Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Künstliche Intelligenz

**Masterarbeit**

von

Philipp **Stangl**

# Entwurf und Implementierung einer Pipeline zur inkrementellen Konstruktion eines Wissensgraphen für die Ermittlung von Betrug mit Kryptowerten

Design and Implementation of an Incremental Knowledge Graph Construction Pipeline for Investigating Crypto Asset Fraud

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Künstliche Intelligenz

**Masterarbeit**

von

Philipp **Stangl**

# Entwurf und Implementierung einer Pipeline zur inkrementellen Konstruktion eines Wissensgraphen für die Ermittlung von Betrug mit Kryptowerten

Design and Implementation of an Incremental Knowledge Graph Construction Pipeline for Investigating Crypto Asset Fraud

| | | |
|---|---|---|
| Bearbeitungszeitraum: | von | 4. Oktober 2023 |
| | bis | 3. April 2024 |

| | |
|---|---|
| 1. Prüfer: | Prof. Dr.-Ing. Christoph P. Neumann |
| 2. Prüfer: | Prof. Dr. Fabian Brunner |

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Selbstständigkeitserklärung

|  |  |
|---|---|
| Name und Vorname des Studenten: | **Stangl, Philipp** |
| Studiengang: | **Künstliche Intelligenz** |

Ich bestätige, dass ich die Masterarbeit mit dem Titel:

**Entwurf und Implementierung einer Pipeline zur inkrementellen Konstruktion eines Wissensgraphen für die Ermittlung von Betrug mit Kryptowerten**

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

|  |  |
|---|---|
| Datum: | 3. April 2024 |
| Unterschrift: |  |

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Masterarbeit Zusammenfassung

| | |
|---|---|
| Student (Name, Vorname): | **Stangl, Philipp** |
| Studiengang: | Künstliche Intelligenz |
| Aufgabensteller, Professor: | Prof. Dr.-Ing. Christoph P. Neumann |
| Durchgeführt in (Hochschule): | OTH Amberg-Weiden |
| Ausgabedatum: | 4. Oktober 2023 |
| Abgabedatum: | 3. April 2024 |

Titel:

**Entwurf und Implementierung einer Pipeline zur inkrementellen Konstruktion eines Wissensgraphen für die Ermittlung von Betrug mit Kryptowerten**

Zusammenfassung:

Kryptowerte sind digitale Vermögenswerte, die Blockchain-Technologie nutzen, um Eigentum nachzuweisen und ein öffentliches Verzeichnis aller Transaktionen zu führen. In dem sich entwickelnden Sektor der Kryptowerte bleiben die Betrugserkennung und -vermeidung nach wie vor eine große Herausforderung. Herkömmliche Ansätze, die sich auf die Analyse von Transaktionsgraphen stützen, bieten eine grundlegende Perspektive, sind jedoch auf Blockchain-Transaktionsdaten beschränkt und erfassen nicht die Semantik in Transaktionen. Folglich sind auch Methoden zur Erkennung und Vermeidung von Betrug eingeschränkt, wenn es darum geht, Blockchain-Adressen mit realen Entitäten zu verknüpfen und Muster betrügerischen Verhaltens von legalem Verhalten zu unterscheiden. Als Antwort auf diese Einschränkungen wird in dieser Arbeit der Kosmosis-Ansatz vorgeschlagen, der darauf abzielt, einen Wissensgraphen schrittweise zu konstruieren, sobald neue On- und Off-Chain-Daten zur Verfügung stehen. Während der Konstruktion wird die Semantik aus Transaktionen extrahiert und Blockchain-Adressen mit realen Entitäten in Verbindung gebracht, indem Blockchain- und Social-Media-Daten in einem Wissensgraphen vereint werden. Die Effektivität und praktische Anwendbarkeit des Kosmosis-Ansatzes wird anhand einer Reihe von realen Rug-Pulls demonstriert, die im Jahr 2021 stattgefunden haben. Dadurch wird veranschaulicht, wie Kosmosis bei der Identifizierung und Verhinderung solcher betrügerischen Aktivitäten helfen kann, indem es präventive Methoden ermöglicht, die Erkenntnisse aus dem konstruierten Wissensgraphen nutzen.

Schlüsselwörter: Blockchain, Cyber Fraud, Knowledge Graph Construction

## Abstract

Crypto assets are digital assets that use blockchain technology to prove ownership and maintain a decentralized and public ledger of all transactions. In the rapidly evolving landscape of crypto assets, the detection and prevention of fraud remain significant challenges. Traditional approaches, primarily reliant on analyzing transaction graphs, offer a foundational perspective but are constrained to on-chain data and fall short in capturing the semantics of transactions. Consequently, fraud detection and prevention methods based on transaction graphs are inherently limited in linking blockchain addresses to real-world entities and discerning patterns of fraudulent behavior from licit behavior. To address these limitations, this thesis proposes the Kosmosis approach, which aims to incrementally construct a knowledge graph as new on- and off-chain data (e. g., social media) becomes available. During construction, this method attempts to extract the semantics in transactions and link blockchain addresses to real-world entities by fusing data from the blockchain and social media in a knowledge graph. The effectiveness and practical applicability of the Kosmosis approach are demonstrated using a series of real-world rug pulls that occurred in 2021. This case illustrates how Kosmosis can aid in identifying and preventing such fraudulent activities by enabling preventative methods to leverage insights from the constructed knowledge graph.

# Contents

**1 Introduction** **1**
1.1 Motivation . . . . . . 2
1.2 Problem Statement and Objectives . . . . . . 3
1.3 Thesis Structure . . . . . . 4

**2 Methods** **5**
2.1 Ontology Development . . . . . . 6
2.2 Prototype Design and Implementation . . . . . . 7
2.2.1 Architecture Design . . . . . . 7
2.2.2 Test-Driven Development . . . . . . 9
2.2.3 Data Acquisition . . . . . . 9
2.3 Evaluation Methodology and Dataset . . . . . . 10

**3 Fundamentals** **11**
3.1 Knowledge Graphs . . . . . . 11
3.1.1 Labeled Property Graph . . . . . . 12
3.1.2 Graphs Based on the RDF Data Model . . . . . . 13
3.1.3 Graph Model Selection . . . . . . 14
3.2 Crypto Assets . . . . . . 14
3.2.1 Crypto Asset Types and Use Cases . . . . . . 15
3.2.2 Minting Crypto Assets . . . . . . 15
3.2.3 Fraud Categories . . . . . . 16
3.3 Blockchain Technology . . . . . . 17
3.3.1 Blockchain Data Structure . . . . . . 17
3.3.2 Smart Contracts . . . . . . 18
3.4 Blockchain Accounting . . . . . . 19
3.4.1 Account-based Model . . . . . . 19
3.4.2 Unspent Transaction Output Model . . . . . . 20
3.5 Privacy Techniques Used in Blockchain . . . . . . 21
3.5.1 Accounting Model Privacy . . . . . . 21
3.5.2 Cooperative Obfuscation . . . . . . 21

**4 State of the Art** **24**
4.1 Incremental Knowledge Graph Construction . . . . . 24
4.1.1 Knowledge Extraction . . . . . 25
4.1.2 Knowledge Processing . . . . . 27
4.1.3 Ontology Development . . . . . 28
4.2 Graph-based Blockchain Data Mining . . . . . 29
4.2.1 Blockchain Address Deanonymization . . . . . 30
4.2.2 Transaction Pattern Recognition . . . . . 37
4.2.3 Illicit Activity Detection . . . . . 38
4.3 Summary . . . . . 39

**5 Rug Pull Prevention Use Case** **40**
5.1 Past User Story . . . . . 41
5.2 From Transaction Graph to Knowledge Graph . . . . . 43
5.3 Alternative User Story . . . . . 43
5.4 Technical Implications . . . . . 45
5.4.1 Competency Questions . . . . . 45
5.4.2 Functional Requirements . . . . . 46
5.4.3 Nonfunctional Requirements . . . . . 47

**6 Kosmosis Approach** **48**
6.1 Architectural Overview of Kosmosis . . . . . 48
6.2 Knowledge Graph Ontology . . . . . 49
6.2.1 Blockchain Account Concept . . . . . 50
6.2.2 Blockchain Transaction Concept . . . . . 51
6.2.3 Social Media Account Concept . . . . . 53
6.2.4 Real-World Entity Concept . . . . . 54
6.3 The Knowledge Graph Construction Pipeline . . . . . 54
6.4 Blockchain Data-Processing Workflow . . . . . 56
6.4.1 Address Relation Extraction . . . . . 57
6.4.2 Address Tagging . . . . . 60
6.4.3 Blockchain Entity Resolution . . . . . 61
6.4.4 RDF Mapping . . . . . 66
6.5 Text-Processing Workflow . . . . . 68
6.6 Enrichment Data-Processing Workflows . . . . . 70
6.6.1 Attributions . . . . . 70
6.6.2 External Knowledge Base . . . . . 70
6.7 Summary . . . . . 71

**7 Evaluation** **72**
7.1 Fitness for Use . . . . . 72
7.2 Comparison with Transaction Graphs . . . . . 74
7.3 Limitations . . . . . 75
7.4 Summary . . . . . 76

**8 Future Work** **77**
8.1 Implementational Features . . . . . 77
8.1.1 Ontology Learning . . . . . 77
8.1.2 Quality Assurance . . . . . 78
8.2 Rug Pull Detection and Prevention . . . . . 78
8.2.1 Rug Pull Detection Algorithms . . . . . 79
8.2.2 Rug Pull Prevention Methods . . . . . 79

**9 Conclusion** **80**

**A Supplementary Material** **83**

**Bibliography** **86**

**Glossary** **96**

**List of Figures** **100**

**List of Listings** **102**

**List of Tables** **103**

# Chapter 1

# Introduction

Crypto assets are digital assets that use blockchain, as a type of distributed ledger technology, to prove ownership and maintain a decentralized and public ledger of all transactions. In the ever-evolving landscape of crypto assets, illicit activities have surged in recent years. Chainalysis, a leading blockchain analytics firm, reported that illicit transaction volume rose for the second consecutive year in 2022, reaching an all-time high of $20.6 billion in illicit activity [1]. Excluding sanctions, figure 1.1 shows that scams have had the highest illicit transaction value for five consecutive years. The substantial increase in illicit transactions suggests that current measures to counter fraud may be inadequate, and there is a necessity for better solutions to identify and prevent fraud. This is especially important as scams pose a significant risk to investors and undermine the integrity of the crypto asset sector.

Transaction graphs, the traditional approach for tracing and identifying fraudulent behavior on blockchain networks fall short due to being limited to information present in blockchain data. This limitation complicates the process of associating blockchain addresses with real-world entities to identify fraudulent actors. However, knowledge graphs (KGs) are increasingly recognized as a powerful means to integrate fragmented knowledge from heterogeneous data sources into a graph of data to facilitate semantic querying and reasoning. A KG provides a holistic view for identifying patterns and hidden connections indicative of fraudulent activities in a highly connected dataset [2]. The KG consists of semantically described entities, each with a unique identifier, and relations among those entities using an ontological representation [3], [4]. Their open-world nature allows for continually integrating new data from blockchains, social media, or other knowledge bases. By leveraging these capabilities, KGs can enhance the analysis of crypto asset fraud and aid in predicting future fraudulent activities.

This thesis proposes Kosmosis, an incremental KG construction pipeline, tailored to construct a KG that serves as a knowledge base for downstream tasks used in the investigation, detection, and prevention of crypto asset fraud. This pipeline integrates new entities and relations from diverse data sources, including blockchains and social media, to keep the graph up-to-date without reconstructing the entire KG.

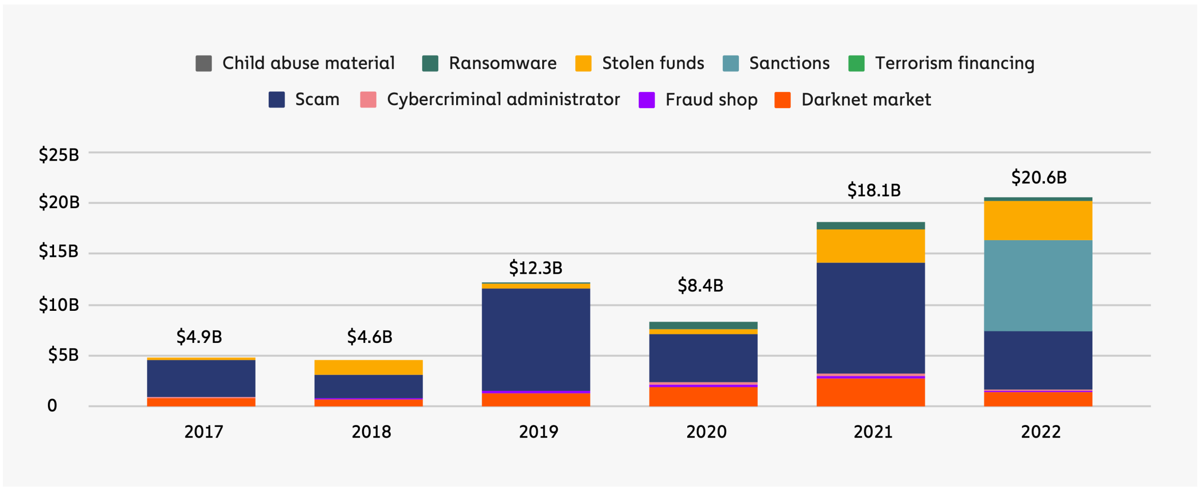


**Figure 1.1:** Total cryptocurrency value received by illicit addresses from 2017 to 2022. Adopted from Grauer *et al.* [1]

## 1.1 Motivation

The predominant approach for identifying patterns indicative of fraudulent activity is the transaction graph analysis within blockchain networks [5], [6], [7, pp. 21–24]. In the context of crypto asset fraud investigations, transaction graph analysis follows a three-step process [8]: (1) blockchain addresses are linked to real-world entities, (2) the annotated graph is run through a graph-analysis framework, and (3) graph analytics methods are used to understand the flow of funds and identify fraudulent behavior. While effective, this approach presents the following three challenges:

1. The transacting parties are pseudonymous and only their blockchain addresses are publicly known. This means that, although the transactions of a specific address can be tracked, linking that address to a real-world entity is challenging, since this approach is limited to information or patterns observable in blockchain data (known as on-chain data).

2. This approach is only concerned with the following aspects of a transaction: (i) the transferred asset, (ii) the quantity, and (iii) the sender and receiver. However, the semantics of a transaction, such as what happened in a transaction that caused the assets to get transferred, is not covered, thereby, limiting the depth of analysis that can be conducted on crypto asset movements and the quality of downstream tasks that can be developed based on this approach.

3. Unlike a KG, which is designed to understand and infer new facts based on the relationships and attributes of the entities it contains, a transaction graph has no ontology. This means it does not possess a structured framework or a set of categories and concepts that could provide context or additional meaning to the data it holds. Therefore, no reasoning or inference of new facts is possible with the transaction graph, as it is purely descriptive and lacks the necessary semantic framework to analyze the underlying causes or implications of the transactions beyond their surface-level characteristics.

## 1.2 Problem Statement and Objectives

To address the shortcomings of transaction graphs, a KG should be constructed that integrates on-chain and off-chain data. This graph can semantically represent the transacting parties on a blockchain and their interconnections. To ensure a high data freshness, the construction should be automated and incremental (i. e., incorporating new data as soon as it becomes available). To design and implement an incremental knowledge graph construction pipeline, the following objectives define the scope and direction of this master’s thesis.

**Objective 1: How to incrementally construct a knowledge graph from on-chain data and off-chain data?**

It is imperative to establish a pipeline capable of integrating updates into the KG in both batch- and streaming-like manners, thereby, maintaining high data freshness by ensuring that the KG consistently reflects the most up-to-date information from the blockchain and other sources. This approach should not entail a reconstruction of the KG, but rather concentrate on integrating new information, avoiding the reprocessing of data that is already incorporated. To commence the construction of the KG, it is necessary to create an ontology that defines the structure and scope of the final KG. It is expected the pipeline will require: (i) data ingestion, where raw data from various sources is collected and prepared for further processing; (ii) knowledge extraction, where relevant information is identified and extracted from the ingested data; and (iii) entity resolution, involving the identification of entities across different data sources.

**Objective 2: How to extract the semantics contained within blockchain transactions?**

Transaction graphs commonly only display transactions with asset transfers and answer questions such as “which” assets were transferred and “where” they were transferred. Extracting the semantics contained in transactions is vital to uncovering sophisticated fraudulent schemes that might otherwise go unnoticed. This gap should be addressed by extracting the semantics of transactions and answering “why” and “how” assets were transferred in a transaction. The extraction of the semantics is expected to be achieved by decoding the input data of a transaction.

**Objective 3: How to automatically link blockchain addresses to their real-world entities during knowledge graph construction?**

Transaction graphs consist solely of blockchain addresses and transactions. To identify the real-world entities involved in these transactions, they require attributions data on the application level. Attributions data is typically sourced from databases that stores details about the real-world identities behind blockchain addresses By merging this attribution data with the transaction graphs, each address in the graph can be associated with information about the real-world entity it represents. It is expected that linking blockchain addresses to real-world entities can be performed during KG construction. As a result, the integrated data is stored in a real-world entity-adjusted representation that allows users to see the actors involved in transactions.

## 1.3 Thesis Structure

The structure of this thesis is organized into three parts, comprising nine chapters in total. The organization of the content is depicted in figure 1.2.

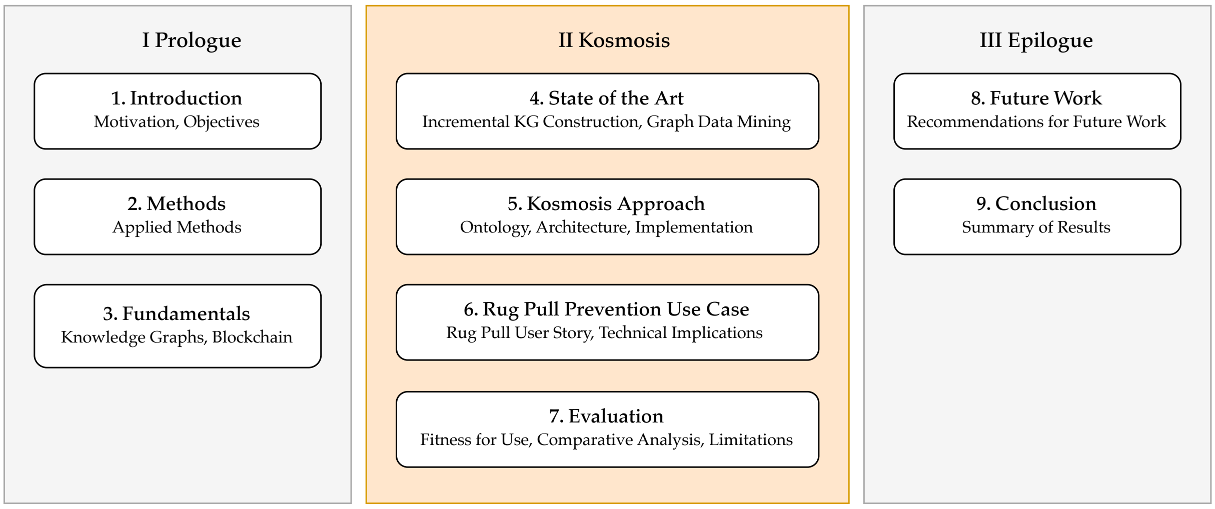

I Prologue
1. Introduction
Motivation, Objectives
2. Methods
Applied Methods
3. Fundamentals
Knowledge Graphs, Blockchain
II Kosmosis
4. State of the Art
Incremental KG Construction, Graph Data Mining
5. Kosmosis Approach
Ontology, Architecture, Implementation
6. Rug Pull Prevention Use Case
Rug Pull User Story, Technical Implications
7. Evaluation
Fitness for Use, Comparative Analysis, Limitations
III Epilogue
8. Future Work
Recommendations for Future Work
9. Conclusion
Summary of Results

**Figure 1.2:** Thesis structure

The prologue to this thesis begins with the introduction in chapter 1, outlining the motivation, objectives, and problem statement. Chapter 2, details the methodologies employed in this work. Including the methods used for the ontology development, prototype design and implementation, and evaluation. Chapter 3, lays the theoretical groundwork, covering KGs, crypto assets, blockchain technology, and blockchain accounting. This ensures a thorough understanding of the background necessary for comprehending the subsequent chapters.

Kosmosis represents the core of the thesis, starting with chapter 4 ("State of the Art") which surveys current advancements in incremental KG construction and graph-based blockchain data mining. Ideas from those two domains are constitutive to the design and implementation of Kosmosis. Chapter 5 uses the rug pull prevention use cases to highlight the technical implications and requirements for the Kosmosis architecture. Subsequently, chapter 6 presents the Kosmosis approach, detailing the KG ontology and the architecture of the KG construction pipeline. The evaluation of the software prototype is presented in chapter 7. The assessment encompasses the fitness for use, a comparative analysis of the KG versus transaction graphs, and a discussion about current system limitations.

The epilogue concludes the thesis, with chapter 8, suggesting directions for further research and development based on the findings of this work. Finally, chapter 9 summarizes the research findings and contributions, providing a comprehensive overview of the study and its significance in the field of crypto asset fraud investigation.

# Chapter 2

# Methods

This master’s thesis follows the three-cycle design science approach by Hevner [9] that comprises three intertwined cycles: relevance, design, and rigor. The chosen methodology guided the iterative development of the software prototype, from gaining an initial understanding of the application domain to the design, implementation, and evaluation of the incremental KG construction pipeline, as follows:

(1) **Relevance Cycle**: Given the rise in illicit crypto asset transactions, there is a pressing need to investigate, detect, and prevent fraud in this domain. To gain an in-depth understanding of the crypto asset fraud domain, a comprehensive literature review was conducted, as a first step. This review aimed to ensure the relevance of the primary artifact of this work, a software prototype, and its applicability to real-world challenges. It covered various aspects, including the historical context of crypto asset fraud, the challenges and current approaches to its detection and prevention, and the technical intricacies of blockchain networks. The primary sources for this review included peer-reviewed papers, whitepapers, and industry reports. Insights derived from this cycle played a crucial role in informing subsequent cycles.

(2) **Rigor Cycle**: The design of the incremental KG construction pipeline was based on the theoretical foundations and the analysis of existing methods. Knowledge gained from the literature was used to ensure the scientific rigor of the design.

(3) **Design Cycle**: In this internal cycle, the actual design was converted into a working implementation. The design principles that guided it are outlined in section 2.2. The pipeline was tested and evaluated using real and synthetic data (section 2.3) to determine its effectiveness and efficiency. The findings from the evaluation were reintroduced into the design to lead to its improvement. This cycle was repeated until a satisfactory result was achieved.

