who can decrypt it and analyze the result.

At the core of homomorphic encryption lies the principle of computing an arbitrary function $f(m_1, ..., m_n)$ on encrypted data, without learning anything about the plaintext inputs $m_1, ..., m_n$. Let pk denote the public key and sk the secret key. The data owner encrypts each input m_i to obtain ciphertexts $c_i = \text{Encrypt}(pk, m_i)$. A third party, without access to sk, can then evaluate a function over the ciphertexts, denoted as $\text{Eval}(pk, f, c_1, ..., c_n)$, which results in a new ciphertext. When decrypted using the secret key sk, the result reveals the correct value of $f(m_1, ..., m_n)$, i.e.,

$$\text{Decrypt}(sk, \text{Eval}(pk, f, c_1, ..., c_n)) = f(m_1, ..., m_n), \tag{1}$$

which guarantees both the correctness of the computation and the privacy of the input data.

HE schemes are categorized into 3 types, based on the number of operations that can be applied to encrypted values [12]: Partial, Somewhat or Fully HE (PHE/SWHE/FHE). PHE allows only one type of operations, but it can be applied infinitely many times, SWHE allows some types of operations, but only a finite number of times and FHE allows infinitely many operations, infinitely many times. Since any Boolean circuit can be represented using only XOR (addition) and AND (multiplication) gates [12], SWHE and FHE provide a lot of versatility. On the other hand, PHE provides speed advantages since it does not require bootstrapping.

2.2 Paillier Cryptosystem

Paillier cryptosystem (introduced by Pascal Paillier in 1999 [13]) represents an additive homomorphic public-key encryption scheme that relies, for its security, on the Composite Residuosity Class Problem, a number-theoretic assumption. It involves determining whether a given element $z \in \mathbb{Z}_{n^2}^*$ is an n-th residue modulo n^2, where $n = p \cdot q$ is a product of two large primes. Since determining whether z is an n-th residue requires knowledge of the factorization of n, the problem is considered computationally hard for an adversary who only knows the public key.

Let $n = p \cdot q$ be the product of two large primes. The public key is (n, g), where g, usually $g = n + 1$, is a value such that $\gcd(\mathrm{L}(g^\lambda \bmod n^2), n) = 1$, with gcd being the greatest common divisor and L is a function such that:

$$\forall u \in S_n, \quad \mathrm{L}(u) = \frac{u-1}{n}. \tag{2}$$

The private key is $\lambda = lcm(p-1, q-1)$, with lcm being the least common multiple. For a plaintext message $m \in \mathbb{Z}_n$, the encryption is defined as $E_{pk}(m) = g^m \cdot r^n \bmod n^2$, with randomly chosen $r \in \mathbb{Z}_n^*$. The Paillier cryptosystem supports additive homomorphism:

$$D_{sk}(E_{pk}(m_1) \cdot E_{pk}(m_2)) = D_{sk}(E_{pk}(m_1 + m_2)). \tag{3}$$

Other important properties of the Paillier cryptosystem are:

$$\forall m_1, m_2 \in \mathbb{Z}_n \quad \text{and} \quad k \in \mathbb{N},$$

$$\mathsf{D_{sk}}(\mathsf{E_{pk}}(m)^k \bmod n^2) = k \cdot m \bmod n, \tag{4}$$

$$\mathsf{D_{sk}}(\mathsf{E_{pk}}(m_1) \cdot g^{m_2} \bmod n^2) = m_1 + m_2 \bmod n, \tag{5}$$

$$\mathsf{D_{sk}}(\mathsf{E_{pk}}(m_1)^{m_2} \bmod n^2) = m_1 m_2 \bmod n, \tag{6}$$

$$\mathsf{D_{sk}}(\mathsf{E_{pk}}(m_2)^{m_1} \bmod n^2) = m_1 m_2 \bmod n. \tag{7}$$

2.3 Related Work on Encrypted Graph Analytics

Several projects have already applied HE to anti-money laundering (AML). For example, Project *Aurora* shows how HE and Local Differential Privacy (LDP) can be used together to build secure, cross-border AML systems[6]. In the same direction, *CryptoGCN*[14] shows how Graph Neural Networks (GNNs) can be run on encrypted data using FHE.

Still, many of these systems only use HE for the final step —*model inference*— while the graph features themselves are computed in plaintext. The *Graph Feature Preprocessor (GFP)*[5], for instance, extracts features like node degrees and graph patterns before the

data is encrypted. Even systems that combine HE with machine learning models (e.g., TFHE + XGBoost) rely on unencrypted graphs during feature extraction[6].