## 2.1 Ontology Development

Competency questions (CQs) guided the development of the ontology by defining the requirements of the ontology and constraining the scope of knowledge represented within it. CQs are formulated in natural language. They are specific, well-defined questions that an ontology should be able to answer if it adequately covers the domain it represents [10]. The CQs were derived from the rug pull prevention user story, described in chapter 5, and formulated using the competency question-driven ontology authoring methodology proposed by Ren *et al.* [11]. The authors outline a framework for categorizing CQs by focusing on their semantic content rather than their linguistic structure, utilizing a feature-based modeling approach. Their framework identifies the following seven criteria for formulating CQs:

1. The **question type** criterion differentiates CQs based on the nature of their expected answers, which can be a set of entities or values for *selection questions*, a Boolean value for *binary questions*, or a numerical count for *counting questions*.
2. **Element visibility** determines whether the elements involved in a CQ, such as class and property expressions, are explicitly stated or implied.
3. **Question polarity** indicates whether the question is framed in a positive or negative manner, influencing how the query is understood and answered.
4. **Predicate arity** identifies the complexity of the main predicate in the question. This can range from unary predicates for a single entity or value and binary predicates concerning relationships between two entities or values to N-ary predicates that involve relationships among multiple entities or values.
5. **Relation type** specifies the kind of relation involved in the CQ, with a focus on object property relations or datatype property relations.
6. **Modifiers** impose restrictions on entities or values within the question to specify concrete values, ranges, or comparative measures.
7. **Domain-independent elements** refer to elements that can occur across different knowledge domains, such as temporal (time) and spatial (location) elements.

*Protégé* [12], an open-source ontology editor for building knowledge-based software, was used to develop and refine the ontology for the KG Kosmosis constructs. It is considered "the leading ontological engineering tool" [13] that offers a graphical user interface to define ontologies. Further, it features deductive classifiers that apply reasoning techniques to deduce the classification of instances or the relationships between concepts that are not explicitly stated but can be logically inferred from the existing axioms. Those classifiers can be used for consistency validation to ensure there are no contradictions or conflicts within the definitions and relationships specified in the ontology. For example, it would check that a class cannot be both a subclass and a superclass of the same concept, as this would be logically inconsistent.

## 2.2 Prototype Design and Implementation

The design and implementation of the software prototype was grounded in object-oriented programming (OOP) and object-oriented design (OOD) principles, with Python serving as the programming language. Python was chosen for its extensive ecosystem of libraries for data engineering and text processing. This included *RDFLib* [14], a library for working with Resource Description Framework (RDF), as well as *Spacy* [15] and the *Natural Language Toolkit* (NLTK) [16] for the processing of natural language text.

### 2.2.1 Architecture Design

OOP allows for decomposing the KG construction pipeline into smaller, manageable components (objects) that can be developed, tested, and maintained independently. Additionally, Unified Modeling Language (UML) diagrams were employed to design and visualize the system architecture and the interactions among these components. The design of the prototype utilized OOD principles to shape the system's architecture. To achieve this objective, the design uses three specific patterns: the singleton and factory method from the category of creational design patterns, and the strategy pattern from the category of behavioral patterns.

#### Singleton

The *singleton* pattern (figure 2.1) is a creational design pattern that ensures a class has only one instance and provides a global point of access to that instance [17]. This pattern involves a class that is responsible for creating just one instance of itself, controlling how it is created, and providing a way to access it globally. The implementation of this pattern typically involves a private constructor to restrict instantiation, a static method to provide access to the instance, and a static member variable to hold the single instance. This pattern is useful when exactly one instance of a class, which should be accessible globally, is needed, for instance, for logging, or database connections.

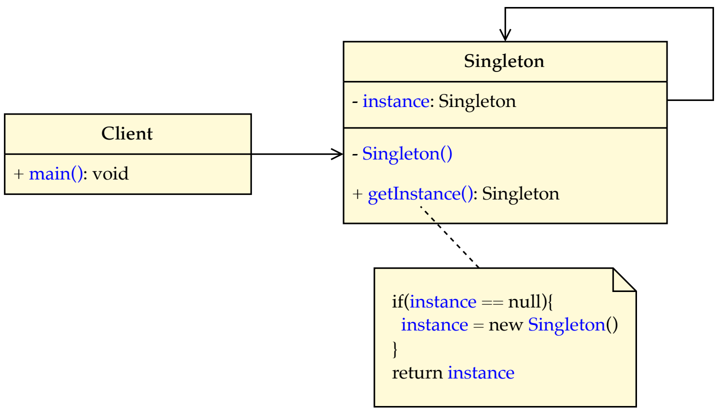


**Figure 2.1:** UML class diagram of the singleton pattern

### Factory Method

The *factory method* (figure 2.2) is a creational design pattern that provides an interface for creating instances of a class, but allows subclasses to alter the type of objects that will be created [17]. This pattern defines an interface for creating objects, but allows subclasses to decide which class to instantiate. Implementation involves a method in a superclass that creates objects, which can be overridden by subclasses to create specific types of objects. It is useful when a class cannot anticipate the class of objects it must create or when a class intends its subclasses to specify the objects it creates.

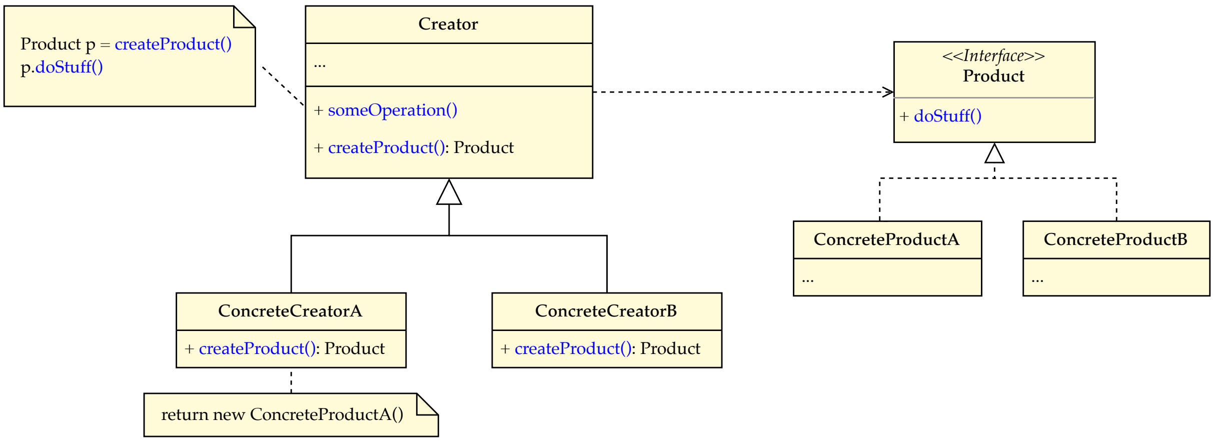


**Figure 2.2:** UML class diagram of the factory method pattern

### Strategy

The *strategy* pattern (figure 2.3) is a behavioral design pattern that defines a family of algorithms, encapsulates each one, and makes them interchangeable [17]. This pattern allows a client class to choose from a family of algorithms at runtime. It involves creating a strategy interface defining the common methods for all supported algorithms, concrete strategy classes implementing these algorithms, and a context class that uses the strategy interface to execute the selected algorithm. This pattern helps to dynamically select an algorithm at runtime or switch between related algorithms.

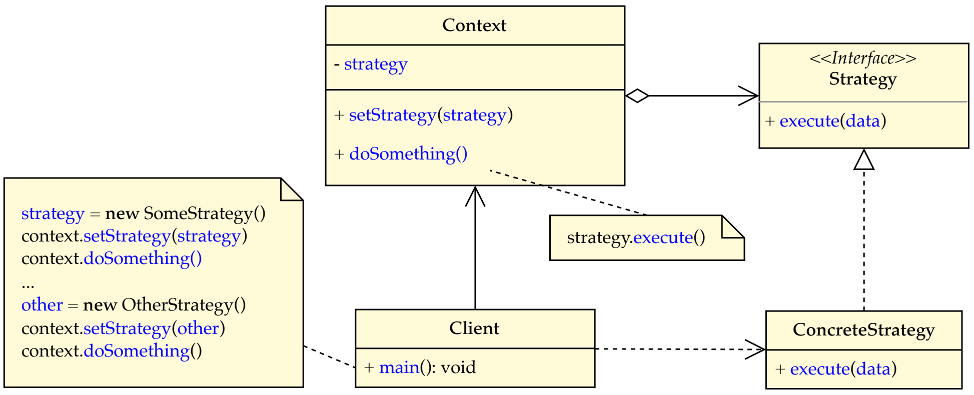


**Figure 2.3:** UML class diagram of the strategy pattern

### 2.2.2 Test-Driven Development

The implementation process adhered to test-driven development (TDD) principles, where test cases were written prior to implementing functionality. This approach ensured that each component of the pipeline was rigorously tested for correctness and reliability. It involved creating unit test cases before the implementation, following an iterative process of writing failing tests, writing the minimum code to pass the test, and then refactoring the code to improve its structure and efficiency without changing its external behavior.

In TDD, developers follow a cycle of adding a test to the suite (red), writing code to pass the test (green), and refactoring the code while keeping tests green. This iterative process helps in creating optimized code, understanding client requirements better, and simplifying the addition and testing of new functionalities during development stages. Using tools like *Pytest* [18] facilitated the TDD approach by providing a robust framework for writing and executing tests. This method encourages cleaner, more maintainable code and fosters a development environment where features can be added or changed with confidence, knowing that the test suite will immediately catch regressions or errors.

### 2.2.3 Data Acquisition

Data was acquired from the following publicly available data sources for the initial and incremental construction of the KG:

- **Blockchain** data was sourced from *Quicknode* [19] archive nodes. An archive node enables queries of historical data by storing all historical blockchain states since the genesis block (i. e., the first block of the blockchain). This includes access to all historical states of smart contracts as well as account balances. For on-chain data from Bitcoin, the *Blockbook RPC add-on* for archive nodes was used to retrieve historical UTXO data.
- **Social media** data was ingested from the 𝕏 *Filtered stream* [20] API endpoint to collect posts related to specific user accounts from the social media platform 𝕏.
- **Attributions** were sourced from community curated datasets from blockchain explorers.[1] Attributions data,[2] associating blockchain addresses to real-world entities, was stored in a *PostgreSQL* [22] database.
- **External knowledge** was integrated from the Golden KG via the *Fact API* [23] to enrich the KG entities with supplementary information such as locations, investors, and funding history, as well as official website and social URLs for organizations. It provides structured data on various entities, including public figures and organizations with a focus on the tech and crypto asset sector.

[1] Blockchain explorers (e. g., *Etherscan* [21]) are web tools designed to facilitate the exploration of blockchain networks. These tools allow users to efficiently search, filter, and sort transactions, blocks, accounts, and other information in the blockchain network.

[2] Example: 0x575 (blockchain address), Treasury (address label), Tether (real-world entity)

## 2.3 Evaluation Methodology and Dataset

To evaluate the completeness of the ontology, competency questions were employed as a methodological tool. These questions were formulated to validate the ability of the ontology to handle specific queries and operations that it is expected to perform. The satisfactory ability to answer these competency questions serves as a validation check, ensuring that the ontology meets the necessary criteria for effectiveness and completeness [10]. This method provides a practical test, verifying the ontology’s capability to function as intended in real-world applications.

A custom dataset was created for the technical evaluation of the Kosmosis prototype, particularly for the use case of rug pull prevention. This dataset comprised historical blockchain data spanning from October 7, 2021, to November 23, 2021. The data was downloaded from Ethereum, Polygon, and Bitcoin archive nodes accessed through Quicknode. Table 2.1 illustrates the scope and scale of the dataset gathered from historical blockchain data between October 7, 2021, and November 23, 2021. It lists the total number of blocks mined on each blockchain alongside the total number of transactions that have been mined within those blocks.

| Blockchain | # Blocks | # Transactions |
|---|---|---|
| Bitcoin | 7152 | 13,218,345 |
| Ethereum | 305,384 | 61,891,925 |
| Polygon | 1,800,221 | 179,122,984 |

**Table 2.1:** Blockchain dataset used for evaluation

The attributions database was populated with data from blockchain explorers that provide community curated datasets. For the Ethereum blockchain, data was sourced from the *Etherscan Label Cloud* [21] with labels for 347,631 addresses across 197 distinct categories. Similarly, for the Polygon blockchain, the database was enriched with information from the *Polygonscan Label Cloud* [24]. This addition comprised labels for 4,997 addresses, distributed over 138 categories. Unfortunately, an equivalent dataset for Bitcoin that included at least one labeled address for this use case was not available.

Finally, social media posts were obtained from the *full-archive search endpoint* [3] of the 𝕏 API, focusing on content posted by the user Homer_eth. A total of 92 posts were collected, all of which originated within a predetermined timeframe. The data extracted from these posts included several key elements: `post.id`, which serves as a unique identifier for each post; `post.created_at`, indicating the time and date each post was created; and author information such as `user.id`, `user.name`, and `user.username`.

[3] `https://developer.twitter.com/en/docs/twitter-api/tweets/search/quick-start/full-archive-search`

# Chapter 3

# Fundamentals

This chapter introduces the foundational concepts and technologies underpinning the research for Kosmosis. Section 3.1 begins by defining KGs and their significance for organizing and interpreting highly connected datasets. Subsequently, section 3.2 provides an overview of crypto assets and the underlying blockchain technology (section 3.3) that enables their existence and transactions. To provide a background for blockchain address de-anonymization methods discussed in subsequent chapters, section 3.4 outlines blockchain accounting principles, and section 3.5 delineates privacy techniques used in blockchain.

## 3.1 Knowledge Graphs

The core idea of a KG is to represent real-world entities and their interrelationships in a structured and machine-understandable way. KGs aim to capture the meaning and context of information by focusing on what they represent and how they relate. Data is organized as nodes (entities) connected by potentially cyclical edges (relationships) forming a graph structure. These entities can be anything from physical objects and people to abstract concepts and events. KGs offer a semantically rich data model by encoding semantics, meaning the relationships between entities carry specific meanings. Employing a graph-based knowledge abstraction allows maintainers to postpone the definition of a schema, allowing the data and its scope to evolve. As a result, their application is particularly useful for capturing incomplete knowledge.

Cross-domain KGs, such as *DBpedia* [25] or the *Google Knowledge Graph* [26], are created to represent general knowledge about the world. They span multiple areas of knowledge, offering a broad overview and linking diverse subjects. For instance, the Google KG integrates data from a variety of sources (e. g., web content), allowing Google to deliver more relevant and informative search results. Domain-specific KGs, conversely, focus on a particular field, such as finance, providing in-depth, expert-level knowledge tailored to specific needs [27]. An application of domain-specific KGs in finance is to detect unusual transaction patterns that may indicate fraudulent activities.

On the methodology of construction, KGs can be categorized into on-the-fly, pre-, and incrementally constructed KGs. On-the-fly constructed KGs are dynamically generated in response to specific queries or tasks, ensuring up-to-date information and flexibility in handling diverse queries. In contrast, a pre-constructed (or static) KG is built in advance and offers the advantage of stability and extensive validation, making them reliable sources for consistent queries over time. Incremental construction of KGs is an emerging methodology that aims to efficiently update the KG when the underlying data sources change, even with minor updates, without starting from scratch.

The term *Knowledge Graph* itself was made popular with the Google Knowledge Graph announcement in 2012 [26]. Today, there is still disagreement over what constitutes a KG [28]–[31]. Across literature [3], [4], a KG is commonly referred to as a knowledge base with semantically described entities, each with a unique identifier, and relations among those entities shown in an ontological representation. To structure the data, a KG uses a graph-based data structure, such as the labeled property graph (LPG) [32], or a graph built from RDF [33] terms.

### 3.1.1 Labeled Property Graph

LPG-based knowledge graphs organize and manage knowledge using graph databases, such as *neo4j* [32]. In the LPG model, both nodes (entities) and edges (relationships) can possess a set of properties as well as labels. Properties, defined as key-value pairs, characterize nodes and relationships, whereas labels serve to categorize the nodes and edges according to their roles or types within the domain being modeled. For instance, in a social network graph, nodes could represent users, with labels such as "User," while edges representing the relationships between users might be labeled as "follows." Additionally, nodes and edges can carry multiple labels, further detailing their roles and relationships within the modeled domain.

Figure 3.1 shows an example of John Doe, who owns a bank account, modeled as an LPG. The node "John Doe" has the label "Customer" and two properties attached: his email *j@doe.com* and a phone number *+49 111 22*. The other node, "A520," has the label "Bank Account" and two properties describing that it is a checking account with a balance of *1000.00*. Both nodes are connected by an "ownership" relationship that describes that John Doe has owned the bank account since January 1, 2022.

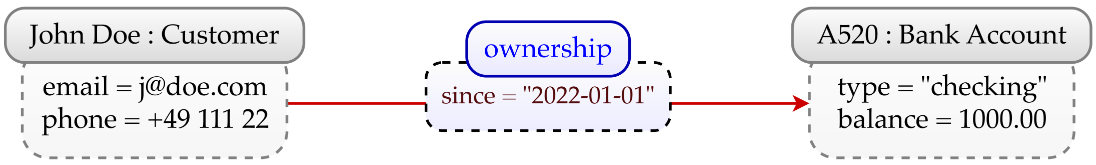


**Figure 3.1:** John Doe bank account example modeled as LPG

An LPG can be formally defined as a tuple $(N, E, L, P, U, e, l, p)$, where $N$ is a set of node ids, $E$ is a set of edge ids, $L$ is a set of labels, $P$ is a set of properties, and $U$ is a set of values. $e : E \to N \times N$ maps an edge id to a pair of node ids; $l : N \cup E \to 2^L$ maps a node or edge id to a set of labels, and $p : N \cup E \to 2^{P \times U}$ maps a node or edge id to a set of property-value pairs [31].

Querying in LPG-based databases is facilitated through specialized query languages such as *Cypher* [34]. These query languages are designed to efficiently navigate and manipulate the structure of the graph, taking full advantage of the database's optimization for graph traversals. The use of these query languages allows for the execution of sophisticated queries that can span multiple nodes and relationships, enabling the extraction of complex patterns and insights from the graph.

Graph databases optimize for read operations by employing index-free adjacency [35]. This means that in a graph database, relationships between nodes are stored directly as pointers, allowing for efficient traversal of connections without the need for indexes, resulting in a $\mathcal{O}(1)$ lookup cost. This design choice enhances the speed of read operations in graph databases, especially when dealing with complex and interconnected data structures. However, [36] shows that this property is only beneficial in scenarios where the database contains a single, large graph.

### 3.1.2 Graphs Based on the RDF Data Model

RDF is used as the data exchange format for the semantic web. In RDF, entities are uniquely identified through Uniform Resource Identifiers (URIs), facilitating the reference to resources within both global and local namespaces.

A KG that represents as a set of RDF triples is known as an RDF graph. Each RDF triple $(s, p, o)$ is an ordered set of the following RDF terms [37]: a subject $s \in U \cup B$, a predicate $p \in U$, and an object $o \in U \cup B \cup L$. An RDF term is either a URI $u \in U$, a blank node $b \in B$, or a literal $l \in L$. Unlike the LPG, standard RDF has no direct way to express edge properties. Instead, RDF can aggregate information into named graphs, transforming traditional triples into quads $(s, p, o, n)$ where $n$ is a named graph. Alternatively, RDF makes edge properties possible through extensions like RDF-star, which allow for statements about statements in a graph. Entities and relations of an RDF graph are semantically described using an ontology [3]. Gruber [38] defines an ontology as "a formal, explicit specification of a shared conceptualization." It defines the types of entities that exist in a domain and the relationships between them. They use a vocabulary, that is a collection of terms or labels that are used to describe a specific domain or subject area. The schema definition capabilities of RDF are further expanded by RDF Schema (RDFS) and the Web Ontology Language (OWL), which enable expressive semantic structures. They enable the definition of classes, properties, and hierarchies, and enrich RDF with semantics through axioms, thereby describing the structure of RDF instances and adding inferential capabilities.

For data querying and manipulation, RDF utilizes *SPARQL* [39], a standardized query language, with SPARQL-star introduced for querying within the RDF-star framework. RDF supports a variety of standard exchange formats, including N-Triples, N-Quads, Turtle, RDF/XML, and JSON-LD, facilitating diverse data interchange scenarios.

RDF does not provide mechanisms for ensuring data integrity or enforcing shape constraints. To validate the semantic correctness, solutions like the Shape Constraint Language (SHACL) [40] or Shape Expressions (ShEx) [41] are required.

SHACL prioritizes SPARQL for validation, with an extension mechanism, and allows expressing constraints on the content, structure, and meaning of a graph, including conditions on property values. ShEx, on the other hand, was designed de novo to meet specific use cases and defines schemas in terms of a grammar [42].

Figure 3.2 shows the same example described in the previous section but this time arranged as RDF terms and visualized in the form of a graph. "John Doe" and "A520" are instances of "Customer" and "Bank Account," respectively. One difference compared with the LPG is that the ownership relationship is represented as a named graph, which allows the relationship itself to have the property "since."

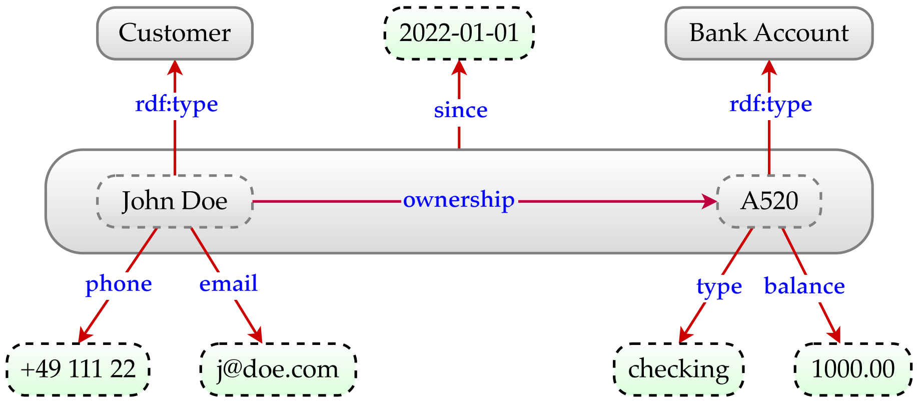


**Figure 3.2:** John Doe bank account example modeled as RDF graph

### 3.1.3 Graph Model Selection

The decision between the use of RDF and LPG depends on the use case of the final KG. Despite the LPG being optimized for queries, especially those requiring deep traversal of the graph (e. g., tracing the flow of assets through multiple transactions or addresses), this work is motivated by an emphasis on semantic relationships between entities in the final KG. This makes them well-suited for encoding, inferring, and querying relationships and metadata related to crypto assets and transactions. Such functionality is particularly useful in identifying patterns and relationships indicative of fraud, such as identifying suspicious transactions that deviate from typical patterns. Therefore, for the remainder of this thesis, the term KG will refer to a graph that is based on the RDF data model.

## 3.2 Crypto Assets

Crypto assets are digital assets that use blockchain technology to prove ownership and maintain a decentralized and public ledger of all transactions. This section starts by outlining the various types of crypto assets along with their use cases, as detailed in section 3.2.1. Following this, there is a brief description of how new assets are created, a process known as minting, in section 3.2.2. Concluding, the different categories of crypto asset fraud types are examined in section 3.2.3.

### 3.2.1 Crypto Asset Types and Use Cases

There are three distinct types of crypto assets (depicted in figure 3.3), each with unique characteristics and use-cases. The most well-known type is cryptocurrencies, like *Bitcoin* [43], *Ethereum* [44], and *Solana* [45]. They function as digital currencies and are used for storing or transferring monetary value. Fungible tokens (short: tokens), another type of crypto asset, are interchangeable units representing various utilities or assets within a blockchain ecosystem. They often play a vital role in Decentralized Finance (DeFi) protocols that provide users with access to a product or service on a blockchain-based platform. They can be designed to provide utility (e. g., *Chainlink* [46]), or to minimize volatility by pegging their market value to an external reference, known as stablecoins, like the US dollar in the case of the *USD Coin* [47]. Lastly, Non-Fungible Tokens (NFTs) are unique digital assets that prove ownership and authenticity of digital, such as *CryptoPunks* [48], or real-world assets [49]. Unlike cryptocurrencies and tokens, each NFT has a distinct value and is non-interchangeable.

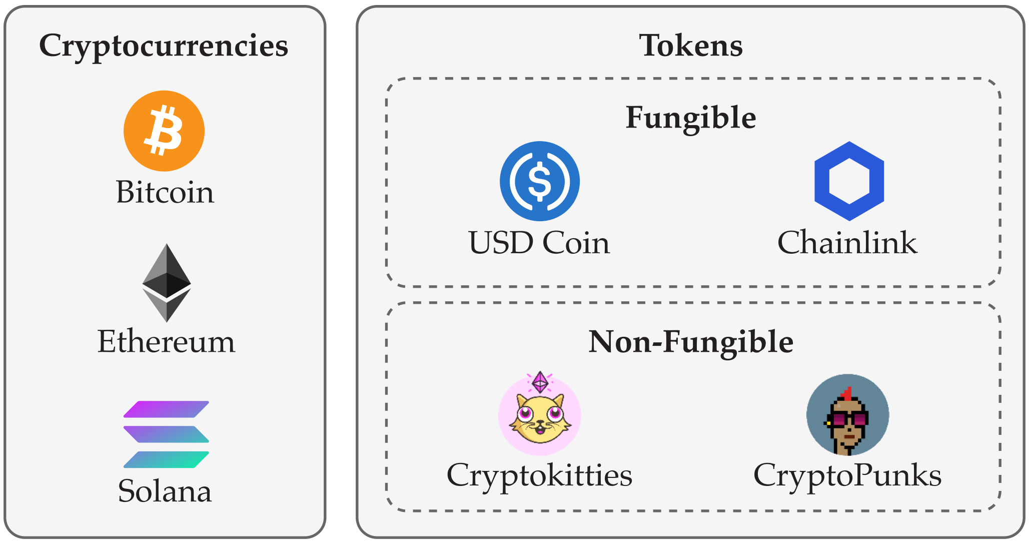


**Figure 3.3:** Overview of different types of crypto assets

Crypto asset trading is primarily conducted through three types of marketplaces: (1) centralized exchanges (e. g., *Coinbase* [50]) serve as intermediaries between buyers and sellers, enabling the trading of cryptocurrencies and fungible tokens. In contrast, (2) decentralized exchanges (e. g., *Uniswap* [51]) operate without a central authority, allowing users to trade directly with one another through automated smart contracts on blockchain networks. Additionally, (3) NFT marketplaces (e. g., *OpenSea* [52]) provide online platforms for the buying, selling, and trading of NFTs.

### 3.2.2 Minting Crypto Assets

Minting is the process of creating new crypto assets on a blockchain network without the interference of central authorities. Tokens are typically minted by the creator either at the inception of the project or progressively over time. This process is often governed by predefined rules or algorithms embedded within the smart contracts of the project.

In contrast, NFT minting involves other individuals besides the token creator, referred to as token minters. They engage by invoking a specific function within a smart contract, in the ERC-721 token standard, called mint. This action results in an increase in the supply of the NFTs and simultaneously assigns these minted tokens to the blockchain address of the minter. The mechanism of minting NFTs often involves utilizing a dedicated minting website. Here, prospective minters or investors are required to invest a predetermined amount, as set by the creator, to initiate the minting process. This investment grants them the ability to mint one or multiple NFTs, depending on the terms set forth in the smart contract. This process not only facilitates the creation of new NFTs but also serves as a means of transferring ownership directly from the creator to the NFT minter.

### 3.2.3 Fraud Categories

Crypto asset fraud encompasses a range of illicit activities that undermine the integrity and security of the crypto asset market. Fraudulent behavior, in crypto and traditional financial markets, is considered to be any suspicious activity that is unauthorized, distinguishing it from scams, which are deceptive transactions that victims may inadvertently authorize. The major categories of crypto asset fraud include scams, protocol hacks, money laundering, ransomware payments, and darknet markets. Each category is outlined in table 3.1 along with a brief description.

| Name | Description |
|---|---|
| Darknet Markets | Darknet markets are websites that facilitate the sale of illicit goods and services. Cryptocurrency, valued for its perceived anonymity and efficiency in international transactions, is used as the means of payment on the markets [1, pp. 70–84], [53]. |
| Money Laundering | The process of disguising the origins of ill-gotten funds that come from criminal activities, making it seem as if the funds originate from lawful sources [54, pp. 10–11]. |
| Ransomware | This form of cyberattack targets individuals and organizations, encrypting critical files, databases, and applications, rendering them unusable until a ransom is paid to obtain the decryption key. Victims are coerced into paying cybercriminals to regain access to their data or systems, usually with cryptocurrencies like Bitcoin to make tracking difficult [1, pp. 26–40]. |
| Scams | Scammers promote fake investment opportunities and give-aways or impersonate influential individuals or companies to defraud their victims. In another type of scam, called *rug pull*, creators of a blockchain project suddenly abandon it and take investors' funds with them. This type of scam is extensively discussed in recent literature [55]–[58]. |

**Table 3.1:** Overview of crypto asset fraud categories

## 3.3 Blockchain Technology

Blockchain technology is based on the principles of immutability, decentralization, transparency, and cryptographic security and has seen various applications in recent years. For instance, in the financial sector (e. g., [43], [44]), or supply chain management (e. g., using a single blockchain [59], or using multiple, interoperable blockchains [60], [61]). The following section offers an introduction to blockchain technology, beginning with the blockchain data structure in section 3.3.1. Proceeding with the concept of smart contracts, section 3.3.2 outlines their mechanisms and applications.

### 3.3.1 Blockchain Data Structure

A blockchain is a data structure whose elements, called blocks, are linked together to form a chronologically-ordered chain of blocks [62]. Figure 3.4 illustrates that each block comprises two parts: a body and a header. The body of the block contains a set of transactions. A transaction typically involves the transfer of assets between a sender and a receiver. These participants are represented by addresses, serving as unique alphanumeric identifiers that enable users to send and receive digital assets on a blockchain network. A blockchain address is typically derived from a public key (pubkey), which itself is generated from a private key through cryptographic algorithms. For instance, Bitcoin uses ECDSA for key generation and the SHA-256 and RIPEMD160 hashing algorithms to generate the Bitcoin address from the pubkey [63]. Further, the block body is used to generate a unique identifier called the block hash. The block header contains a reference to its immediate predecessor, the parent block.

Blockchain nodes, including archive and full nodes, are utilized to access the blockchain. Archive nodes store all historical states of a blockchain from the genesis block to the present. This means that archive nodes keep a record of all balances and contracts at every block, not just the current state. These types of nodes are primarily used for data analysis and verification purposes. Full nodes, on the other hand, store and maintain the latest block data (specifically, the most recent 128 blocks) on disk [64]. Their purpose is to validate blocks and transactions against the consensus rules of the blockchain to ensure the integrity of all blocks and states.

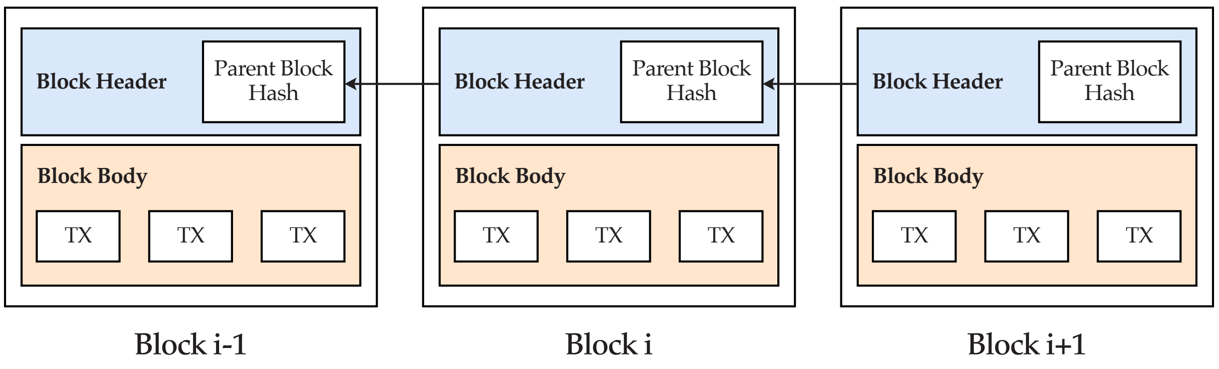


**Figure 3.4:** Blockchain datastructure. Adapted from Zheng *et al.* [62]

### 3.3.2 Smart Contracts

Smart contract platforms are a subset of blockchains that enable the development of decentralized applications through smart contracts. Through smart contacts, which are executable source codes that enforce the terms and conditions of particular agreements, a smart contract platform like Ethereum facilitates the development of decentralized applications [65]. Once deployed on the blockchain, the smart contract is assigned an address where the code resides and cannot be altered or tampered with. By writing custom smart contracts, developers can create and manage tokens that adhere to the ERC-20 [66] or ERC-721 (NFT) [67] standard [65]. An Application Binary Interface (ABI) specifies the functions and data structures exposed by a smart contract, allowing external applications to understand the capabilities of the contract. Further, an ABI defines a format for encoding and decoding data that is passed between smart contracts and external applications. This ensures a standardized way to exchange information.

The Ethereum blockchain operates with the Ethereum Virtual Machine (EVM) as a fundamental building block, serving as the execution environment for smart contract code. Smart contracts, primarily written in a high-level language such as Solidity, undergo compilation into EVM bytecode. This bytecode is the executable format used by the EVM to enact smart contract functions. To interact with this bytecode, a contract ABI is utilized, which acts as a bridge between the high-level language and the low-level bytecode. In this context, an EVM disassembler plays a crucial role; it reverses the bytecode back into a more readable format, aiding developers in understanding and analyzing the code deployed on the Ethereum blockchain. Figure 3.5 shows the processes involved in deploying smart contracts to the Ethereum blockchain and reading contract data from it. The left side shows the compilation and deployment process of a smart contract, and the right side depicts an interaction with the contract. An example for a contract creation transaction is provided in listing A.1 and A.2.

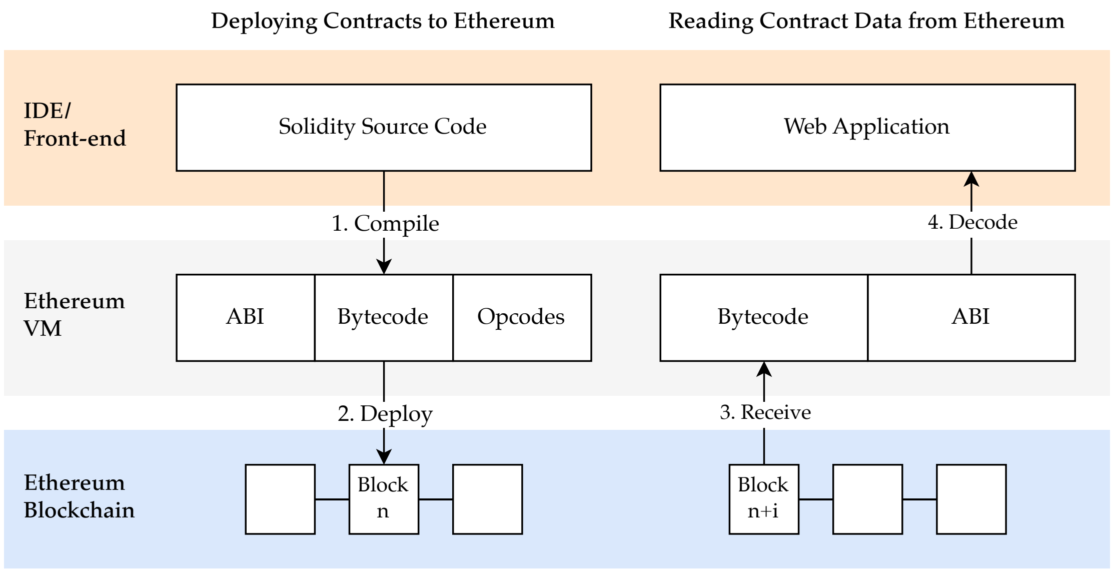


**Figure 3.5:** Schematic representation of deploying and reading from smart contracts. Adapted from Takeuchi [68]

## 3.4 Blockchain Accounting

Blockchain accounting models are fundamental to how transactions are recorded and managed across different blockchain systems. Each model offers a distinct approach to tracking and verifying ownership and transfers of assets. This section outlines the two relevant accounting models for this work: the account-based model (section 3.4.1) and the unspent transaction output (UTXO) model (section 3.4.2).

### 3.4.1 Account-based Model

The account-based model is used by Ethereum [44], Polygon, and other blockchain platforms designed to support smart contracts and decentralized applications. Like a bank account that tracks the inflow and outflow of funds, thereby reflecting the current balance, the account-based model in Ethereum maintains a global state of accounts and balances of Ether (denoted in ETH). Each transaction results in a direct adjustment to this balance, akin to a deposit or withdrawal in a bank account. This stateful nature of the account-based model ensures that at any given moment, the system can accurately reflect the total amount of Ether held in each account, offering an up-to-date view of account balances within Ethereum. Account-based blockchains distinguish between externally owned accounts (EOAs), which are controlled by private keys and used by individuals, and smart contract accounts, which are governed by their contract code.

Consider the example under the account-based model, depicted in figure 3.6: Initially, Alice is credited a balance of 30 ETH in her account, while Bob holds 15 ETH in his account. When Alice decides to transfer 15 ETH to Bob, she initiates a transaction that triggers changes in their respective account balances. As the transaction is processed, Alice's account shows a deduction of 15 ETH, effectively reducing her holdings to 15 ETH. Simultaneously, Bob's account balance increases by 15 ETH, reflecting the receipt of the transferred funds. This action results in Bob's new balance increasing to 30 ETH.

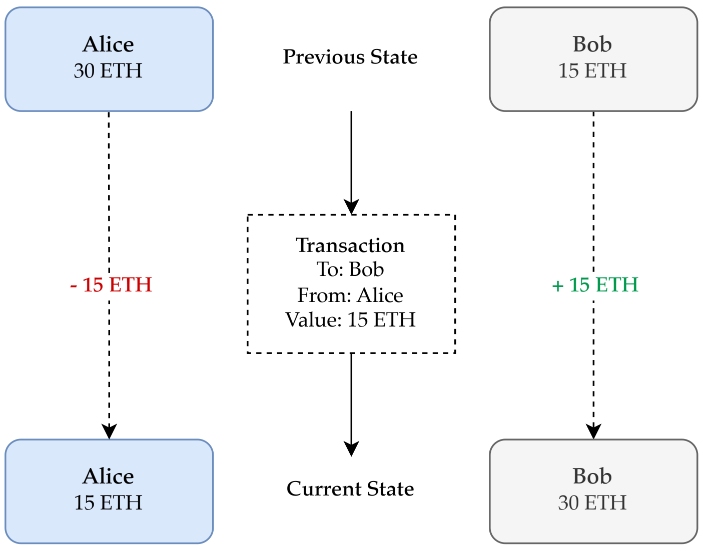


**Figure 3.6:** Transaction between Alice and Bob using ETH within the account-based model

### 3.4.2 Unspent Transaction Output Model

The Bitcoin blockchain employs the UTXO model, which is distinctive for tracking ownership through outputs. In this model, transactions are composed of inputs and outputs, where outputs from one transaction become inputs for another (an example is provided in the appendix in listing A.3). These outputs represent the unspent funds that a user has available to spend, and the sum of a user's UTXOs constitutes their total balance. When initiating a Bitcoin transaction to send funds, the sender specifies the recipient's address and the amount in Satoshis, alongside a ScriptPubKey. This ScriptPubKey is a cryptographic puzzle that locks the specified amount to the recipient's address, requiring a correct public key and signature for spending.

The UTXO model facilitates transaction processing by leveraging scripting languages. It employs two scripts: ScriptPubKey, which locks the transaction output to a recipient, and ScriptSig, which the spender uses to unlock this output for spending in a new transaction. This model ensures that each transaction output remains unspent until used by the recipient, allowing Bitcoin wallets to calculate their total balance based on accessible UTXOs. Unlike the account-based model, the UTXO model generates change when the transferred amount does not exactly match the payment amount.

The UTXO accounting model can be best understood through the analogy of banknotes in a wallet. In this model, the balance of a Bitcoin address is akin to the total sum of banknotes (UTXOs) it contains. These UTXOs are unspent, meaning they are locked by the address and are available to be used in future transactions. Much like banknotes, UTXOs are indivisible; they cannot be split into smaller units. Consider the example under the UTXO model, depicted in figure 3.7: In order to pay Bob 15 Bitcoin (BTC), Alice must send Bob a total amount of 20 BTC by bundling two UTXOs (each with 10 BTC) to make up the total input value of 20 BTC for this transaction. As a result of the transaction, two new UTXOs are created. Output 1 is the value sent to Bob. Output 2 is the "change" created through the transaction and returned to Alice.

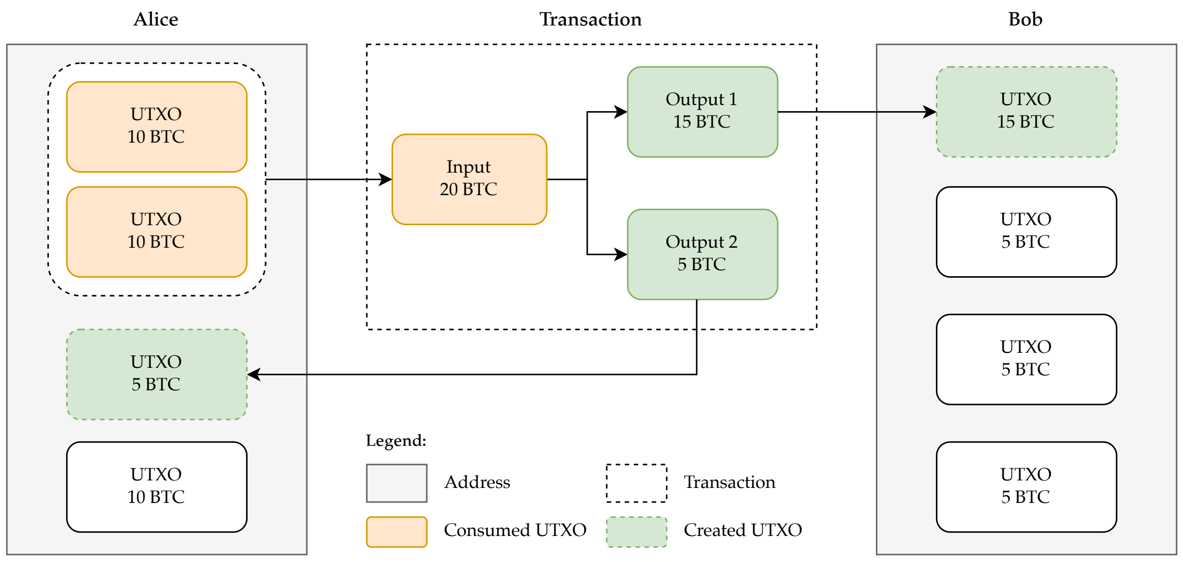


**Figure 3.7:** Transaction between Alice and Bob using BTC within the UTXO model

## 3.5 Privacy Techniques Used in Blockchain

Blockchain networks provide pseudo-anonymity as there is no user identification required to join the network and only the addresses are publicly known. The following section only covers the relevant privacy techniques for the scope of this thesis. For a more comprehensive overview of privacy and security features, I refer to [69].

### 3.5.1 Accounting Model Privacy

The privacy afforded by different accounting models varies significantly, presenting a spectrum of implications for users engaged in blockchain transactions. The UTXO model is a distinct paradigm that provides enhanced privacy features compared with its counterpart, the account-based model. This distinction stems from the structure and operational mechanics of the UTXO model. Unlike the account-based model, where all funds are consolidated into an account balance, the UTXO model allows an account holder to possess multiple instances of UTXOs without combining them into one total amount. Moreover, the UTXO model supports the usage of disparate addresses for individual transactions. This feature significantly obfuscates the linkage between different transactions and, by extension, the accounts involved. Consequently, tracing transactions back to a single owner becomes considerably challenging.

An account holder, for instance, referred to as Alice, can execute numerous transactions simultaneously, each utilizing distinct UTXO instances. When engaging in a transaction with a payee, say Bob, Alice is required only to disclose the specific UTXO instances allocated for the payment. This selective disclosure shields the total balance or the aggregate sum of Alice’s UTXO holdings from the payee. To illustrate, Alice can seamlessly distribute one BTC to Bob and two BTC to Carol, from a single three BTC UTXO instance. Throughout this process, neither Bob nor Carol gains visibility into the UTXO portfolio of Alice, preserving the privacy of her total portfolio value.

### 3.5.2 Cooperative Obfuscation

In the realm of digital currencies, particularly within the Bitcoin ecosystem, the UTXO model has been lauded for its inherent privacy features. However, the research work conducted by Meiklejohn *et al.* [70] shows that even with the UTXO model, some Bitcoin payments can be traced. To counter this, obfuscation techniques have become increasingly sophisticated. Among the various strategies devised to reinforce the anonymity of Bitcoin transactions, the category of “cooperative obfuscation” methods has gained substantial traction in recent years [71]. This category encompasses mixing services and CoinJoin, which are predicated on the collective action of users to obscure the traceability of transactions. Mixing services are not exclusive to the Bitcoin network and have been adapted across various blockchain networks to enhance privacy by blending the origins of transactions. Conversely, CoinJoin remains a unique approach within the Bitcoin network, utilizing a specific method of transaction batching to merge multiple inputs and outputs, thereby complicating the task of tracing transactions.