Some researchers have looked at how graph algorithms could run on encrypted data from the start. For example, Dockendorf et al.[15] adapted algorithms like Bellman-Ford and Kruskal's to work with encrypted graphs using HE. But these methods are often very complex or require special encryption schemes. Many also focus on full graph analytics rather than on simple but useful features like node degrees.

In contrast to these more complex systems, this work focuses on computing simple yet powerful graph features—fan-in and fan-out degrees—using additive homomorphic encryption (Paillier)[13]. This bridges the gap between privacy and utility without revealing sensitive transaction details.

3 Our Protocol

To address the issues described in the introduction, our proposed protocol allows each bank to encrypt its transaction data and send it to a semi-trusted third party that computes simple graph metrics, specifically fan-in and fan-out degrees, to help detect anomalies in account behavior. The third party performs the computation on encrypted data using an additive homomorphic encryption scheme (specifically the Paillier cryptosystem) and only returns the encrypted results to the requesting bank. This preserves the privacy of transaction data while enabling the extraction of useful analytics.

We consider a scenario that contains two categories of parties: Party A (the third party analytics provider) and banks (with a varying number of banks that want to collaborate, and party B - the bank that investigates for illicit activities). We consider the threat model where an adversary can see all the data of the third party, but cannot insert its own data. The protocol also allows the presence of a regulator/auditor, for compliance oversight. Banks are the owners of the data and the protocol initiators. They encrypt transaction data using deterministic homomorphic encryption and send it to the third-party, with their goal being to gain insight into transaction patterns (e.g., fan-in/fan-out) for anti-money laundering. Party A performs encrypted computations over the incoming data and can access and manipulate encrypted data, maintaining a persistent structure (in local memory). There are multiple types of adversaries: curious third parties (may attempt to deduce patterns from ciphertext such as frequency, size, or structure), observers (network attackers monitoring data flows between banks and Party A) and colluding parties (two or more parties sharing encrypted inputs or outputs to reverse-engineer sensitive information). These adversaries want to learn relationships between accounts, detect transaction volume or frequency patterns, or identify account identities. The protocol can also include optional regulators/auditors who may be granted restricted access to metrics or summaries (not raw data) for compliance oversight and could be included in future iterations of the protocol with differential privacy or zero-knowledge proofs.

Now, we describe our algorithm. We start by outlining the assumptions, followed by the four phases of the protocol, and conclude with some remarks on the cryptographic design and data structures.

3.1 Assumptions

For this protocol to properly work, we assume that before sending the money to the third party, each bank converts the sum to the same currency, ideally using the same global

scale to minimize the possible differences. Account IDs are only known by the banks, and an observer cannot link account IDs to real users. All banks trust the cryptographic scheme and the integrity (but not confidentiality) of the third party. It is assumed that Party A does not have access to decryption keys and is honest-but-curious (i.e., follows protocol but may attempt to infer data from ciphertext access patterns or frequency). Adversaries have no access to decryption keys or plaintext data, but may observe ciphertexts, protocol outputs, and possibly the adjacency structure if not hidden. Finally, Party A verifies the identity and trust level of all implied banks (can be done using a digital certificate). Party A maintains a whitelist of trusted senders (banks) that may submit encrypted data.

3.2 Algorithm

Our algorithm starts with the **Setup Phase**, where Party B, the bank who checks for illicit activities, generates and distributes it's public key. After that, in the **Data Submission Phase**, each bank locally encrypts its transaction data using Party B's public key. The encryption method varies per field based on the use case, as shown in Table 1. Next, banks send their encrypted transaction batches to Party A over a secure channel (e.g., TLS). The **Graph Construction** phase starts with party A verifying the authenticity of each transaction. If the bank associated with the transaction is authorized, then the graph constructed by Party A is updated accordingly, as shown in Pseudocode 1. In the final phase, namely the **Query Phase**, Party B requests fan-in and fan-out degree patterns from Party A, as described in Pseudocode 2.