**Mixing Services**

Mixing services, often referred to as tumblers, offer a compelling solution to the challenge of transactional privacy. These services operate by pooling and mixing the coins of several participants, effectively disassociating the original funds from their final destinations. Users participate by sending their Bitcoins to a mixing service, which then blends their coins with those of other users. Subsequently, the mixed coins are redistributed to the participants, but because they are intermingled with the assets of others, tracing the path from sender to recipient becomes more challenging.

Nowadays, mixing services have become widely used in account-based blockchains due to the latter's inferior privacy. Tornado Cash is one of the most popular mixing service on Ethereum, with almost 1.6 million Ether deposited into its mixing contracts as of 2021 [72]. This protocol utilizes a two-step process (depicted in figure 3.8): The initial phase requires a user to send an amount of Ether to the Tornado Cash smart contract from an originating address. This action generates a deposit note, which is required for the subsequent withdrawal phase. The essence of this mechanism lies in its ability to dissociate the deposited amount from its withdrawal. After a certain latency period, which serves as a further measure to disassociate transactional links, the user is able to initiate the withdrawal phase. Utilizing the previously obtained deposit note, the user creates a withdrawal transaction for the same amount of Ether. This prompts the contract to transfer the deposited Ether to a specified refund address, chosen by the user. The refund address can be any Ethereum address, allowing for the possibility of directing funds to newly created addresses with no transaction history.

The security and anonymity of Tornado Cash transactions are underpinned by zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) [73]. This cryptographic technology ensures that the deposit and withdrawal transactions are entirely independent of each other. Moreover, it guarantees that each deposit correlates with a singular withdrawal, thereby eliminating the possibility of linking transactions to their participants.

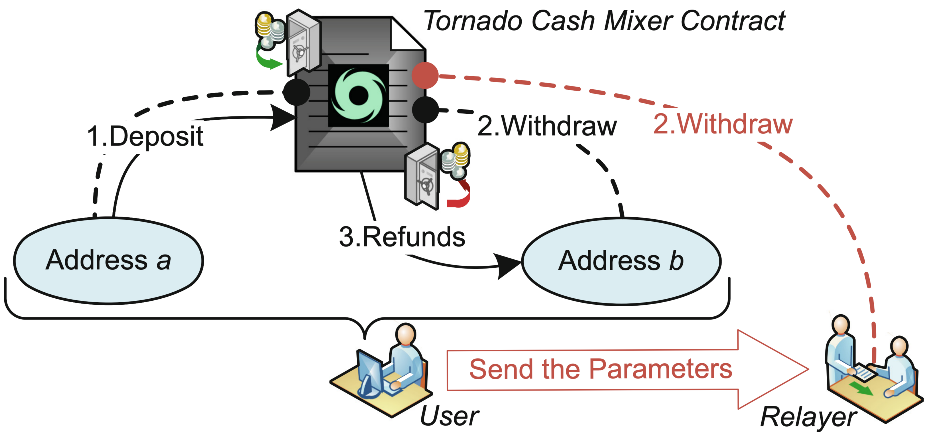


**Figure 3.8:** Process of Tornado Cash coin mixing. Adopted from Tang *et al.* [72]

## CoinJoin

CoinJoin represents another facet of cooperative obfuscation, distinguished by its peer-to-peer approach that eliminates the need for a centralized service. In a CoinJoin transaction, multiple participants agree to combine their transactions into a joint transaction. By doing so, they obscure the link between the input (sending) and output (receiving) addresses involved in the transactions. This approach was proposed by Maxwell [74] in 2013 to address privacy concerns with the *common-input ownership* heuristic (cf. section 4.2.1). At present, different implementations of CoinJoin exist:

- *Chaumian CoinJoin* [74], [75]: CoinJoin based on Chaumian blind signatures
- *WabiSabi* [76]: a generalization of Chaumian CoinJoin for centrally coordinated CoinJoins with variable amounts
- *JoinMarket* [77]: an implementation based on a maker/taker structure where the maker provides their Bitcoin (for a fee) as part of the taker's transaction
- *ZeroLink* [78]: mixes multiple transactions into one batch, further hindering source and destination tracing
- *Whirlpool* [79]: software implemented in the Samourai wallets that allows users to engage in CoinJoin rounds through a Whirlpool entrusted coordinator using a Chaumian CoinJoin implementation

Figure 3.9 illustrates the concept of a CoinJoin transaction compared to a regular Bitcoin transaction. The left side depicts two regular, individual transactions ($T_0$, $T_1$) where each sender transacts with each recipient separately. Transaction $T_0$ transfers funds from Alice to Carol, with some change being returned to Alice. Similarly, transaction $T_1$ shows the transfer of funds from Bob to Dave, with some change being returned to Bob. The right side of the image, depicts the CoinJoin transaction $T_{CoinJoin}$ with inputs from both Alice and Bob. Instead of separate outputs for each recipient, there are multiple outputs from the CoinJoin transaction $T_{CoinJoin}$ that go to Carol, Dave, and the change addresses.

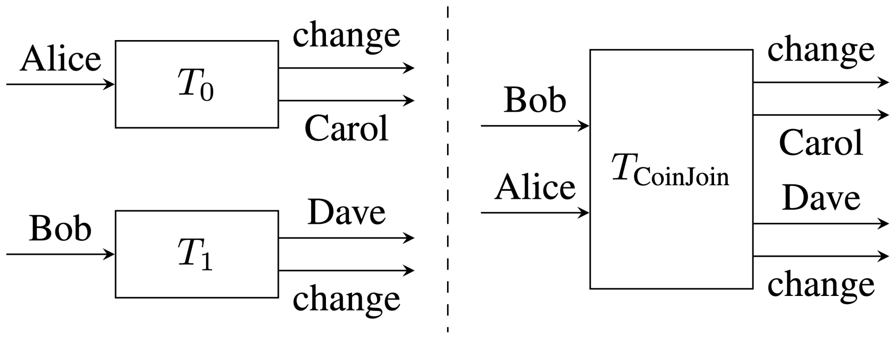


**Figure 3.9:** Two individual transactions on the left are combined into a CoinJoin transaction on the right. Adopted from Möser and Böhme [80]

Now that the fundamentals of KGs and blockchain technology have been explored, the following chapter analyzes the state of the art.

# Chapter 4

# State of the Art

This chapter discusses related work explored in the research for Kosmosis. Firstly, the methodologies and advancements in incremental KG construction are discussed in section 4.1. Subsequently, section 4.2 examines graph-based blockchain data mining that can be utilized in the domain of crypto asset fraud.

## 4.1 Incremental Knowledge Graph Construction

The work of Tamašauskaitė and Groth [81] analyzes existing approaches for KG construction. They show that current approaches for the construction of KGs are based on a batch-based approach. This approach requires the recreation of the entire KG and thus a significant amount of repetitive processing to repeatedly extract and modify the same (unchanged) data. Data integration and the elimination of inconsistencies must be performed repeatedly. For this reason, recent studies [4], [81] have proposed a generic KG development life cycle consisting of the following five tasks:

1. **Data Acquisition**: Identification of relevant data sources. Acquisition and initial transformation of source data. Initial data cleaning, if required.
2. **Knowledge Extraction** (section 4.1.1): Derivation of structured information and knowledge from un- or semi-structured data. Extracting entities and relations.
3. **Knowledge Processing** (section 4.1.2): Identification of matching entities in the data, mapping data to the ontology, and integrating new knowledge into the KG.
4. **Ontology Management** (section 4.1.3): Creation and incremental development of an ontology that is applied to the KG.
5. **Quality Assurance**: Maintaining a high data quality in the KG by evaluating and updating it accordingly. Additionally, Hofer *et al.* [4] consider knowledge completion as part of quality assurance. This includes the completion and enrichment of the knowledge in the KG (e. g., predicting new relations, learning missing type information).

### 4.1.1 Knowledge Extraction

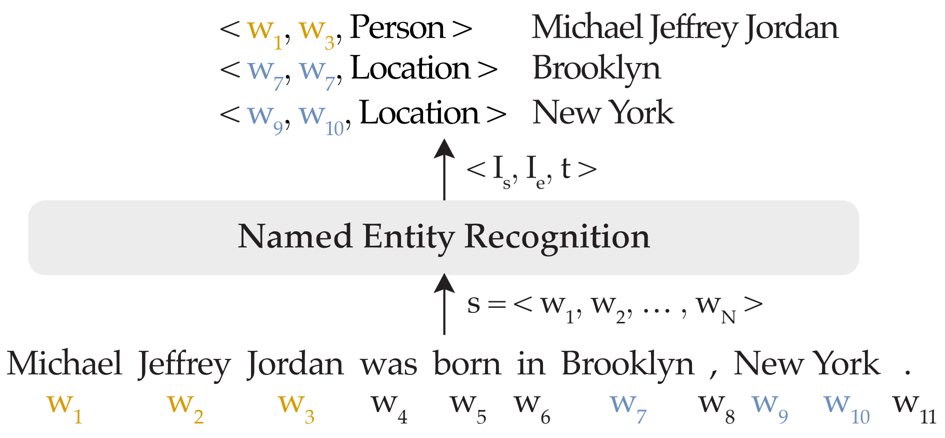
Data ingested into the KG construction pipeline must be processed in several steps, depending on the data source. The first step is to extract knowledge from raw data.

#### Named Entity Recognition

Named entity recognition (NER) refers to identifying mentions of named entities in an input text [82], [83], demarcating mentions of people, organizations, locations, and other types of entities [84], [85]. Figure 4.1 shows an instance in which an NER system identifies three named entities within a given sentence. Identified named entities are used to either generate new candidate RDF triples for the KG, or to link them to existing RDF triples using entity resolution (sometimes referred to as entity linking). Li *et al.* [84] formally define NER as follows:

> "Given a sequence of tokens $s = \langle w_1, w_2, \ldots, w_N \rangle$, NER is to output a list of tuples $\langle I_s, I_e, t \rangle$, each of which is a named entity mentioned in $s$. Here, $I_s \in [1, N]$ and $I_e \in [1, N]$ are the start and the end indexes of a named entity mention; $t$ is the entity type from a predefined category set."

For NER, two kinds of approaches can be distinguished, namely, knowledge-based and learning-based. Knowledge-based approaches [86], [87] utilize explicit resources such as dictionaries, rules, and gazetteers, which are often handcrafted. These methods rely on a predefined set of rules or lists that help in identifying and classifying named entities within the text. Dictionary-based methods map the labels of desired entities to identifiers in a knowledge base. Therefore, they provide recognized entities in a text with the right link to the knowledge base and thus solve the tasks of NER and entity resolution in one step. However, dictionaries are usually incomplete because they cannot possibly cover all entities, especially those that are newly emerging, less common, or highly specific to niche domains. This limitation means that knowledge-based systems may fail to recognize or incorrectly classify new or obscure named entities that do not exist in their predefined resources. Additionally, language evolves over time, introducing new terms and phasing out old ones.



< w1, w3, Person > Michael Jeffrey Jordan
< w7, w7, Location > Brooklyn
< w9, w10, Location > New York
< Is, Ie, t >
Named Entity Recognition
s = < w1, w2, ... , wN >
Michael Jeffrey Jordan was born in Brooklyn , New York .
w1 w2 w3 w4 w5 w6 w7 w8 w9 w10 w11

**Figure 4.1:** Illustration of the named entity recognition task. Adapted from Li *et al.* [84]

Gazetteers offer exhaustive lists that include, for instance, common first names, last names, or countries. Toral and Muñoz [88] distinguish between two kinds of gazetteers:

- *Trigger gazetteers* contain keywords that suggest the likely presence of a particular type of entity. These keywords usually are common nouns (e. g., "Ms." indicates that the following entity is a person).
- *Entity gazetteers* contain entities themselves, which usually are proper nouns (e. g., "New York" could be an instance in a location gazetteer).

Learning-based approaches use learning frameworks to recognize and categorize named entities. They can be further distinguished by supervised, bootstrapped, and deep learning-based approaches. Supervised approaches are based on a tagged corpus that was used to train a supervised learning algorithm, enabling the NER algorithm to improve its ability to recognize and categorize named entities in new texts by learning from examples. Lexical features, such as part-of-speech (POS) tags and dependency parse trees, provide contextual clues that help in the accurate identification and classification of entities. For instance, POS tags can indicate whether a word functions as a proper noun, which is an indication of a named entity [85]. Similarly, dependency parse trees reveal the grammatical relationships between words, aiding in the understanding of entity mentions that may span multiple words or phrases. Bootstrapped methods [85], on the other hand, only require a small set of seed examples of entity mentions from which patterns can be learned and applied to unlabeled text.

Current state-of-the-art NER approaches are deep learning-based. NER benefits from the nonlinear transformation, which generates nonlinear mappings from input to output: "The key advantage of deep learning is the capability of representation learning and the semantic composition empowered by both the vector representation and neural processing. This allows a machine to be fed with raw data and to automatically discover latent representations and processing needed for classification or detection [89]" [84, p. 5]. As a result, deep learning-based NER can save efforts on engineering NER features compared with traditional feature-based approaches.

### Relation Extraction

While NER serves to identify and extract entities (e. g., people, organizations), these entities remain isolated. The task of relation extraction is to determine and extract the relationship between these named entities [82], as illustrated in figure 4.2 for text data.

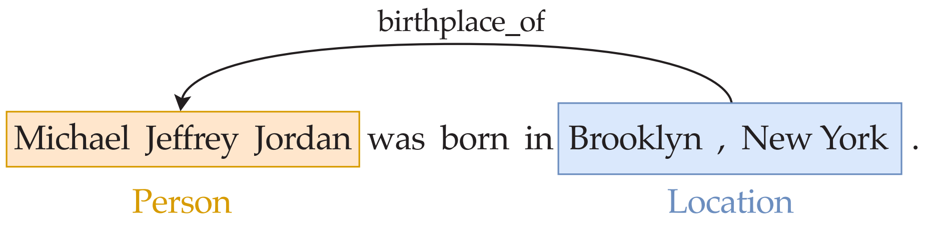


**Figure 4.2:** Illustration of the relation extraction task

Relation extraction varies by data type:

- Structured data presents relations in an explicit and easily identifiable manner, often found in databases where relationships are predefined.
- Semi-structured (e. g., JSON or XML) data, requires pattern-, rule-, and machine learning-based methods to identify relationships. These data types offer some level of structure that can be leveraged to extract relations.
- For unstructured, textual data, which comprises the majority of the data available on the internet (e. g., news articles, social media posts), the extraction process becomes more challenging. It requires interpreting semantic information embedded within the text.

However, the task of relation extraction is not without its challenges. These include, but are not limited to, dealing with linguistic variability, which refers to the different ways in which a relationship can be expressed; handling implicit relations that are not directly stated; disambiguating entities and relations to ensure accuracy; and transferring knowledge across domains.

### 4.1.2 Knowledge Processing

Following the extraction of knowledge, it must be processed and refined before it can be stored in the KG. This comprises two tasks: entity resolution and knowledge fusion. Entity resolution, also known as record linkage or deduplication, involves identifying and linking multiple representations of the same entity across different datasets or within a single dataset. Knowledge fusion is concerned with the integration of information about an entity from various sources into a coherent and unified representation within the KG. It involves resolving conflicts in the data obtained from different sources, which may vary in reliability, freshness, and accuracy.

#### Entity Resolution

Entity resolution aims to identify and link records across multiple datasets that refer to the same entities. This task is complex due to the vast amount of data and the discrepancies in how entities are represented across different sources. The standard approach to tackling this challenge is structured around three successive phases (illustrated in figure 4.3) [90]: blocking, matching, and clustering. Initially, the blocking phase aims to significantly reduce the number of entity pairs that must be evaluated. This is achieved through a partitioning strategy, ensuring that only entities within the same partition are compared with each other. The core of entity resolution lies in the linking or matching phase, where the main objective is to ascertain the similarity between pairs of entities to identify potential matches. This step often culminates in the formation of a similarity graph, where nodes symbolize entities and edges connect pairs deemed similar. Although not always included, an optional clustering phase leverages this similarity graph to aggregate all matching entities, thereby enhancing the quality of the entity resolution process.

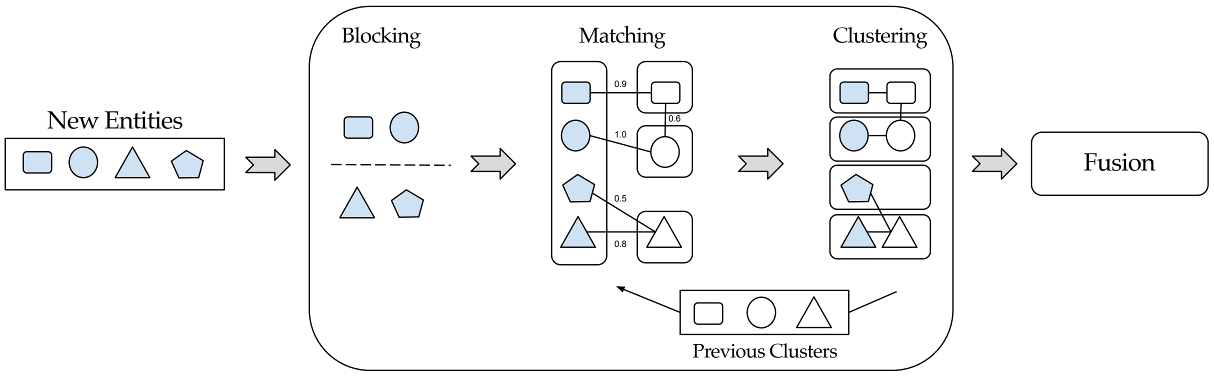


**Figure 4.3:** Illustration of the entity resolution task. Adopted from Hofer *et al.* [4]

**Knowledge Fusion**

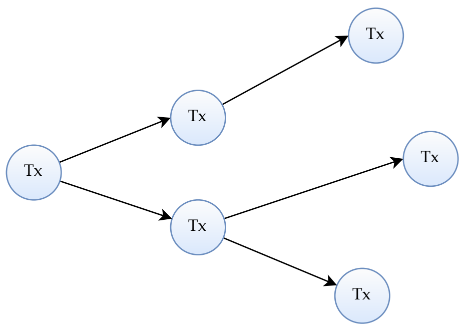
Merging multiple records of the same entity to identify true $(s, p, o)$ triples is known as data fusion [91], or, in the context of knowledge processing, as knowledge fusion [92]. Initially, there might be discrepancies in the naming of attributes across records, necessitating the selection of a single preferred name. This chosen name should align with the naming conventions of similar entities to simplify data querying. The next step involves addressing inconsistencies or disputes at the attribute level. Instead of resolving these conflicts directly, the system might preserve the differing attribute values or pass the issue to the user’s application for resolution. The system adopts a uniform approach to managing all data, giving precedence to information from reputable sources. Before implementing a specific strategy, such as selecting the most common, most recent, or a randomly chosen value, it thoroughly reviews all data and metadata. By fusing multiple records representing the same entity into a single and consistent representation [92], knowledge fusion allows for a more accurate representation of an entity in the final KG.

## 4.1.3 Ontology Development

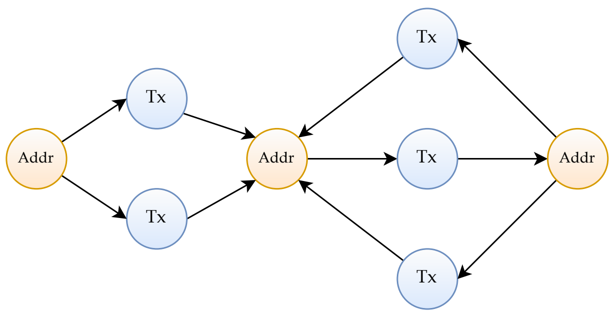
The first iteration of ontology development encompasses the creation or extension of an ontology that is applied to a KG. Subsequent iterations incorporate new information to reflect a growing understanding of the domain. The creation and curation of these ontologies currently rely on manual efforts or crowdsourcing, although there has been some movement towards semi-automatic approaches. Typically, the initial version of an ontology is rooted in a single source (e. g., public web wikis, catalogs, APIs, or databases curated by the crowd) providing a structured dataset from which the ontology is shaped. However, the transformation of this raw data into a coherent and comprehensive ontology necessitates cleaning and enrichment processes to ensure it adequately represents the intended domain and adheres to the high-quality standards required for KG integration. Another important step in the integration of ontologies is to match ontology and schema elements (e. g., classes) to find equivalencies.

## 4.2 Graph-based Blockchain Data Mining

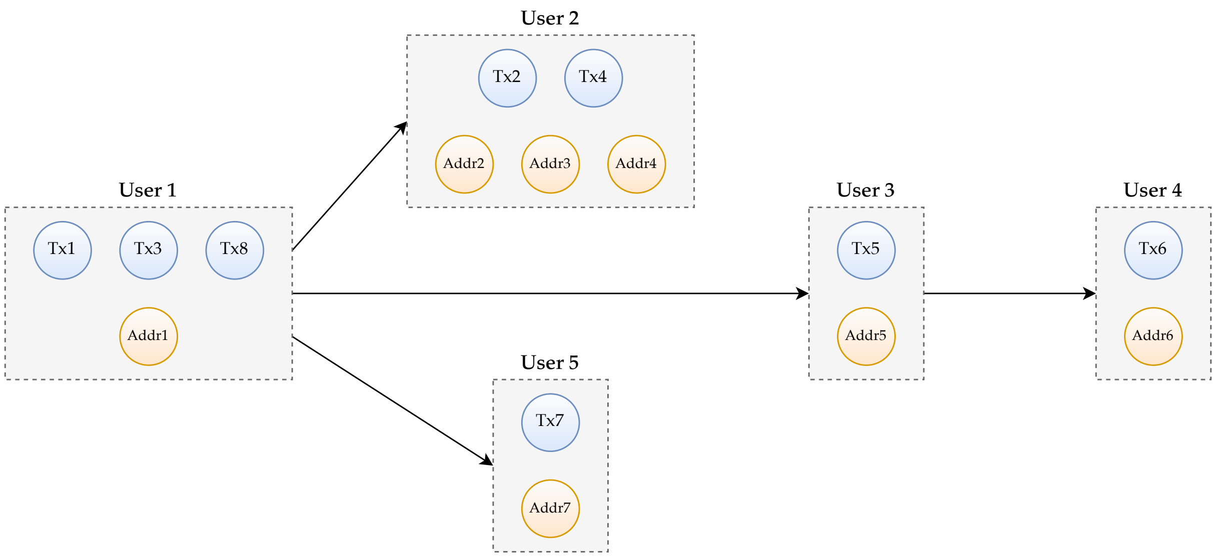
Graph theories and graph-based data mining methods are applicable for discovering information in blockchain network graphs because blockchain transactions can be easily structured into graphs [7]. Figure 4.4 depicts three types of graphs, identified by Elmougy and Liu [93], that are applicable to any blockchain network: money flow transaction graphs visualize the asset flow over time, address-transaction graphs show the flow of an asset across transactions and addresses, and user entity graphs that use deanonymization methods, outlined in section 4.2.1, to link addresses controlled by the same user to deanonymize their identity and purpose. These graphs serve as a foundation for downstream tasks such as transaction pattern recognition (section 4.2.2) to track and observe transactions from specific addresses, and illicit activity detection (section 4.2.3) to detect scams and other sorts of illicit activities [94].



**(a) Money Flow Transaction Graph**. This is a directed graph where nodes represent transactions and edges represent directed asset flows from one transaction to the next.



**(b) Address-Transaction Graph**. This is a heterogeneous directed graph with transaction (blue) and address (orange) nodes, and sender-to-transaction and transaction-to-receiver edges.



**(c) User Entity Graph**. This graph is generated by address clustering analysis over an address-transaction graph.

**Figure 4.4:** Overview of the different types of blockchain graph models

### 4.2.1 Blockchain Address Deanonymization

The objective of address deanonymization is to uncover the (real-world) entities behind pseudonymous addresses involved in blockchain transactions. The term "entity" was introduced by Ron and Shamir [95, p. 5] to describe the common owner of addresses.

#### Bitcoin Address Deanonymization

To deanonymize the identity of Bitcoin users, several studies [70], [96]–[98] have examined Bitcoin address clustering, a method that can potentially link addresses owned by a single user. Despite the increasing interest in applying machine learning techniques for deanonymization, the predominant methodology for address clustering remains rooted in heuristic-based approaches. These methods seek to deanonymize Bitcoin addresses by exploiting how wallet software creates transactions or utilizing the structural information of certain transactions to reason about entities involved in a transaction. The following Bitcoin heuristic focuses on the input side of transactions. Subsequent heuristics pivot to the output side of transactions, specifically for the identification of change addresses.

*Common-input-ownership*

The common-input-ownership heuristic, also called the *common spend*, *multi-input*, or *Nakamoto/Meiklejohn* heuristic, is predicated on the assumption that all input addresses to a transaction are owned by the same real-world entity. This assumption stems from the requirement for the initiator of a transaction to possess the private keys for all involved input addresses simultaneously. Such transactions, which involve multiple inputs, are typically necessitated when a single UTXO does not have a large enough value to cover the desired payment amount. For instance, as illustrated in figure 4.5, the three input addresses for the transaction can be attributed to the same real-world entity. Notably, the original Bitcoin whitepaper [43] referenced this heuristic, albeit with the inaccurate assertion of its universal applicability. The CoinJoin method presents a notable countermeasure to this heuristic, enabling multiple parties to jointly execute a transaction with inputs from distinct owners, thereby complicating the application of the common-input-ownership heuristic.

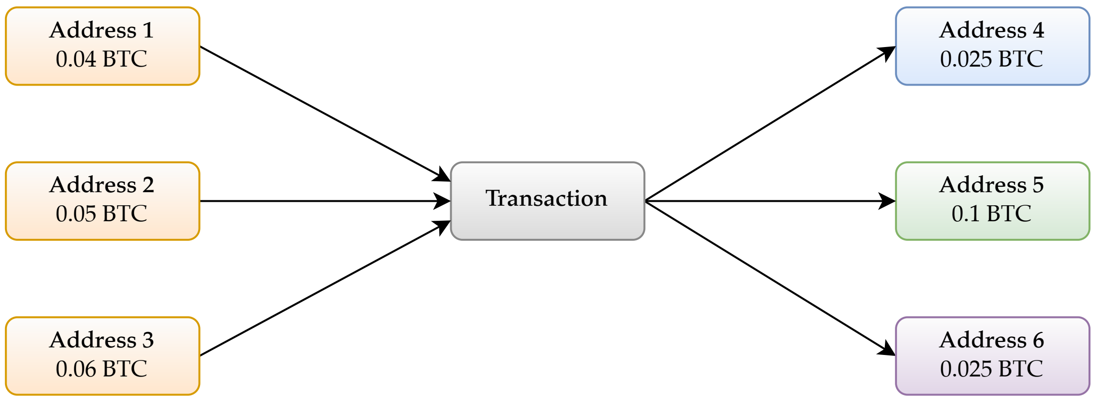


**Figure 4.5:** Illustration of the common-input-ownership heuristic

Möser and Böhme [80] argue that participants agreeing on a common output size in a transaction form a characteristic that distinguishes CoinJoins from normal transactions. Further, they note, the difficulty of finding matching subsets for a CoinJoin transaction increases as the number of participants increases. While enhancing anonymity, this in turn also makes the transaction more distinct compared with other transactions.

*Address Reuse*

The *address reuse* heuristic, often referred to as the *shadow heuristic*, centers on the observation of wallet software practices where, instead of generating a new change address for transactions, the change is redirected back to the originating input address. Such behavior conspicuously marks the change address, undermining the privacy of the transaction parties by making it easier to trace transactions back to them. Furthermore, when an address was never previously observed in a transaction, it is most likely a change address. This assumption is based on the rationale that the creation of a new address within a transaction typically aims to redirect the UTXO back to the sender as change. Additionally, this heuristic gains utility from the fact that most addresses subjected to reuse are often disclosed on various internet platforms,[1] including forums and social networks.

*Equal-output CoinJoin*

The equal-output CoinJoin heuristic involves a scenario in which input addresses are contributed by multiple distinct entities (cf. section 3.5.2). This method allows these entities to send identical amounts of funds to several addresses, thereby enhancing privacy and making it more challenging to trace transactions back to their origin. For example, as illustrated in figure 4.6, both "Address 21" and "Address 32" receive the same output value, which complicates the task of discerning which entity transferred funds to each address. Despite this obfuscation, it remains feasible to identify the change address by subtracting the total output value from the sum of input values. For instance, if the input adds to 0.011 (0.006 + 0.005) BTC and the output is 0.01 BTC, the difference of 0.001 can be attributed to the change address.

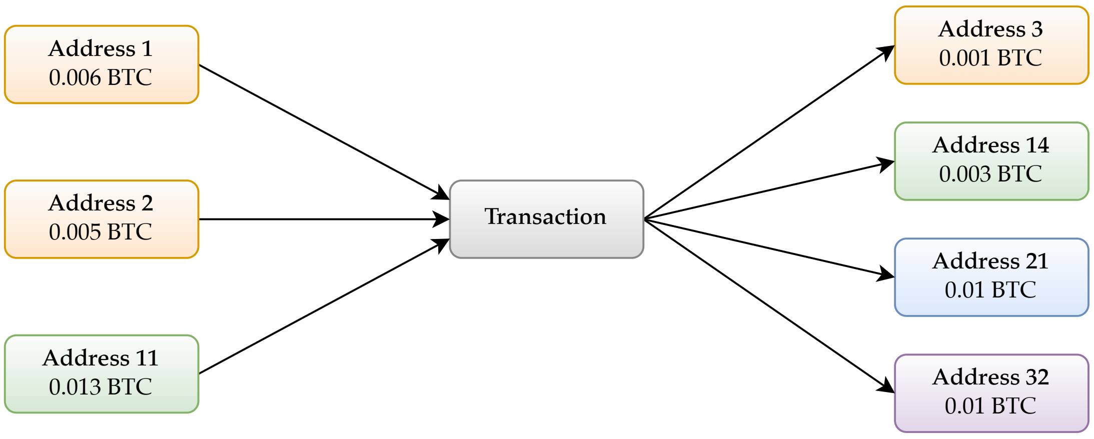


**Figure 4.6:** Illustration of the equal-output CoinJoin heuristic

[1]https://checkbitcoinaddress.com/

*Script Type*

The *script type* heuristic for Bitcoin transactions provides a method for identifying the change output in a transaction, which can be crucial for analyzing and understanding transaction flows on the blockchain. In transactions, it is common for an entity to utilize a single script type for all operations, meaning that the output that maintains the same script type as the inputs is typically the change address. This becomes particularly evident in transactions where inputs and outputs consist of differing script types; the output that shares the script type of the input is likely designated for change, while the other output is directed toward payment. Even in cases where inputs possess mixed script types, an output matching any script type of the input, while the other does not, suggests that the unmatched output is the intended payment. In figure 4.7, it can be inferred that "Address 4" is the change address as it is the only output address with the same script type as the input addresses. Early adopters of protocol upgrades, such as Pay-to-Script Hash (P2SH) or Pay-to-PubKey Hash (P2PKH), may inadvertently highlight their change outputs due to the scarcity of transactions sharing the same script type at the onset of the upgrade's introduction. However, this effect diminishes as the new address types become more widely adopted.

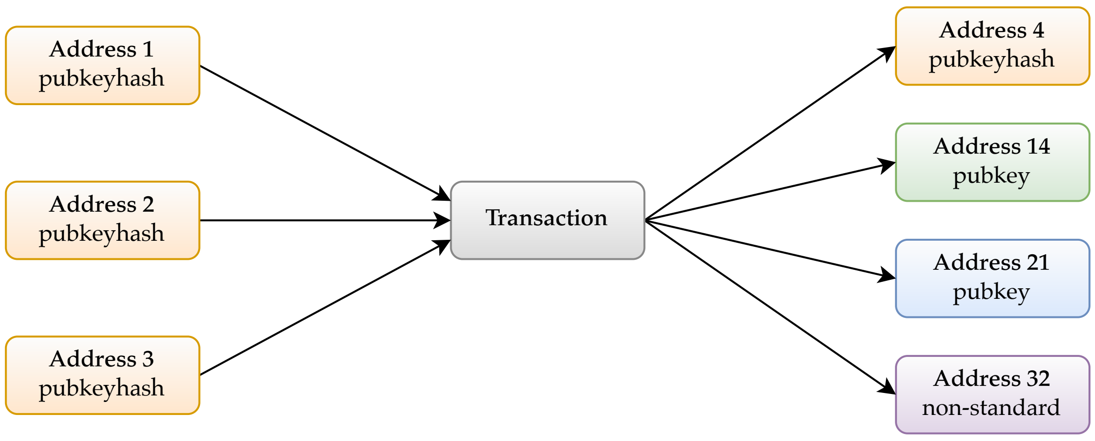


**Figure 4.7:** An illustration of the script type heuristic

*Optimal Change*

The *optimal change* heuristic (also known as the *unnecessary input* heuristic) is premised on the assumption that wallet software is designed to optimize transactions by avoiding the expenditure of unnecessary outputs. This entails wallets strategically selecting outputs to not only cover the transaction amount but also to ensure the total surpasses the intended sum to be sent. In line with this, optimal transactions are characterized by the exclusion of outputs that, if not spent, would still fulfill the desired transaction value, thereby circumventing additional fees. Moreover, it stipulates that transactions should not feature change values exceeding the amount of any spent output as this indicates that certain outputs could have been omitted to minimize fees. Therefore, discerning the smallest change value as the "optimal change output" can aid in identifying the true change address within a transaction.

*Round Numbers and Decimal Places*

Typically, change addresses exhibit a higher number of decimal places compared with other outputs. This discrepancy arises both from the mechanics of transaction fees, which necessitate fine adjustments to the transaction amount, and from the prevalent human preference for using round numbers when specifying transaction amounts. While earlier transactions might have shown a tendency to reduce the precision for change addresses, the current norm involves all output addresses featuring decimal points to some extent. Identifying change addresses accurately is crucial and requires precise parameters. The methodology of the *round numbers and decimal places* heuristic entails a search for addresses that contain more than seven decimal places and comparing these with addresses with fewer than two decimal places to ascertain the likelihood of an address being used for change. This heuristic, despite its simplicity and potential for exceptions, has shown to be effective if there is a is a clear discrepancy in the decimal length parameters.

**EVM Address Deanonymization**

The domain of address deanonymization has predominantly been concentrated on Bitcoin. Despite the widespread adoption of EVM blockchains, research efforts directed toward address deanonymization and clustering have been minimal. In this context, Victor [99] proposed heuristics for identifying real-world entities within the Ethereum network, specifically through methods such as deposit address reuse and airdrop multi-participation. Among these, the reuse of deposit addresses emerges as the most effective strategy for entity identification, whereas the approach based on transfer authorization is deemed negligible due to its minimal address coverage and is thus not covered here. To address coin mixing, Tang *et al.* [72] proposed heuristics based on statistical analysis for linking Ethereum addresses that interacted with Tornado Cash.

*Deposit Address Reuse*

The *deposit address reuse* heuristic for Ethereum, proposed by Victor [99], provides a methodological approach to identify deposit addresses associated with centralized exchanges. This heuristic is predicated on the understanding that when customers wish to deposit their assets into their accounts on a centralized exchange, they must first transfer their assets to a specific deposit address linked to their account. This deposit address, which is controlled by the exchange and can be either an EOA or a smart contract, is where the assets remain until they are automatically transferred to the hot wallet[2] owned by the exchange. Further, deposit addresses are typically unique to each customer, suggesting that multiple addresses funneling funds to the same deposit address are likely under the control of a single entity. The heuristic focuses on the forwarding principle — tracking asset flows from deposit addresses to the main accounts of exchanges — to cluster addresses generated by centralized exchanges for the purpose of receiving customer deposits.

[2] Hot wallets are connected to the internet for everyday transactions. In contrast, cold wallets are offline and typically stored on hardware devices to provide secure long-term storage for crypto assets.

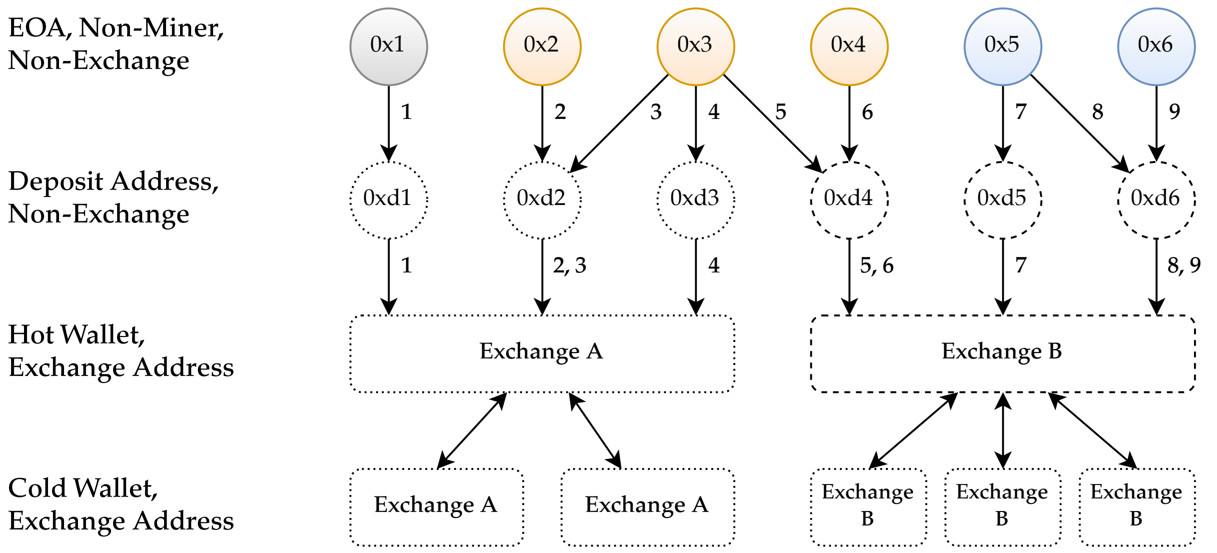


**Figure 4.8:** Illustration of the deposit address reuse heuristic, adopted from Victor [99], and extended with the fund-gathering pattern by Wang *et al.* [100]. Colors indicate the same real-world entity. Arrows indicate the flow of crypto assets between blockchain addresses.

Wang *et al.* [100] expand on this concept by identifying deposit addresses as part of a fund-gathering pattern, which aids in identifying the related hot and cold wallets of an exchange. They assert that the public availability of information regarding hot and cold wallets (e. g., through platforms like Etherscan for Ethereum) is instrumental in labeling hot wallets accurately. Once a hot wallet is identified, it becomes feasible to pinpoint customer deposit addresses, which characteristically transfer assets exclusively to the hot wallet. Using this heuristic, figure 4.8 depicts an example where 0x2–0x4, as well as 0x5 and 0x6, are indicated to be owned by the same real-world entities, respectively.

*Airdrop Multi-Participation*

An airdrop is a method to automatically distribute tokens to multiple recipients on the Ethereum blockchain using smart contracts. The determination of token allocation to recipients can be: (1) *tiered systems*, where different levels of involvement in the protocol grant users different allocation tiers; (2) *participation*, where users who actively participate in the protocol might receive more tokens; or (3) *fixed amounts* to a broad group of users, regardless of their level of interaction with the protocol. Post-airdrop, it is a common practice among recipients to consolidate their dispersed tokens into a single address for convenience or management purposes [99]. This behavioral pattern of consolidation presents an opportunity to identify individual real-world entities that may receive tokens multiple times. This identification is achieved by tracking the transfer of tokens from their initial recipient addresses to secondary ones. However, Victor [99] emphasizes the importance of carefully excluding addresses controlled by centralized exchanges, smart contracts associated with decentralized exchanges, and, generally, addresses exhibiting no activity from this analysis to avoid false positives. Figure 4.9 illustrates an example where 0xa1–0xa3, 0x1 and 0xa4–0xa6, as well as 0x2 and 0xa7–0xa10, are indicated to belong to the same real-world entities, respectively.

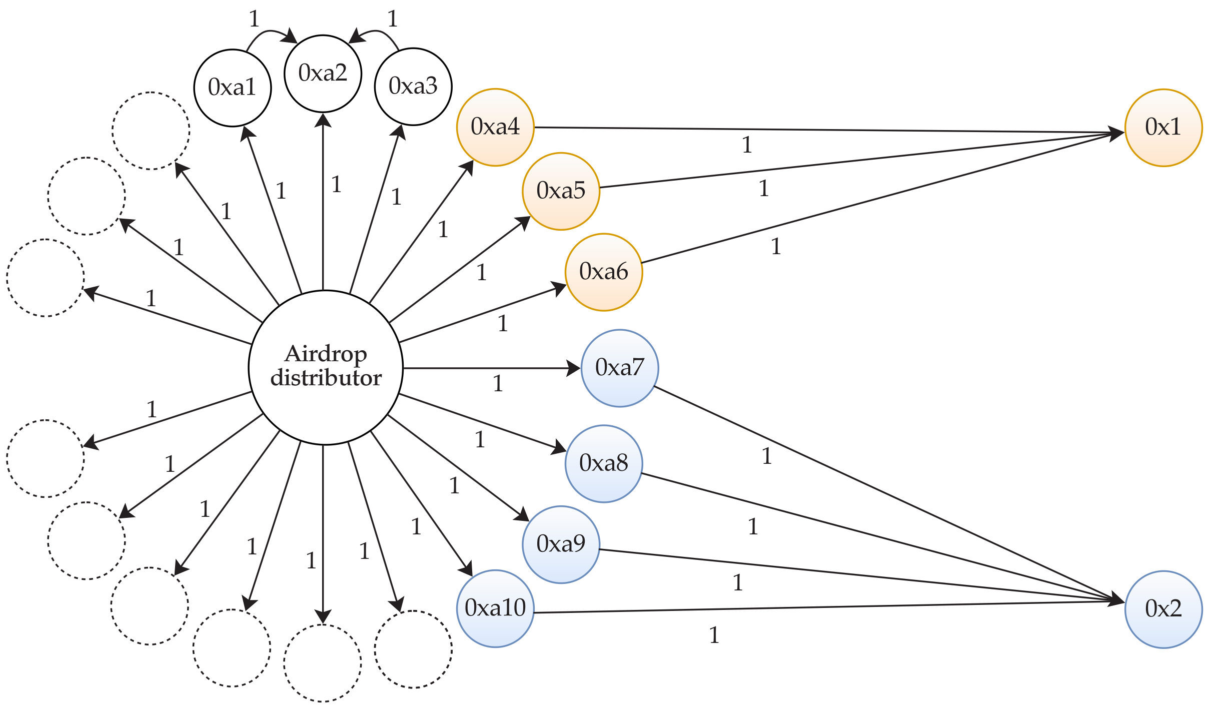


**Figure 4.9:** Illustration of the airdrop multi-participation heuristic. Colors indicate the same real-world entity. Adopted from Victor [99]

Tang *et al.* [72] conducted an analysis of transaction data from Ethereum, focusing specifically on interactions with the coin mixing service Tornado Cash. During their research, they identified two transaction behavior patterns: single deposit-withdraw, and multi-deposit and multi-withdraw coin mixing. For each of these behavioral patterns, they developed clustering heuristics as follow.

*Single Deposit-Withdraw Coin Mixing*

A single deposit-withdraw involves a deposit transaction ($d$), where a user deposits an amount of ETH using address $a$. After a certain time interval (denoted as $\delta$), the user withdraws the same amount of ETH in a withdraw transaction ($w$) from a different address $b$. This pattern is captured by a triplet $\langle d, w, \delta \rangle$, with $\delta$ representing the time difference between the withdraw and deposit transactions ($\delta = w.\text{ts} - d.\text{ts}$). The deposit ($d$) and withdraw ($w$) transactions must meet the following conditions:

- The source of the deposit ($d$.from) must be the address $a$.
- The recipient of the input for the withdraw transaction ($w$.input.recipient) must be the address $b$.
- The destination of the deposit transaction ($d$.to) must be the same as the destination of the withdraw transaction ($w$.to).

The addresses in the transaction pairs $\{w.\text{input.recipient}, d.\text{from}\}$ are considered to belong to the same user if $\delta \leq 180s$.

*Multi-Deposit and Multi-Withdraw Coin Mixing*

In this scenario a user performs a series of deposit transactions, at least $n(\geq 2)$, denoted as $D = \{d_1, d_2, \ldots, d_n\}$. These deposits are made to a set of addresses $A = \{a_1, a_2, \ldots, a_n\}$. After a certain interval ($\Delta$), the same user executes a series of withdraw transactions corresponding in number to the deposit transactions ($W = \{w_1, w_2, \ldots, w_n\}$) using a different set of addresses $B = \{b_1, b_2, \ldots, b_n\}$. The pattern is represented by a sequence $\langle \delta_d, D, \delta_w, W, \Delta, n \rangle$, where:

- $\delta_d$ is the largest time difference between any two deposit transactions.
- $\delta_w$ is the largest time difference between any two withdraw transactions.
- $\Delta$ is the time difference between the first withdraw transaction and the last deposit transaction.

For this heuristic, the following conditions must be satisfied to consider the addresses $\{d_1.\text{from}, \ldots, d_n.\text{from}, w_1.\text{input.recipient}, \ldots, w_n.\text{input.recipient}\}$ in the $n$ deposit and withdraw transactions to belong to the same user:

- $d_1.\text{from} = d_2.\text{from} = \cdots = d_n.\text{from}$;
- $w_1.\text{input.recipient} = w_2.\text{input.recipient} = \cdots = w_n.\text{input.recipient}$;
- $\delta_d, \delta_w \leq 10\text{min}$ and $\Delta \leq n * 12h$.