Table 1: Overview of encrypted transaction parameters

Parameter name	**Parameter description**	**Encryption method**
Timestamp	Year/Month/Day Hour/Minute	NA
From Bank	Numeric code for bank where transaction originates	deterministic version of Paillier cryptosystem
Account	Hexadecimal code for account where transaction originates	deterministic version of Paillier cryptosystem
To Bank	Numeric code for bank where transaction ends	deterministic version of Paillier cryptosystem
Account	Hexadecimal code for account where transaction ends	deterministic version of Paillier cryptosystem
Amount Received	Monetary amount received in From account (in currency units of the next row)	non-deterministic version of Paillier cryptosystem
Receiving Currency	Currency such as dollars, euros, etc of From account	NA
Amount Paid	Monetary amount paid (in currency units of next row)	non-deterministic version of Paillier cryptosystem
Payment Currency	Currency such as dollars, euros, etc of From account	NA
Payment Format	How transaction was conducted, e.g. cheque, ACH, credit cards, etc.	NA

Pseudocode 1
Graph Construction

Input: Encrypted transactions from all banks
Output: Encrypted adjacency map (implicit)

for all transactions (t) **do**
    if bank associated with t is not authenticated **then**
        Reject transaction t
    else
        // s, r and amount from t are encrypted
        Let $s \leftarrow$ sender account in t
        Let $r \leftarrow$ receiver account in t
        Let $A[s][r] \leftarrow A[s][r] + t.\text{amount}$
    end if
end for

Pseudocode 2
Query phase

Input: Encrypted ID
Output: Encrypted fan-in and fan-out degrees for the requested ID

Let $i \leftarrow$ encrypted id
Let $fan_{out\_number} \leftarrow len(A[i])$
Let $fan_{out\_value} \leftarrow 0$
for all indexes j **do**
    $fan_{out\_value} = fan_{out\_value} + A[i][j]$
end for
Let $fan_{in\_number} \leftarrow 0$
Let $fan_{in\_value} \leftarrow 0$
for all indexes j **do**
    $fan_{in\_value} = fan_{in\_value} + A[j][i]$
    $fan_{in\_number} = fan_{in\_number} + 1$
end for
return $fan_{out\_number}, fan_{out\_value}, fan_{in\_number}, fan_{in\_value}$

3.3 Cryptographic Design

For this protocol, we decided to use three different ways to encrypt the data, as we explain next. For data that is not relevant for the algorithm, we decided to use a hashing algorithm, since it has speed advantages. For account numbers and bank IDs, we opted for a deterministic version of Paillier, because it allows for deterministic comparison, which is needed for the graph construction phase. Amounts are encrypted using a non-deterministic version of Paillier, as they need to be secret and should be easy to be summed in an encrypted way. Another possible solution would be to use the Okamoto-Uchiyama cryptosystem[16], but, because of the broader message space and being more widely used and studied, we decided to go with the Paillier cryptosystem.

3.4 Data Structures

The initial idea was to store the data as an adjacency matrix, where at row i and column j, it would keep the amount send from account i to account j. Due to the big size of the matrix and the large number of empty entries (most accounts do not send/receive money to/from many different accounts), we decided to change our approach and opt for a hash map of hash maps, where the first map is used to keep track of accounts that have transactions between them and the second one keeps track of the amount of money transferred between these two accounts.

The use of a hash map of hash maps makes the computation of fan-out patterns really easy: for each key in the first map, just count the number of entries in the second map. For the fan-in patterns, there are two approaches that we can follow, depending on what we aim to optimize: speed or memory. For speed efficiency, we can keep in memory 2 hash maps of hash maps (with the second one being the reverse of the first one), so basically the approach for fan-in patterns is the same as the one for fan-out patterns. For memory efficiency, the algorithm would need to go over all entries in the first hash map and check whether the desired account is in the second hash map.

4 Analyses

4.1 Complexity Analysis

In this section, we analyze our proposed protocol in terms of storage, computation and communication complexity. Through the analysis, we denote: n - the number of unique accounts, m - the number of transactions, k - the average number of transactions per account, and l - the bit length of the encryption key.

4.1.1 Space complexity

Each unique sender-receiver pair is processed as a single entry in a nested hash map $A[s][r]$, which represents the total amount transferred from s to r. Multiple transactions between the same pair update the existing hash map entry using homomorphic addition, rather than creating a new one. Let e be the number of unique edges. This results in $O(l)$ bits needed for each entry. Thus, the total storage complexity is $O(e \cdot l)$. In the worst case scenario (when each transaction is between a distinct pair of sender and receiver), the complexity becomes $O(m \cdot l)$.

4.1.2 Time complexity

This section explains the considerations taken when calculating the time complexity. For a quick overview, see Table 2. There are two parts that need to be considered for the time complexity: graph construction phase and query phase. For the graph construction, for each transaction, a single addition is performed to update $A[s][r]$, which takes constant time $O(1)$ to access the index and $O(l^2)$ for Paillier addition, resulting in $O(m \cdot l^2)$ time complexity. However, this complexity can be improved, if the multiplication is optimized. For example, Karatsuba's Algorithm[17] can be used to bring down the complexity to $O(m \cdot l^{\log_2 3})$. Assuming very large keys ($l \gg 4096$ bits), Harvey-Hoeven's Algorithm[18] can be used, which reduces the multiplication complexity to $O(l \cdot \log l)$, resulting in $O(m \cdot l \cdot \log l)$. Other multiplication algorithms can also be used, such as Toom-Cook[19, 20] or Schönhage-Strassen[21], but they do not provide any advantage compared to Karatsuba or Harvey-Hoeven. For the query phase, for the fan-out, we need either $O(1)$ or $O(k)$ for the degree count (depending on the implementation, but most programming languages provide $O(1)$ access time for length of a hash map) and $O(k \cdot l^2)$ for the value (since we need to sum k encrypted values). These lead to a total of $O(k \cdot l^2)$ for fan-out patterns. For fan-in degree, it depends on the chosen implementation. If a reverse map is used, the complexity is the same as the one for fan-out degree, so $O(k \cdot l^2)$. Without a reverse map, we need to iterate through all outer keys, and check all their neighbors, which gives $O(n \cdot k)$. The amount has the same complexity, $O(k \cdot l^2)$, which leads to a total time complexity of $O(n \cdot k + k \cdot l^2)$.

Table 2: Overview of time complexity

Phase	**Multiplication Algorithm (ma) or Adjacency Structure (as)**	**Complexity**
Graph construction	Default Multiplication (ma)	$O(m \cdot l^2)$
Graph construction	Karatsuba (ma)	$O(m \cdot l^{\log_2 3})$
Graph construction	Toom-Cook (ma) with y parts	$O(m \cdot l^{\log_y (2 \cdot y - 1)}$
Graph construction	Schönhage-Strassen (ma)	$O(m \cdot l \cdot \log l \cdot \log(\log l))$
Graph construction	Harvey-Hoeven (ma)	$O(m \cdot l \cdot \log l)$
Query - fan-out	NA	$O(k \cdot l^2)$
Query - fan-in	with reverse map (as)	$O(k \cdot l^2)$
Query - fan-in	no reverse map (as)	$O(n \cdot k + k \cdot l^2)$

4.2 Security and Privacy analysis

Confidentiality: transaction amounts are encrypted using a probabilistic version of Paillier's cryptosystem, which prevents the adversary from learning the values or performing equality tests on ciphertexts. Account IDs and bank IDs are encrypted using a deterministic version of Paillier to allow grouping and comparisons. Although this leaks frequency and equality patterns, the actual identity of the account remains protected since the mapping between encrypted IDs and real identities is known only by the originating bank.

Access Pattern Leakage: our protocol currently does not hide the graph structure. This means that Party A, or any adversary, can infer how many encrypted accounts interact with each other, which allows statistical or frequency-based inferences, but only if the volume of data is small or skewed. The fan-in and fan-out patterns returned to Party B are encrypted, so even Party A cannot learn the results of a query without colluding with Party B.

Injection Attacks: since Party A validates each transaction's sender before accepting it and maintains a whitelist of trusted banks, unauthorized injection is mitigated. A drawback is that Party A cannot determine if banks are honest with their transactions, but this is out of the scope of the algorithm.

Colluding Parties: all transactions are encrypted using Party B's public key. This raises a new potential risk: Parties A and B can collude to decipher the transactions of all other banks. Although this sounds dangerous, in reality, Parties A and B can only learn the decrypted version of the node IDs and transaction amounts, but the correlation between these and real customers is still only known by the bank that owns the transactions.

4.3 Correctness

Our protocol correctly computes fan-in and fan-out degrees as well as the total transferred amounts between accounts, using encrypted transaction data. For each sender-receiver pair, all amounts are homomorphically summed using the additive properties of the Paillier cryptosystem. All arithmetic is performed on fixed-point integers, so there is no loss of precision due to rounding. Since account identifiers are encrypted using a deterministic version of Paillier, the graph structure is preserved, and repeated interactions between the same users are grouped under the same encrypted keys. The correctness of the protocol also relies on the honest data submission by the banks. When this condition is met, our protocol produces reliable results.

5 Responsible Research

Data & Privacy: this research does not involve the gathering of data related to human subjects. Although the account IDs associated with users are collected and stored, only the banks know the link between an ID and a certain user. This research provides a theoretical idea of a new protocol, rather than a concrete implementation, so no data is actually used at all during this research.