**Cross-Chain Address De-Anonymization**

Yousaf *et al.* [101] conducted a study on tracing transactions across cryptocurrency ledgers. Specifically, they focused on the ability to link together the blockchains of multiple different cryptocurrencies using trading platforms like *ShapeShift* [102] and *Changelly* [103] that allow users to exchange their cryptocurrency, with the possibility that they receive their exchanges cryptocurrency on another blockchain. For this study, they collected data by engaging with ShapeShift and Changelly and collecting data from their interactions via the APIs. Additionally, they downloaded and parsed the blockchain data of eight different blockchains, resulting in a total of 434 GB, to examine on-chain transactional behavior.

They observed various behavioral patterns in cross-currency transactions conducted on platforms like ShapeShift (depicted in figure 4.10): pass-through, U-turn, and round-trip transactions. Pass-through transactions represent the flow of money from one currency to another via deposit and withdrawal transactions. U-turns occur when a user shifts into one currency but then immediately shifts back, linking two transactions within a short timeframe and similar value. Round-trip transactions involve a user sending money to another currency and then back to the original one, identifying the movement of funds across different ledgers. Similar to mixing services, Yousaf *et al.* [101] define the process in two phases. Phase 1 refers to the deposit of coins from the user to the service on the input blockchain, while phase 2 refers to the withdrawal of coins from the service to the user on the output blockchain.

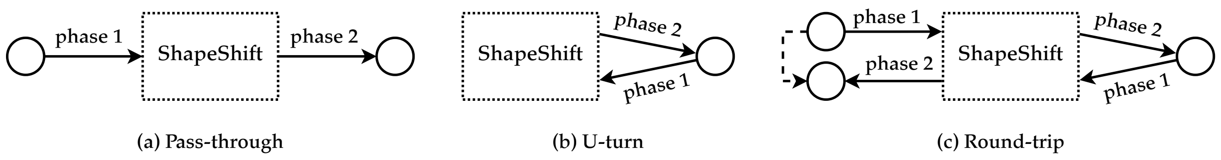


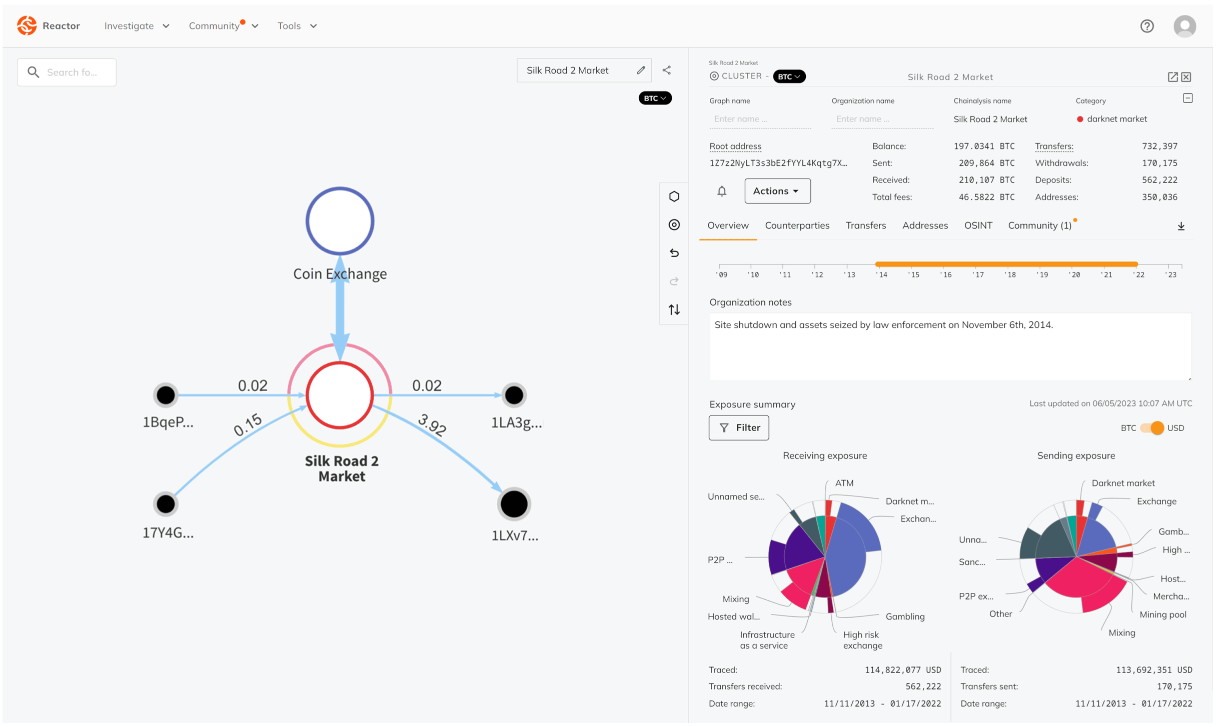
**Figure 4.10:** Cross-chain transaction patterns. Adopted from Yousaf *et al.* [101]

Additionally, Yousaf *et al.* [101] defined a *common relationship* heuristic for relationships between individual ShapeShift users, where they consider what it means for two different addresses, in potentially two different blockchains, to have sent coins to the same address; they refer to these addresses as belonging in the *input cluster* of the output address. Analogously, if a single address sends coins to multiple addresses, they refer to it as *output cluster*. Based on these clusters, the authors present three possible scenarios of common relationship:

1. In a closest link, the activity might indicate that a single user is consolidating funds from different currencies into one.
2. Alternatively, there could be interactions among various users with a shared service, such as a cryptocurrency exchange.
3. Another possibility is that two unrelated users might coincidentally send money to the same recipient, suggesting a coincidental overlap rather than a meaningful connection between the users.

### 4.2.2 Transaction Pattern Recognition

*Transaction pattern recognition* deals with tracking and observing transactions from specific addresses in order to identify patterns within the blockchain network [94]. Thereby, this method provides insights into standard and anomalous behaviors within the network. The primary approach for the recognition of transaction pattern is through the use of visualization software by leveraging graphical representations to elucidate the transactions and interactions between various addresses on a blockchain network. The core premise behind visualization is that by translating complex transactional data into a visual format, stakeholders can more intuitively comprehend and identify patterns, trends, and anomalies within the network. This direct observation from the visualization results facilitates a better understanding of network behaviors, enabling users to spot irregularities that may suggest fraudulent activities or other notable events. Several solutions exist to visualize blockchain transactions, for instance, the *Visualizer* by Arkham Intelligence [104] and *Reactor* by Chainalysis [1] (depicted in figure 4.11). They offer functionalities for zooming into specific transactions, filtering by certain criteria, and highlighting the connections between different network participants. Such capabilities empower analysts, regulators, and other stakeholders to make informed decisions based on a comprehensive visual analysis of the blockchain network. Through the strategic use of visualization methods, stakeholders involved in the blockchain ecosystem can bolster their monitoring and investigative efforts.



Silk Road 2 Market
CLUSTER - BTC Silk Road 2 Market

Graph name: Enter name ...
Organization name: Enter name ...
Chainalysis name: Silk Road 2 Market
Category: darknet market

Root address: 1Z7z2NyLT3s3bE2fYYL4Kqtg7X...

| | | | |
|---|---|---|---|
| Balance: | 197.0341 BTC | Transfers: | 732,397 |
| Sent: | 209,864 BTC | Withdrawals: | 170,175 |
| Received: | 210,107 BTC | Deposits: | 562,222 |
| Total fees: | 46.5822 BTC | Addresses: | 350,036 |

Organization notes
Site shutdown and assets seized by law enforcement on November 6th, 2014.

Exposure summary
Last updated on 06/05/2023 10:07 AM UTC
BTC USD

| | | | |
|---|---|---|---|
| Traced: | 114,822,077 USD | Traced: | 113,692,351 USD |
| Transfers received: | 562,222 | Transfers sent: | 170,175 |
| Date range: | 11/11/2013 - 01/17/2022 | Date range: | 11/11/2013 - 01/17/2022 |

**Figure 4.11:** Reactor blockchain forensic software by Chainalysis

### 4.2.3 Illicit Activity Detection

The detection of illicit activities within blockchain networks has garnered considerable attention in recent academic research, showcasing a variety of innovative methods aimed at identifying fraudulent transactions and scam accounts. As depicted in the introduction chapter in figure 1.1, scams are the most common category of crypto asset fraud. Therefore, a wealth of research efforts has been devoted to detecting scams with crypto assets. Pham and Lee [105] marked an early attempt in this field by applying the trimmed k-means algorithm to unearth fraud within the Bitcoin network, utilizing hand-crafted features derived from the transaction network. Progressing further, Chen *et al.* [106] expanded the scope to Ethereum, where they developed techniques for Ponzi scheme detection by analyzing both account and code features extracted from transactions and opcodes, respectively. This allowed for the early identification of such financial scams. Recognizing scam account detection as a node classification challenge, some researchers have turned to network embedding methods to automate the feature extraction process from transaction networks. A notable advancement was made by Chen *et al.* [107], who employed a Graph Convolutional Network (GCN) to detect phishing scams in Ethereum, surpassing the efficacy of previous methods reliant on handcrafted features. Concurrently, Tam *et al.* [108] introduced *EdgeProp*, a GCN-based approach that not only learns embeddings for nodes and edges within transaction networks but also integrates edge attributes to enhance the identification of illicit accounts and decipher transaction patterns, specifically in Ethereum. Additionally, Lin *et al.* [109] have contributed to this evolving field by developing random walk-based embedding techniques that consider transaction-specific attributes like amount, timestamp, and multi-edge interactions, proving to be effective in phishing detection.

Apart from detecting scams, other methods using graph-based methods have been proposed to detect money laundering (e. g., [110]), using methods like *deepwalk* and *node2vec*, and other illicit activities such as the use of cryptocurrencies in ransomware payments (e. g., [111]) based on topological information.

## 4.3 Summary

The state of the art in KG construction encompasses a comprehensive methodology that includes data acquisition, knowledge extraction (with subtasks such as NER and relation extraction), knowledge processing (featuring entity resolution and knowledge fusion), ontology management (beginning with the creation of an initial ontology and its subsequent incremental development), and quality assurance to ensure the high data quality within the KG. This sequential process facilitates the systematic transformation of raw data into a structured KG.

In parallel, graph-based data mining methods have been developed for transaction graphs, focusing on address deanonymization, transaction pattern recognition, and detection of illicit activities. Despite the success of these techniques in their respective areas, there remains a notable gap in the application of KG-based approaches to the detection of crypto asset fraud. Such an approach could potentially offer enhanced expressivity and a more nuanced understanding of fraudulent activities by leveraging the semantic richness of KGs. At present, there is currently no pipeline available for incrementally constructing a KG from blockchain data.

# Chapter 5

# Rug Pull Prevention Use Case

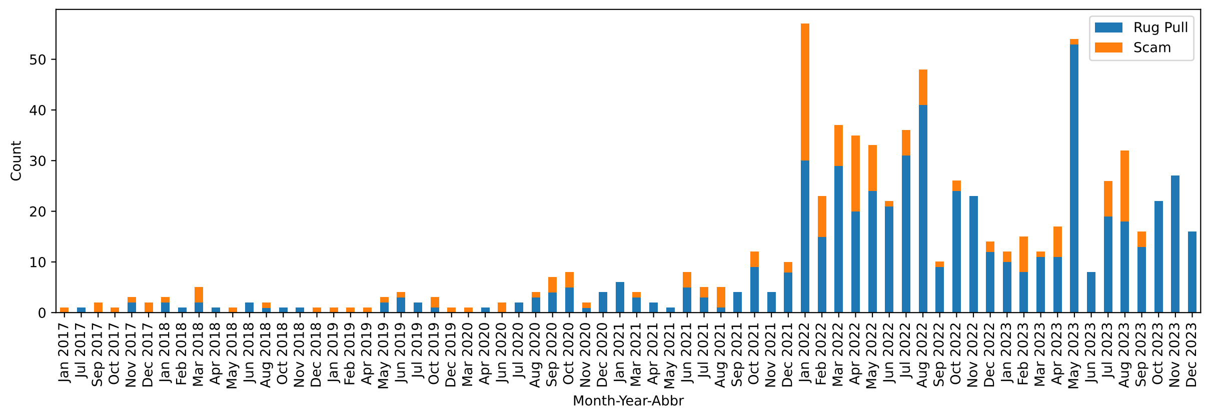
A rug pull can be categorized as a scam, where the victim authorizes the transaction. This type of scam is typically conducted in five stages, according to [56]:

1. Project creation with roadmap and total supply of tokens (optional)
2. Pre-mint hype
3. Setting the token mint price
4. Token mint, accumulation of more capital, and increase in popularity
5. Creators cash out, abandon the project, and leave the investors defrauded

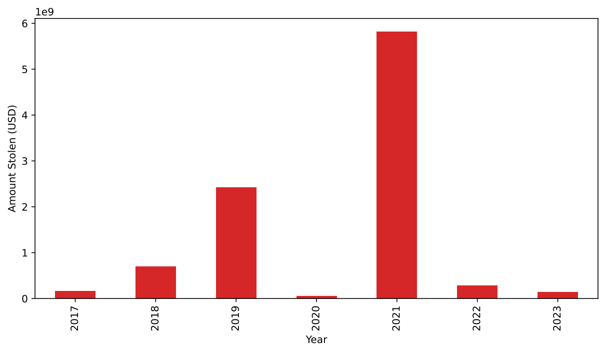
To attract users and investments for rug pulls, Sharma *et al.* [56] suggest the involvement of individuals or groups that possess substantial technical skills and knowledge of blockchain technology and demonstrate a proficiency in marketing techniques. This specific use case is particularly relevant given the findings in [56] and [55]: Mazorra *et al.*, who analyzed ERC-20 tokens listed on decentralized exchanges in their 2022 study and labeled 97.7% out of 27,588 analyzed tokens as rug pulls [55]. Similarly, Sharma *et al.* analyzed NFTs and identified a cluster of 168 of them associated with what they termed the "Rug-Pull Mafia," a group of creators responsible for orchestrating multiple and repeated rug pulls [56]. There is a growing trend in both the frequency and the financial impact of crypto rug pulls and scams [112], illustrated in figure 5.1. Notably, 2021 marks a peak in the amount stolen, while 2022 shows a sharp rise in the frequency of these fraudulent activities, which has remained elevated since.

To illustrate the vision of Kosmosis-enabled rug pull prevention methods, this section introduces a hypothetical user story[1] centered around a character named Bob, a crypto market participant. This user story is designed to provide a relatable perspective on how individuals like Bob are affected by such fraudulent activities. The story of Bob, while fictional, is grounded in a series of real-world rug pulls that occurred in 2021. All rug pulls were performed by the same fraudulent NFT creator and 𝕏 user known as Homer_eth. The following sections outline how the series of rug pulls experienced by Bob might have unfolded differently had he been equipped with a Kosmosis-enabled fraud prevention mechanism at the time.

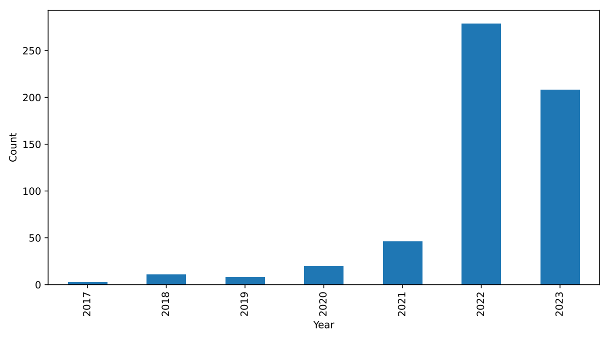
[1]The user story method of use case illustration is adopted from our previous work in [113], [114]



**(a)** Number of Rug Pulls & Scams by Month and Year



**(b)** Amount Stolen at Time of Rug Pull by Year (USD)



**(c)** Number of Rug Pulls by Year

**Figure 5.1:** Rug pulls and scams since 2017, using data from comparitech [112]

## 5.1 Past User Story

In the span of two months, from October to November 2021, a fraudulent NFT creator and 𝕏 user known as Homer_eth executed five different NFT project rug pulls, accumulating over $2.8 million in profits. Table 5.1 provides an overview of Homer_eth’s rug pull projects, each with the launch date and estimated profit from rug pulling the projects. The transaction graph in figure 5.2 shows the blockchain addresses, depicted as EOA nodes, that the rug puller used and how they are connected through transactions. Dashed arrows represent aggregated transactions, meaning the total value that was transferred between two addresses over multiple transactions. Notably, the graph is disconnected because there are no transactions that connect 0xf580 and 0xc8a6 to the other addresses controlled by Homer_eth.

| Project Name | Launch Date | Estimated Profit (USD) | NFT Collection Size |
|---|---|---|---|
| Ether Bananas | 10/07/2021 | 125,000 | 750 |
| Ether Monkeys | 10/11/2021 | 1,770,000 | 10,000 |
| Zombie Monkeys | 10/15/2021 | 413,000 | 8888 |
| Ether Reapers | 10/20/2021 | 282,000 | 10,000 |
| ETH Banana Chips | 11/23/2021 | 208,000 | 5000 |

**Table 5.1:** Rug pull projects by Homer_eth

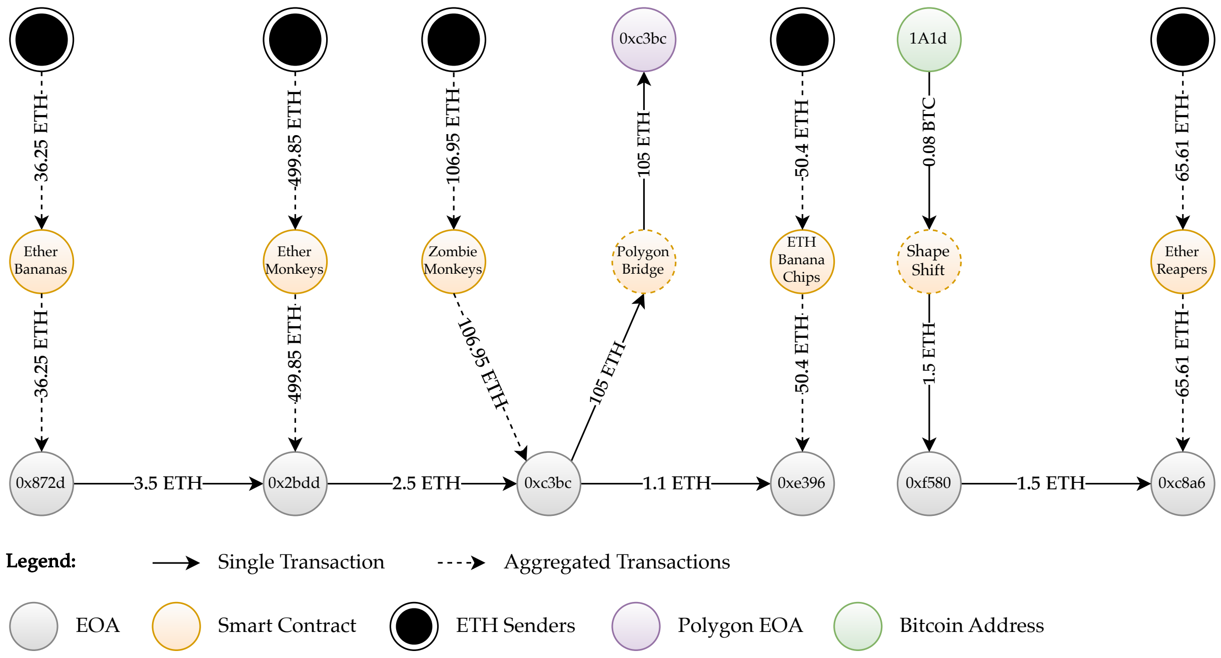


**Figure 5.2:** Simplified transaction graph of Homer_eth's NFT rug pulls, showing only transactions that are directly related to the rug pulls

Bob's story begins with a common enthusiasm for the burgeoning world of NFTs. His journey into the NFT market is marked by excitement and optimism, spurred by the success stories he sees online. Homer_eth, an NFT creator and 𝕏 user, has caught the attention of many people like Bob by sharing his NFT projects on 𝕏. His first NFT collection *Ether Bananas*, consisting of 750 NFTs, was launched on October 7, 2021.

Only four days later, on October 11, Homer_eth continued with the release of *Ether Monkeys*, followed by the release of *Zombie Monkeys*. The buzz around Homer_eth's projects, especially *Ether Monkeys*, which promised additional utility through a casino to gamble in and a decentralized autonomous organization to govern the NFTs, according to [115], attracted Bob. Being relatively new to the NFT market, Bob viewed this as an opportunity not to be missed. He bought his first NFT from Homer_eth, an *Ether Reapers*, and with that purchase, he was no longer just a bystander; he was now an active participant in Homer_eth's growing community.

Bob's involvement in the community deepened over time. He engaged in discussions, shared his excitement with fellow members, and reveled in the rumors of more NFT launches in the future. His commitment paid off when he earned himself a whitelist spot that allowed him to mint the upcoming NFT project *ETH Banana Chips* by Homer_eth. Convinced of its potential, Bob did not hesitate to mint an *ETH Banana Chips* NFT when the opportunity arose. With a click to confirm the transaction in his browser wallet (e. g., MetaMask [116]), Bob became the owner of an *ETH Banana Chips* NFT, unaware of the underlying risks associated with his investment. However, the reality of the situation was far from the optimistic scenario Bob had envisioned. Unknown to him, since Bob had a limited understanding of blockchain transactions, the proceeds from the *Ether Reapers* mint were not being locked in the smart contract for future development as promised. Instead, they were directly funneled into the deployer address owned by Homer_eth. From there, he would either later transfer those mint proceeds to a new deployer address to fund his next rug pull or bridge the proceeds to another blockchain, in an attempt to obfuscate his trail (known as chain hopping), and deposit them to a centralized exchange to pay out his profits.

After the launch of *ETH Banana Chips*, a tense silence enveloped the community. For months, there was no news from Homer_eth, no updates on the project, leaving everyone to wonder about the future. It was not until March 2022 that Homer_eth broke the silence with the announcement of one last NFT project, dubbed *Froggy Frens*. Due to backlash from the community, he deleted his 𝕏 account and vanished [115].

## 5.2 From Transaction Graph to Knowledge Graph

Methods to detect and prevent rug pulls require a semantically rich knowledge base to discern patterns of fraudulent behavior from licit behavior. This includes differentiating blockchain addresses by their users (individuals, protocols, exchanges) and the different ways they interact with the blockchain. Therefore, the KG depicted in figure 5.3 should semantically describe blockchain transactions to discern different types of transactions: value transactions, such as *mintMonkey* and *Transfer*, and non-value transactions (absent in transaction graphs), such as the deployment of a smart contract, denoted as *Deploy*. These semantics allow it to describe (i. e., tag) sender and receiver addresses as NFT minter and deployer, compared with the previously less descriptive *ETH Sender* and *EOA* in figure 5.2. Additionally, the KG integrates off-chain data from the social media platform 𝕏 to provide a deeper understanding of the context and relationships surrounding these rug pulls. The data integrated from 𝕏 extends the KG with off-chain user accounts (*X Account*) and their posts (*X Post*). For instance, this enables users to establish a connection among previously unrelated entities, such as the deployer address *0xc8a6*, the 𝕏 account Homer_eth, and his other blockchain addresses. Both, blockchain addresses and social media accounts are connected to a real-world entity in the KG. For brevity, these connections are not depicted in figure 5.3.

## 5.3 Alternative User Story

In an alternative scenario where Bob would have had access to Kosmosis-enabled rug pull prevention, his introduction into the NFT market would have been safer, beginning with his initial transaction to purchase an *Ether Reapers* NFT.

As soon as Bob initiated his transaction, the rug pull prevention mechanism would have accessed the KG to analyze the rug pull risk of the contract. Based on the integrated knowledge from 𝕏, the system would have been able to link the contract Bob was about to interact with to all of Homer_eth’s prior blockchain activity. The KG would have revealed a critical anomaly. Instead of the mint proceeds being transferred to the contract address of the project for future development, they were being diverted to the *Ether Reapers* deployer address via the *MintReaper* function. With smart contracts

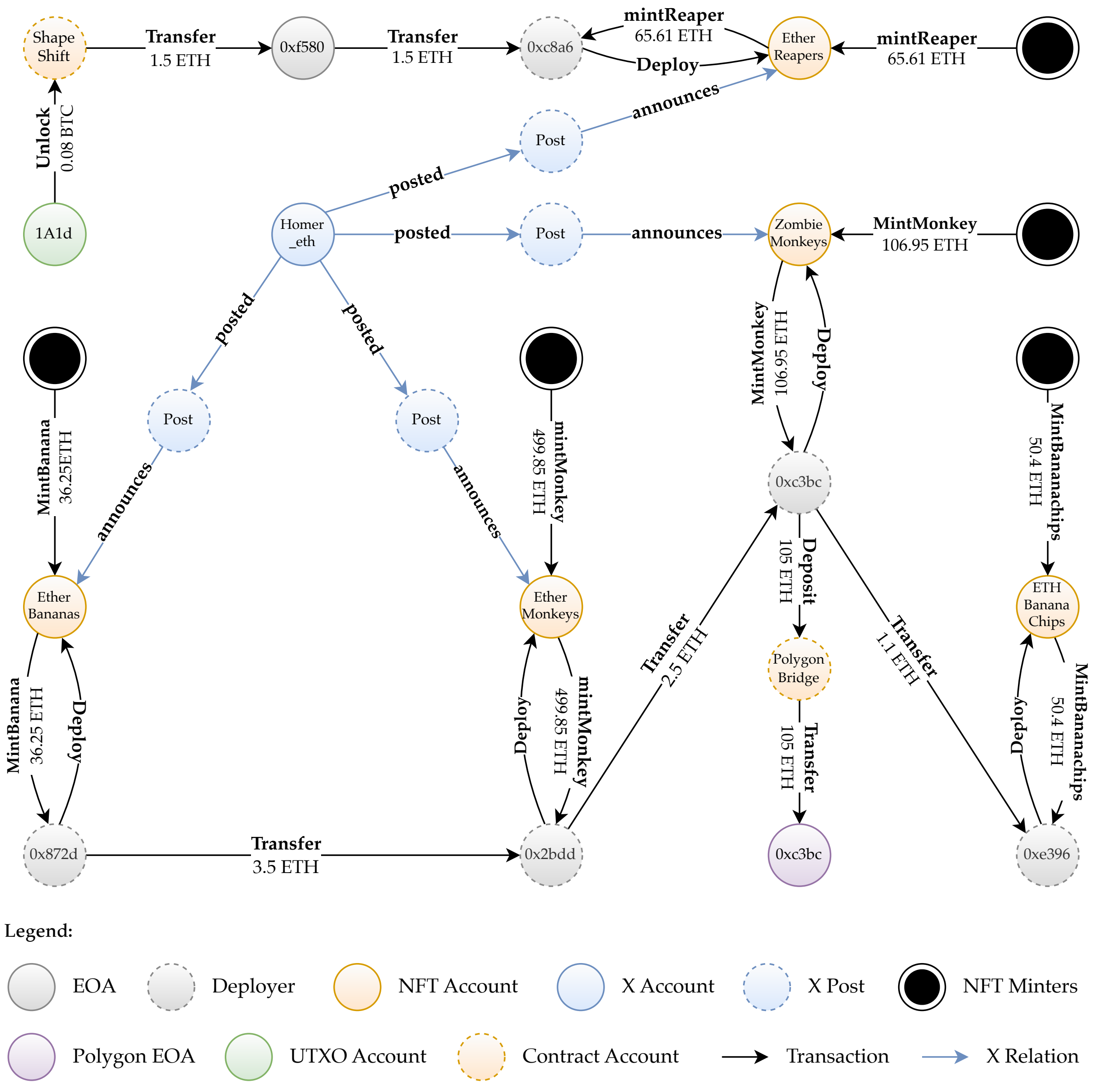


**Figure 5.3:** KG of Homer_eth's NFT rug pulls

acting as an automated and trustless intermediary, where the code of the contract dictates the flow of funds according to predefined rules, this pattern of fund diversion is absent in legitimate projects. When funds are sent directly to the address of a team member — in this case, the deployer address — they can be moved to exchanges or other addresses with ease (i. e., pulling liquidity from the project without fulfilling the promises). This is a common tactic in rug pulls, where the developers abandon the project and disappear with investor funds, therefore, signaling potential rug pull behavior. Detecting this anomaly, the system would have immediately issued a rug pull warning to Bob, prompting him to make an informed decision by asking whether he wishes to proceed with the transaction despite the identified risk. This proactive approach empowers Bob to reconsider his decision with full awareness of the potential danger, offering him a chance to opt out before potentially falling victim to a rug pull.

## 5.4 Technical Implications

The previous section introduced an alternative user story, which set the foundation for exploring the technical implications for the Kosmosis architecture. To address these implications, the KG must effectively combine on- and off-chain data, a process underpinned by an ontology modeled through CQs following the competency question-driven ontology authoring methodology, as outlined in section 5.4.1. Furthermore, to ensure the KG construction pipeline operates as intended, functional and non-functional requirements are compiled and systematically outlined in sections 5.4.2 and 5.4.3, respectively.

### 5.4.1 Competency Questions

The following CQs are formulated based on the user story presented in section 5.3 to ensure the ontology supports the detection of potential rug pulls.

1. **Binary questions**

   CQ1 *Does the address 0x872d belong to the real-world entity Homer_eth?*
   This tests whether blockchain addresses can be successfully linked to their corresponding real-world entity. The system should answer with `true`. Likewise, `true` should be returned for Homer_eth's other Ethereum addresses.

   CQ2 *Did 0xc8a6 deploy the NFT account Ether Reapers?*
   The system is expected to return `true`. This question explores whether the semantics in blockchain transactions (e. g., deploy, mintReaper) get extracted.

2. **Selection questions**

   CQ3 *Which blockchain addresses belong to Homer_eth on the Ethereum blockchain?*
   This question probes if identical addresses on multiple blockchains can be uniquely identified. For instance, the address 0xc3bc should be returned only once.

   CQ4 *How are the blockchain transactions of Homer_eth linked to his social media accounts and posts on the social media platform $\mathbb{X}$?*
   This question tests whether the system can connect blockchain addresses to social media activities. The system should return the RDF triples that describe that Homer_eth created at least one post on $\mathbb{X}$, and in this post, one of his NFT contract addresses is mentioned.

   CQ5 *Can the abnormal transaction pattern of fund diversion from the Ether Reapers contract to its deployer address be identified?*
   This question focuses on recognizing transaction patterns that deviate from legitimate NFT project behavior, which could indicate a rug pull. The system should return the RDF triples that describe that NFT minters transferred Ether to the NFT contract via the MintReaper function and, likewise, the amount of funds that was transferred from the Ether Reapers contract to the deployer account 0xc8a6.

3. **Counting questions**

CQ6 *How many NFT token accounts did Homer_eth deploy on Ethereum?*
It is expected that the number 6 is returned. This question tests if the system is able to distinguish among the various types of blockchain accounts, such as externally owned, token, and non-token accounts.

CQ7 *How many Ethereum addresses minted an Ether Reapers NFT?*
It is expected that the number 379 is returned. This question solidifies if the system can extract and identify the purpose of blockchain transactions — in this case, if the purpose of the transaction is to mint an NFT.

### 5.4.2 Functional Requirements

In the following, the functional requirements for Kosmosis are outlined.

FR1 **Blockchain Data Ingestion**
The system shall continuously ingest all mined transactions as JSON-formatted data from the Bitcoin, Ethereum, and Polygon blockchains. Each transaction will include, at a minimum, the transaction ID, sender and receiver addresses, transaction amount, and timestamp. To keep the KG up to date with the latest blockchain state, the system shall listen for new transactions from blockchain archive nodes via websocket connections.

FR2 **𝕏 Posts Ingestion**
The system will authenticate with the 𝕏 Filtered stream API using bearer token authentication to fetch posts based on predefined rules (e. g., “bc1” to look for posts with Bitcoin addresses). The posts should be fetched as JSON-formatted data. Required data fields include post ID, timestamp, content, and author ID.

FR3 **Enrichment Data Ingestion**
The system integrates enrichment data from the Golden KG using its RESTful API to provide additional information for real-world entities. Ingestion shall be triggered every time a new real-world entity is added to the KG. Enrichment data should be requested using the 𝕏 user ID, if available.

FR4 **Blockchain Address Relation Extraction**
The system shall extract the relationship between addresses on a per-transaction basis. This includes the extraction of semantics in blockchain transactions (i. e., decoding the input data) to determine the type of relationship. Relationship types may include sender$\leftrightarrow$receiver (transfer), token contract$\leftrightarrow$deployer (contract creation), and NFT contract$\leftrightarrow$NFT minter (mint).

FR5 **Real-World Entity Canonicalization**
The system shall identify real-world entities across blockchain transactions and social media data, acknowledging that a real-world entity can have different identifiers in these sources (e. g., blockchain addresses, social media profile IDs). A real-world entity is considered canonicalized if it exists only once with its name in the KG and has an associated 𝕏 user ID and blockchain address.

FR6 **Incremental Updates**
As new data becomes available, the pipeline must extract relevant knowledge and integrate it into the existing KG without a full rebuild. This includes adding new RDF triples and updating existing ones if necessary.

FR7 **RDF Mapping**
Following data processing, the system will map the processed (un-, semi-, or structured) data to the RDF format in accordance with a defined ontology that is applied to the KG.

FR8 **SPARQL API**
The KG is stored in a triplestore that provides a SPARQL API to serve two primary purposes: enabling the pipeline to load results into the triplestore and providing an interface for external applications to query and access the constructed KG.

### 5.4.3 Nonfunctional Requirements

In addition to the functional features that Kosmosis should offer, there are also several nonfunctional requirements about how these features should be implemented.

NR1 **Scalability**
A scalable system ensures the KG construction pipeline can handle future growth and evolving data demands as either (i) the volume of blockchain transactions and/or (ii) the number of data sources increases. Scalability should be achievable through modular components that can be added as needed.

NR2 **Data Quality**
Ensuring data quality is crucial for the reliability and trustworthiness of the resulting KG. The system must ensure that the KG entities are free from errors and inaccuracies and do not exhibit any contradictions or disparities within themselves or when compared to other sources.

# Chapter 6

# Kosmosis Approach

This chapter presents the Kosmosis approach and its core components. Section 6.1 establishes the foundation for understanding the design and functionality of Kosmosis by providing an architectural overview. Subsequently, section 6.2 discusses the applied ontology, a crucial element for structuring and interpreting data within the system. The following sections describe the KG construction pipeline (section 6.3) and data-processing workflows, including blockchain data processing in section 6.4, natural language text processing in section 6.5, and enrichment data processing in section 6.6. Each of these workflows contributes to extracting knowledge from raw data.

## 6.1 Architectural Overview of Kosmosis

To incrementally construct a KG that integrates data in a continuous and periodic way, a multistage pipeline is utilized, as proposed in [117]. This pipeline is composed of three stages: data ingestion, data processing, and knowledge storage. These stages are depicted in figure 6.1, illustrating the sequential flow from data ingestion to the final storage of structured knowledge.

The initial stage, data ingestion, captures the raw data from the primary data sources as well as the enrichment data sources. This phase is characterized by its versatility in the frequency of data acquisition as follows:

- **Continuous**, to capture recently mined transactions from blockchain nodes via websocket connections.
- **Incremental**, to fetch new social media posts from the social media platform 𝕏 via the 𝕏 Filtered stream API endpoint.
- **Periodic**, to pull new entries from structured data sources like relational databases at regular intervals.
- **Event-based**, to capture data in response to emitted events, such as fetching supplementary information from an external knowledge base when a new entity is added to the KG.

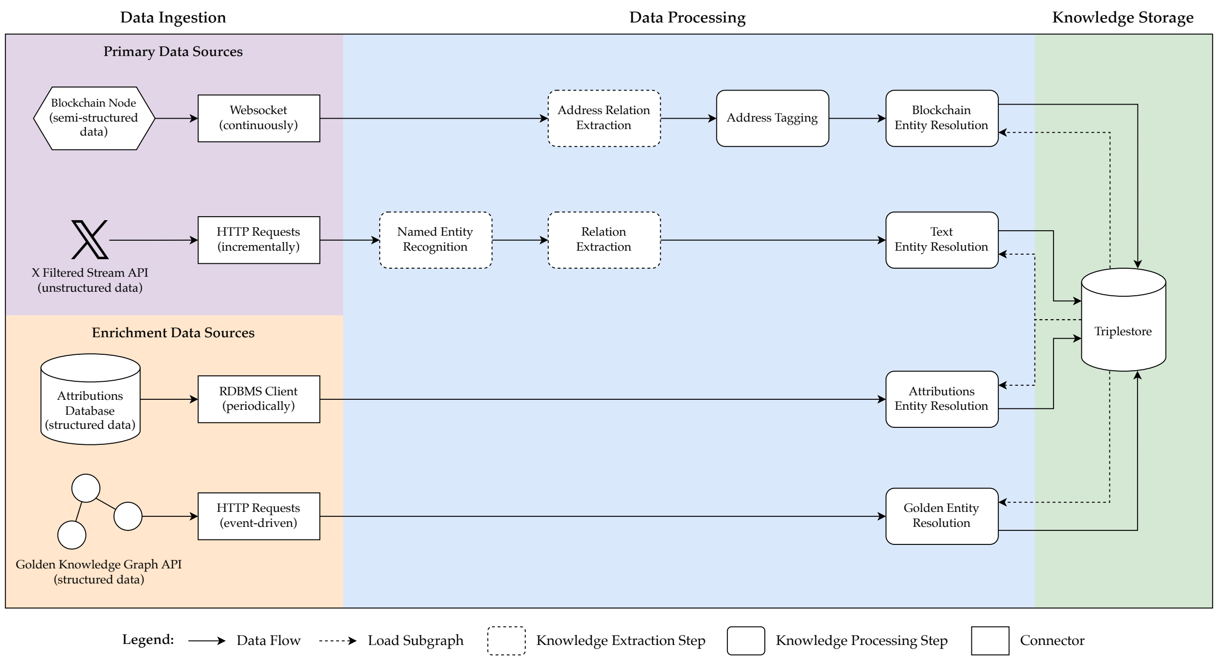

Data Ingestion
Data Processing
Knowledge Storage
Primary Data Sources
Blockchain Node (semi-structured data)
Websocket (continuously)
Address Relation Extraction
Address Tagging
Blockchain Entity Resolution
X Filtered Stream API (unstructured data)
HTTP Requests (incrementally)
Named Entity Recognition
Relation Extraction
Text Entity Resolution
Triplestore
Enrichment Data Sources
Attributions Database (structured data)
RDBMS Client (periodically)
Attributions Entity Resolution
Golden Knowledge Graph API (structured data)
HTTP Requests (event-driven)
Golden Entity Resolution
Legend:
Data Flow
Load Subgraph
Knowledge Extraction Step
Knowledge Processing Step
Connector

**Figure 6.1:** Overview of Kosmosis, the incremental KG construction pipeline

Following data ingestion, knowledge is extracted from the data and processed in the data processing stage. This stage comprises four distinct data-processing workflows to processes the ingested data depending on its data source. A data-processing workflow can have knowledge-extraction steps (e. g., relation extraction), if required, and knowledge-processing steps (e. g., blockchain entity resolution).

In the third and final stage, the refined data is loaded into the knowledge storage, where it is systematically organized within a triplestore. Production-ready triplestores are for example *Apache Jena TDB* [118], *Ontotext GraphDB* [119], and *OpenLink Virtuoso* [120]. For the persistence of the KG, *Ontotext GraphDB* was the database of choice, motivated by prior experience with it in [121]. The triplestore can then be used for semantic querying capabilities to extract actionable insights from the KG for downstream tasks.

## 6.2 Knowledge Graph Ontology

This section introduces the ontology specifically designed to formalize the concepts and relations within the domain of crypto assets, with a particular focus on fraud investigations. The design of the ontology is aligned with the CQs delineated in section 5.4, which guided the development of the ontology.

The KG ontology is written with RDF, RDFS, and OWL and comprises four concepts: blockchain account, blockchain transaction, social media account, and real-world entity. Each class can have datatype properties owl:Class ←owl:dataTypeProperty— Datatype or object properties owl:Class —owl:objectProperty→ owl:Class and can be a sub class of another class owl:Class —rdfs:subClassOf→ owl:Class in the ontology. Furthermore, enumerated classes specifically define a set of permissible RDF resources (i. e., a list of Instance values).

### 6.2.1 Blockchain Account Concept

The blockchain account concept (depicted in figure 6.2) models blockchain addresses and their characteristics as accounts. A blockchain account is categorized as either an EOA, contract account, or UTXO account.

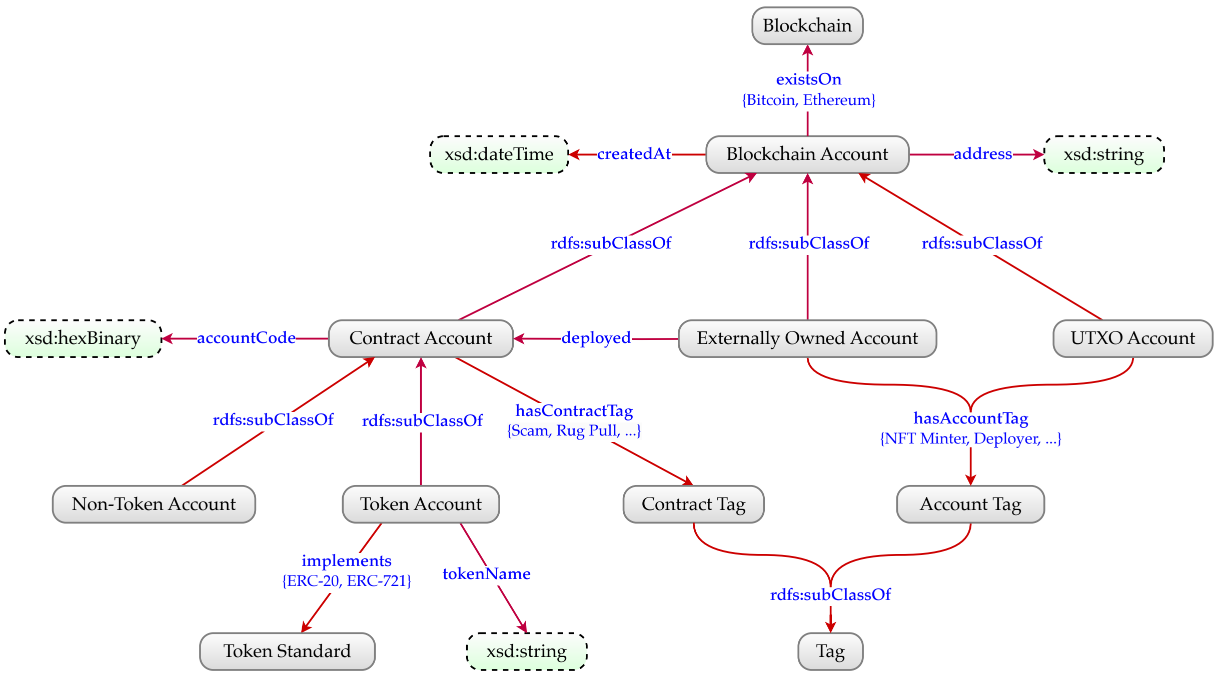


**Figure 6.2:** Illustration of the blockchain account concept

- **Blockchain**: A distributed database that maintains a continuously-growing list of records, called blocks, secured from tampering and revision.
- **Blockchain Account**: This is the main entity of the account concept. An account can represent any blockchain user who can own cryptocurrencies or tokens, and send transactions on the network. An account can be uniquely identified by its address and the blockchain it exists on. Ethereum addresses have a fixed format and encoding as hexadecimal numbers xsd:hexBinary. This is not the case for Bitcoin addresses that have variable format and encoding (Base58Check or Bech32) addresses [63]. As a result, addresses are modeled with the more generic datatype xsd:string and application-level validation logic outside the triplestore to allow for greater flexibility when integrating other blockchains with different format (e. g., Solana uses a 32-byte array, encoded with the Base58 alphabet [45]).
- **EOA**: An Externally Owned Account —rdfs:subClassOf→ Blockchain Account is controlled by a private key and is typically associated with a real-world entity. This could be a person or an organization that controls an externally owned account. They can send transactions and deploy smart contracts.
- **Contract Account**: A Contract Account —rdfs:subClassOf→ Blockchain Account is a smart contract on the blockchain. A contract account has a non-empty associated EVM code to perform operations on the blockchain when transactions are sent to them.

- **Tag**: Tags help to identify the owner of a blockchain address or the type of real-world entity it represents. Tags can be used to mark addresses associated with fraudulent activity, such as a Scam or Rug Pull. Instead of defining different subclasses for accounts, associating tags with accounts has shown to be a more flexible approach during the design cycle of Kosmosis.
- **Account Tag**: An Account Tag —rdfs:subClassOf→ Tag holds additional information associated with an externally owned account or UTXO account. For instance, an account tag can be Exchange Deposit, or Deployer.
- **Contract Tag**: Similar to an account tag, a Contract Tag —rdfs:subClassOf→ Tag is additional information associated with a contract account. However, the set of contract tags is disjoint with the set of account tags. For instance, a contract can be tagged as a Scam or Rug Pull.
- **Account Code**: The executable EVM bytecode of a smart contract in xsd:hexBinary format that is executed when an externally owned account triggers the execution via a call transaction.
- **Non-Token Account**: A Non-Token Account —rdfs:subClassOf→ Contract Account handles operations other than managing tokens. For instance, a voting contract or custom smart contract.
- **Token Account**: A Token Account —rdfs:subClassOf→ Contract Account manages the logic and ownership information of a particular token. This includes managing tokens, such as *USD Coin* [47], as well as NFTs, for example, *CryptoPunks* [48].
- **Token Standard**: The ERC-20 [66] and ERC-721 [67] token standards define a set of rules for creating fungible or non-fungible tokens. Both token standards are supported on EVM-compatible blockchains (e. g., Ethereum, Polygon). The token standards are used to further differentiate between the type of token a smart contract manages.