Research Integrity: all the sources and data sets used in this investigation are credited properly. Although AI was not used to generate ideas for the algorithm, it was used to check the correctness of some parts, both in terms of content and syntax. The prompts used had the format '*Consider the following paragraph: "[...]". Does it contain any grammatical mistake?*'. AI was also used to help with the formatting of this paper, in order to prettify the aspect of tables, without alternating the data inside.

Replicability/Reproducibility: we strongly think that our contribution can be recreated by other researchers, especially since the paper contains the description and analysis of a protocol, but no proper implementation or results. All the sources and software used are properly referenced and can be accessed by other researchers.

Bias: since this protocol has some limitation, as detailed in Section 6, it can introduce implicit technical bias, depending on the data used. The future implementations should be able to mitigate this risk.

Beyond the project: while we think that there are no risks emerging from output of the protocol, we strongly recommend that the output is not used as a sole decision-making

mechanism for freezing account, but rather as a decision-support tool reviewed by human analysts.

6 Discussion and Future Work

Our proposed protocol provides a theoretically practical and privacy-preserving method for computing fan-in and fan-out degree patterns on encrypted transactional data. By leveraging the advantages of additive homomorphic encryption, Party A, a semi-trusted third party, can perform graph-based computations without access to plaintext data. Compared to centralized data collection methods, our approach offers better privacy guarantees without sacrificing too much performance. While not as versatile as fully homomorphic encryption or secure multi-party computation, the Paillier-based algorithm provides a reasonable tradeoff between efficiency and security.

6.1 Limitations

Our protocol has several limitations. First, our current version of the protocol ignores the timestamps of transactions, which prevents temporal filtering and the analysis of activity over a specific period of time. This limits its ability to detect bursts of suspicious activities and can incorrectly flag constantly active accounts.

Second, the protocol assumes that all amounts are converted to a common currency before encryption. This simplification adds more complexity for the banks and may, in rare cases, lead to erroneous results generated by the different conversion rates used by the banks.

Furthermore, the protocol currently supports only basic pattern detection and does not enable more advanced graph analytics such as scatter-gather, gather-scatter, simple paths and so on. These simple patterns, although crucial for detecting money laundering, fail to detect more complex forms of fraud, which usually happen in a real-world scenario.

Potential colluding parties (Party A and some banks) are also a limitation of our protocol. Although this does not represent a big concern in terms of security and privacy, some methods that mitigate this would be a good addition, in order to gain the complete trust of other banks (and thus willingness to participate in collective fraud detection).

Finally, another limitation of our protocol is access pattern leakage. Despite the fact that transactions and IDs are encrypted, our protocol does not hide the structure of the graph. As a result, this can lead to the reveal of frequency-based information over time.

6.2 Future work

A key direction for future work is enabling support for time-window filtering. This would allow the third party to compute patterns within specific time frames, which could be used to detect bursts of suspicious activities. The protocol can also be improved to allow the request of partial queries (for example, only the degree of an account and not the total sum), which would decrease the complexity to constant time $O(1)$ (if implemented correctly). Another direction would be to handle different currencies within Party A. Since multiplication with a scalar can be done with Paillier, the third party can also convert the amounts before adding them to the graph, to remove any potential differences generated by different conversion rates, but the banks need to make clear what currency is used for each transaction (which is possible if the transactions follow the format described in the IBM AML dataset from Kaggle[22]). Support for more graph analytics would also be a great addition, as they can be

used to detect more complex forms of money laundering. A final direction for future work would be to conduct a comparison between this protocol and other existent approaches, such as Centralized Data Collection, other PHE schemes, SWHE/FHE, MPC or Differential Privacy.

7 Conclusions

This paper introduces a new privacy-preserving protocol that enables the secure computation of fan-in and fan-out degrees on encrypted financial transaction graphs using additive homomorphic encryption. Our approach addresses a critical need for collaborative financial crime detection between financial institutions without compromising the confidentiality of transaction data. By combining deterministic and non-deterministic (probabilistic) encryption, we preserve structural graph properties while hiding sensitive information from the computing party, supporting real-time analytics with minimal leakage. Although there are some current limitations, such as the lack of temporal filtering and support for complex graph patterns, our work creates a theoretically practical groundwork for future research in encrypted graph analytics. Extending the protocol to support more complex patterns, handle multiple currencies and to integrate time-window queries are promising directions to enhance its utility and impact in real-world financial scenarios.

References

[1] Deicy Pareja. Consequences of money laundering for businesses. Pirani Risk Blog, January 2025. Accessed: 2025-06-12. Archived at: `https://archive.ph/jGbdr`.

[2] Haobo Zhang, Junyuan Hong, Fan Dong, Steve Drew, Liangjie Xue, and Jiayu Zhou. A privacy-preserving hybrid federated learning framework for financial crime detection. *arXiv preprint arXiv:2302.03654*, 2023.

[3] Marie Beth van Egmond, Vincent Dunning, Stefan van den Berg, Thomas Rooijakkers, Alex Sangers, Ton Poppe, and Jan Veldsink. Privacy-preserving anti-money laundering using secure multi-party computation. In Jeremy Clark and Elaine Shi, editors, *Financial Cryptography and Data Security - 28th International Conference, FC 2024, Willemstad, Curaçao, March 4-8, 2024, Revised Selected Papers, Part II*, volume 14745 of *Lecture Notes in Computer Science*, pages 331–349. Springer, 2024.

[4] TU Delft. Fintech - expertise. `https://www.tudelft.nl/fintech/expertise#c1549679`, 2024. Accessed: 2025-06-08. Archived at: `https://archive.ph/GPmmT`.

[5] Jovan Blanuša, Maximo Cravero Baraja, Andreea Anghel, Luc von Niederhäusern, Erik R. Altman, Haris Pozidis, and Kubilay Atasu. 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, Brooklyn, NY, USA, November 14-17, 2024*, pages 222–230. ACM, 2024.

[6] Fabrianne Effendi and Anupam Chattopadhyay. Privacy-preserving graph-based machine learning with fully homomorphic encryption for collaborative anti-money laundering. In Johann Knechtel, Urbi Chatterjee, and Domenic Forte, editors, *Security, Privacy, and Applied Cryptography Engineering - 14th International Conference, SPACE 2024, Kottayam, India, December 14-17, 2024, Proceedings*, volume 15351 of *Lecture Notes in Computer Science*, pages 80–105. Springer, 2024.

[7] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

[8] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.

[9] Erik Altman, Jovan Blanuša, Luc von Niederhäusern, Beni Egressy, Andreea Anghel, and Kubilay Atasu. Realistic synthetic financial transactions for anti-money laundering models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 29851–29874. Curran Associates, Inc., 2023.

[10] Konstantin G Kogos, Kseniia S Filippova, and Anna V Epishkina. Fully homomorphic encryption schemes: The state of the art. In *2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 463–466. IEEE, 2017.

[11] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[12] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Comput. Surv.*, 51(4):79:1–79:35, 2018.

[13] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[14] Ran Ran, Wei Wang, Quan Gang, Jieming Yin, Nuo Xu, and Wujie Wen. Cryptogcn: Fast and scalable homomorphically encrypted graph convolutional network inference. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[15] Mark Dockendorf, Ram Dantu, and John Long. Graph algorithms over homomorphic encryption for data cooperatives. In Sabrina De Capitani di Vimercati and Pierangela Samarati, editors, *Proceedings of the 19th International Conference on Security and Cryptography, SECRYPT 2022, Lisbon, Portugal, July 11-13, 2022*, pages 205–214. SCITEPRESS, 2022.

[16] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, volume 1403 of *Lecture Notes in Computer Science*, pages 308–318. Springer, 1998.

[17] Anatolii Alekseevich Karatsuba and Yu P Ofman. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk*, volume 145(2), pages 293–294. Russian Academy of Sciences, 1962.

[18] David Harvey and Joris Van Der Hoeven. Integer multiplication in time o(nlog\,n). *Annals of Mathematics*, 193(2):563–617, 2021.

[19] Andrei L Toom. The complexity of a scheme of functional elements simulating the multiplication of integers. In *Doklady Akademii Nauk*, volume 150(3), pages 496–498. Russian Academy of Sciences, 1963.

[20] Stephen A Cook and Stål O Aanderaa. On the minimum computation time of functions. *Transactions of the American Mathematical Society*, 142:291–314, 1969.

[21] Arnold Schönhage and Volker Strassen. Fast multiplication of large numbers. *Computing*, 7:281–292, 1971.

[22] Erik Altman. IBM Transactions for Anti-Money Laundering (AML). `https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml?select=HI-Large_Trans.csv`, 2019. Accessed: 2025-06-13. Archived at: `https://archive.ph/6BePe`.