### 6.2.2 Blockchain Transaction Concept

The blockchain transaction concept (figure 6.3) covers the mechanisms through which crypto assets are transferred and managed on blockchains. This concept encompasses a variety of transaction types, including UTXO and account-based transactions, each with distinct processes for handling asset transfers.

- **Transaction**: Transactions are messages between two accounts that may transfer Ether and may contain a payload. Transactions always originate from an external account that is controlled by an external actor by means of a private key. Instances of the transaction class can have the following properties: `value`, the amount of a crypto asset being transferred; `minedInBlock`, the block height at which the transaction was added to the blockchain; `minedOn`, describing on which blockchain the transaction was minded; and `action`, the specific action being taken, such as transferring tokens or minting an NFT.

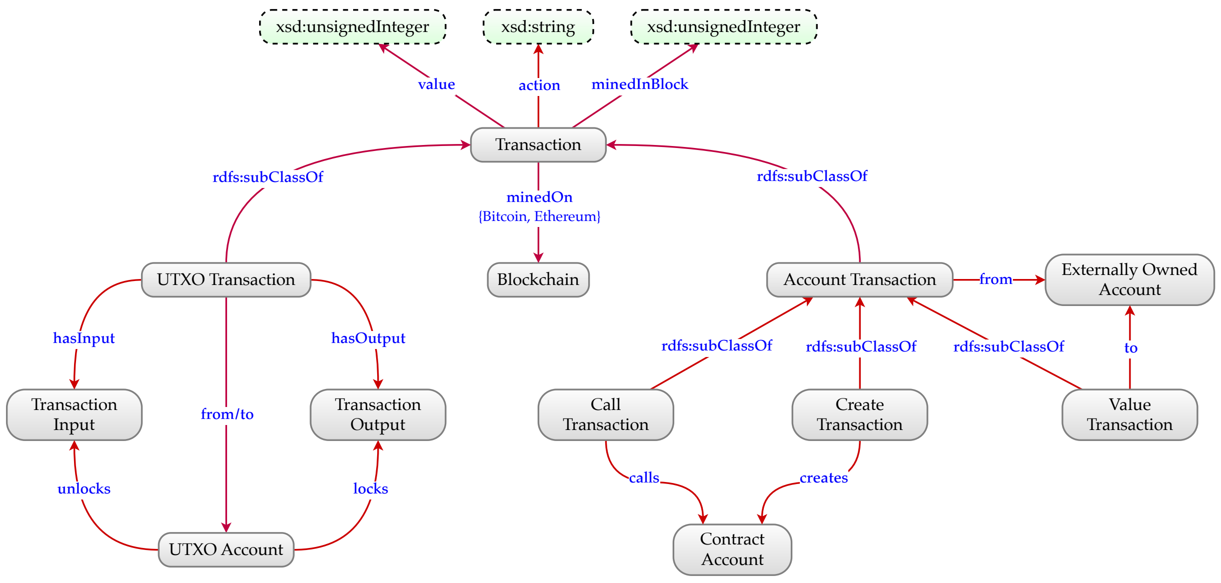


**Figure 6.3:** Illustration of the blockchain transaction concept

For UTXO-based blockchains:

- **UTXO Transaction**: When a UTXO Transaction —rdfs:subClassOf→ Transaction is made, the output of a previous transaction is referenced as the input for the new transaction, showing the movement of currency.
- **Transaction Output**: They are created by the sender and represent the destination of a transaction. A single transaction can have multiple outputs, allowing a sender to have multiple recipients in one transaction. Once a transaction output is created, it remains unspent until it is used in a new transaction. A UTXO account adds a claiming condition (ScriptPubKey) to the transaction output that `locks` the output to the recipient.
- **Transaction Input**: The transaction input references previous UTXOs that are being being spent in a transaction. Each UTXO represents the remaining funds after a transaction and is associated with a UTXO account. In order to spend a UTXO in a transaction, a UTXO account `unlocks` the transaction input by submitting a valid ScriptSig.

For account-based blockchains:

- **Account Transaction**: An Account Transaction —rdfs:subClassOf→ Transaction represents the transfer of tokens between accounts which have balances, as opposed to UTXO where previously unspent outputs are spent.
- **Value Transaction**: A Value Transaction —rdfs:subClassOf→ Account Transaction does not call a function in a smart contract and does not create a new smart contract. Even though it is called value transaction, transactions with a value of 0 can be a value transaction. They can have associated input data (e. g., a text message) as long as this input data does not trigger the execution of smart contract code.

- **Call Transaction**: A Call Transaction —rdfs:subClassOf→ Account Transaction is a type of transaction in which an externally owned account interacts with a contract account by calling a method in the code of its underlying contract.
- **Create Transaction**: A Create Transaction —rdfs:subClassOf→ Account Transaction creates a new smart contract on the blockchain.

### 6.2.3 Social Media Account Concept

The social media concept, depicted in figure 6.4, provides a structured representation of how data is related within a social media context, particularly one that involves interactions with blockchain technology.

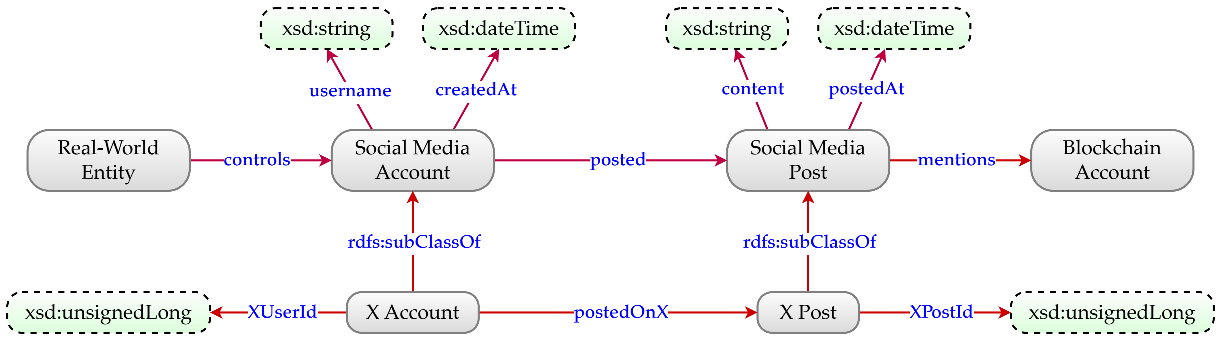


**Figure 6.4:** Illustration of the social media account concept

- **Social Media Account**: This represent a social media profile controlled by a real-world entity, such as an individual or an organization. Regardless of the platform, a social media account is characterized by two attributes: it has a creation date and time (`createdAt`), in xsd:dateTime format, and it is identified by a `username`, which is an alphanumeric xsd:string.
- **X Account**: An X Account —rdfs:subClassOf→ Social Media Account represents a social media account on the social media platform 𝕏. This could be an account that belongs to a public figure or an organization that is associated with the crypto asset sector. An 𝕏 account can be uniquely identified by the `XUserId` of type xsd:unsignedLong.
- **Social Media Post**: A social media post represents individual posts on social media platforms. Social media accounts can create and share either a single post or a series of posts. Each post is characterized by its content, formatted as xsd:string in natural language, and the time it was published (`postedAt` in xsd:dateTime format). In case a social media post mentions the address of a blockchain account, that can be an externally owned, contract, or UTXO account.
- **X Post**: An X Post —rdfs:subClassOf→ Social Media Post represents a post, such as an announcement, can be uniquely identified by its `XPostId`, which is of the same data type as `XUserId` xsd:unsignedLong.

### 6.2.4 Real-World Entity Concept

The real-world entity concept, as illustrated in figure 6.5, describes and relates real-world entities to blockchain and social media accounts.

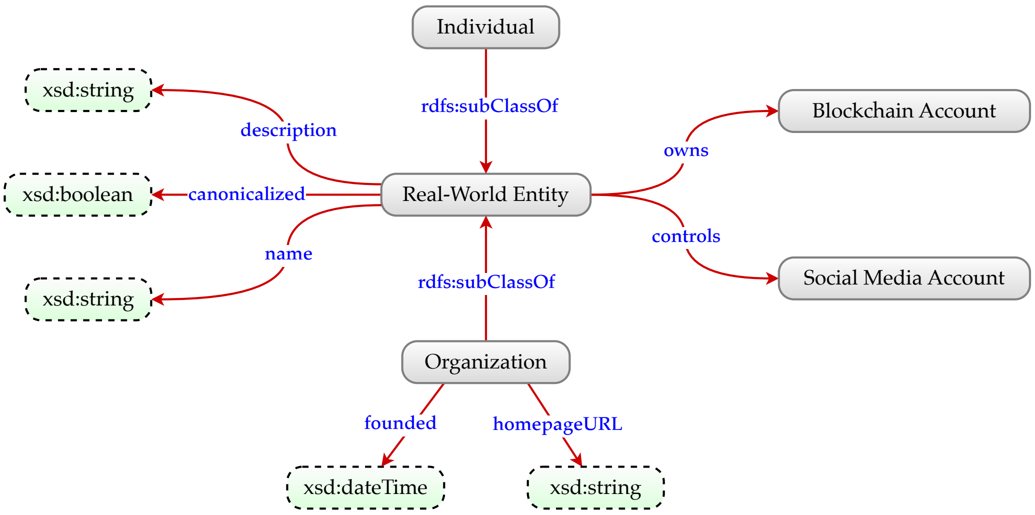


**Figure 6.5:** Illustration of the real-world entity concept

- **Real-World Entity**: An entity in the real world, such as an `Individual` or an `Organization`. It can own one or multiple blockchain accounts and control one or multiple social media accounts. A real-world entity has a name of data type xsd:string. Further, it can have a description that provides additional information about the entity. A real-world entity may not be *canonicalized* at first. For instance, if the same real-world entity exists multiple times with different names in the KG or if two blockchain addresses are considered to belong to the same real-world entity, but there is not enough evidence to assign a name.
- **Individual**: An Individual —rdfs:subClassOf→ Real-World Entity is a single person in the real world. An individual inherits the name and description attribute from its superclass the Real-World Entity.
- **Organization**: The Organization —rdfs:subClassOf→ Real-World Entity class represents structured groups of people (i. e., organizations). An organization has a founding date, which is in xsd:dateTime format, and typically has an associated homepage URL in xsd:string format.

## 6.3 The Knowledge Graph Construction Pipeline

The KG construction pipeline consists of four main classes (visualized in figure 6.6): `Pipeline`, `DataProcessingWorkflow`, `ProcessingStep`, and `IngestionStrategy`. The `Pipeline` class serves as the central orchestrator for the data processing workflows and implements the singleton pattern to ensure that there is only a single pipeline instance. To manage the life cycle of data processing workflows it exposes four methods:

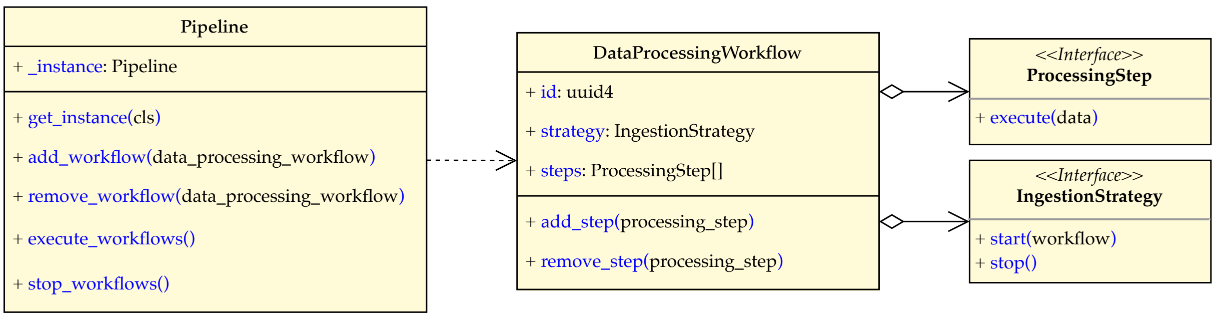


**Figure 6.6:** UML class diagram for the pipeline

- `add_workflow` adds a `DataProcessingWorkflow` to the pipeline.
- `remove_workflow` removes a `DataProcessingWorkflow` from the pipeline.
- `execute_workflows` starts the execution of all defined processing workflows.
- `stop_workflows` shuts down running processing workflows gracefully.

The `DataProcessingWorkflow` class acts as the context in the strategy pattern and represents the individual workflows that are managed by the `Pipeline`. It allows different `ProcessingStep` implementations (strategies) to be added or removed and executed in sequence. Each `DataProcessingWorkflow` can be identified by a universally unique identifier (UUID) and has the following two methods:

- `add_step` adds a `ProcessingStep` to the processing workflow.
- `remove_step` removes a `ProcessingStep` from the processing workflow.

The `ProcessingStep` interface defines a common interface for executing a step, and each concrete processing step class implements its specific behavior. For ingesting data, the `IngestionStrategy` defines a common set of methods to ensure that all ingestion strategies follow a uniform approach to starting and stopping data ingestion.

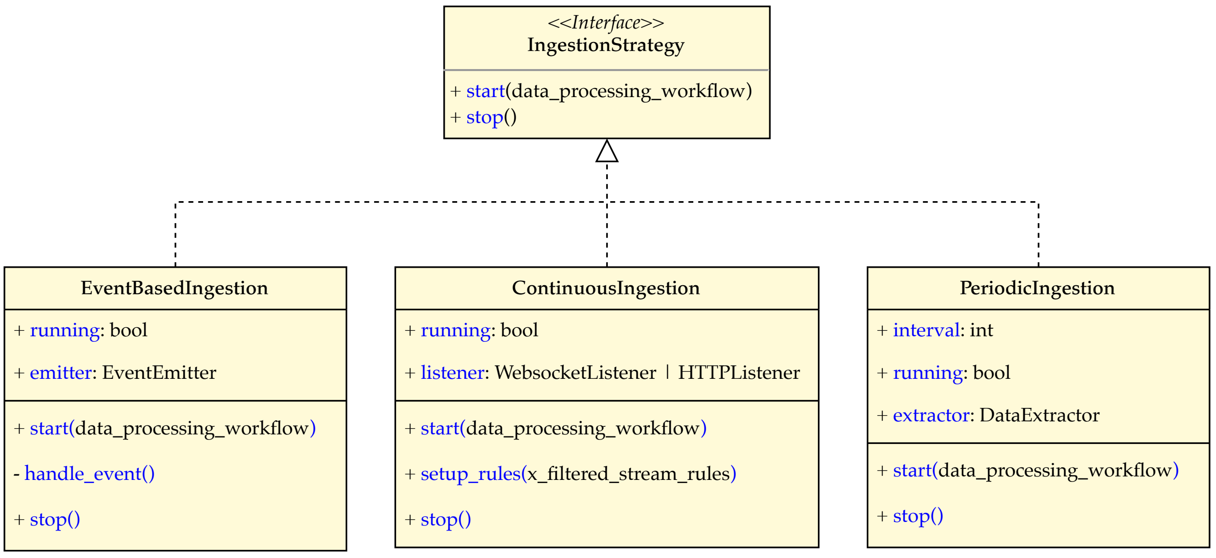

<<Interface>>
IngestionStrategy
+ start(data_processing_workflow)
+ stop()
EventBasedIngestion
+ running: bool
+ emitter: EventEmitter
+ start(data_processing_workflow)
- handle_event()
+ stop()
ContinuousIngestion
+ running: bool
+ listener: WebsocketListener | HTTPListener
+ start(data_processing_workflow)
+ setup_rules(x_filtered_stream_rules)
+ stop()
PeriodicIngestion
+ interval: int
+ running: bool
+ extractor: DataExtractor
+ start(data_processing_workflow)
+ stop()

**Figure 6.7:** UML class diagram of the concrete ingestion strategies

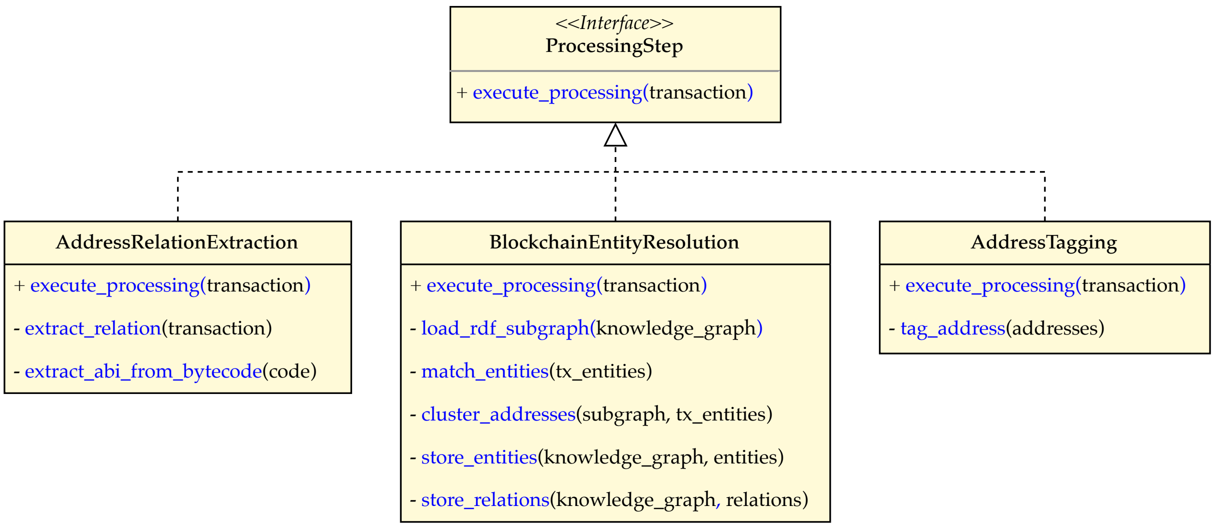
Figure 6.7 depicts the different ingestion strategies the KG construction pipeline uses. The `IngestionStrategy` interface mandates the implementation of two methods: `start` to initiate ingestion with a specified data-processing workflow and `stop` to terminate the process. The `EventBasedIngestion` class has an `EventEmitter` for event-triggered ingestion. It adds a private `handle_event` method for internal event handling. For the `ContinuousIngestion` class, its listener is polymorphic, capable of being a `WebsocketListener` or `HTTPListener`, which facilitates continuous data streaming via websockets as well as incremental ingestion through long-lived HTTP requests. Lastly, the `PeriodicIngestion` class is characterized by an interval attribute for scheduling data extraction events and a `DataExtractor` component to execute the data retrieval.

The client implements a `get_processing_workflow_factory` function that returns an instance of a factory class based on the `ingestion_type` as follows:

- INCREMENTAL — to create a workflow specifically for the 𝕏 Filtered stream
- EVENT — to create a workflow that is triggered by events
- STREAM — to create a workflow that continuously ingests data in real time
- SCHEDULED — to create a workflow that ingests data at predefined intervals

## 6.4 Blockchain Data-Processing Workflow

The blockchain data-processing workflow continuously ingests new transactions from the blockchain via websocket connections. Websockets enable open and interactive communication sessions between a client and server, facilitating real-time data transfer without the need for repeated polling. Figure 6.8 depicts the classes responsible for the blockchain data processing. Each class represents a processing step and is described in the following sections.



<<Interface>>
ProcessingStep
+ execute_processing(transaction)
AddressRelationExtraction
+ execute_processing(transaction)
- extract_relation(transaction)
- extract_abi_from_bytecode(code)
BlockchainEntityResolution
+ execute_processing(transaction)
- load_rdf_subgraph(knowledge_graph)
- match_entities(tx_entities)
- cluster_addresses(subgraph, tx_entities)
- store_entities(knowledge_graph, entities)
- store_relations(knowledge_graph, relations)
AddressTagging
+ execute_processing(transaction)
- tag_address(addresses)

**Figure 6.8:** UML class diagram of the blockchain data-processing workflow

### 6.4.1 Address Relation Extraction

The *address relation extraction* module is designed to decode a blockchain transaction input data by using the ABI of the smart contract. This enables the module to extract semantic information from transactions where an EOA interacts with a smart contract, including the purpose of a transaction derived from the smart contract function that is called, and context in form of arguments that are passed to this function (e. g., the number of NFTs to be minted in a transaction). Consider the following raw transaction in listing 6.1 that was obtained from Ethereum for the Homer_eth rug pull use case. Herein, the address 0xB24 transfers 0.08 ETH to 0x0a5, a token contract named Ether Reapers (retrieved through calling the `name` function of the contract). The rest of this interaction is encoded in the input data, which requires the ABI for decoding.

```json
{
  "method": "eth_get_transaction",
  "params": {
    "result": {
      "transaction": {
        "blockHash": "0x3d3...577",
        "blockNumber": 13456446,
        "from": "0xB24",
        "gas": 255099,
        "gasPrice": 110054388777,
        "maxFeePerGas": 188538774740,
        "maxPriorityFeePerGas": 2500000000,
        "hash": "0x819",
        "input": "0xa844...001",
        "nonce": 57,
        "to": "0x0a5b9b930fc5bE638232D8b9b69Cb5B46249c06e",
        "transactionIndex": 208,
        "value": 80000000000000000,
        "type": 2,
        "accessList": [],
        "chainId": 1,
      }
    }
  }
}
```

**Listing 6.1:** Transaction on the Ethereum blockchain

First, the ABI is requested from *Etherscan* [21] and *Sourcify* [122] via their respective RESTful APIs. If the ABI cannot be successfully fetched from one of the aforementioned sources, the module resorts to reconstructing the ABI from the smart contract bytecode, which is available at any time since the bytecode is deployed on the blockchain. This reconstruction, outlined in algorithm 1, enables the decoding of transactions and the interaction with smart contracts beyond their compiled state.

**Algorithm 1** Extract ABI from Bytecode

```
1: procedure ABIFROMBYTECODE(bytecode)
2:     $p \leftarrow$ DISASM$(bytecode)$, $abi \leftarrow$ empty array
3:     for each $(selector, offset)$ in $p.selectors$ do
4:         if $offset$ not in $p.destinations$ then
5:             continue
6:         end if
7:         $fn \leftarrow p.destinations[offset]$
8:         $tags \leftarrow$ FUNCTIONTAGS$(fn, p.destinations)$
9:         $abiFunction \leftarrow$ new AbiFunction with $payable = \neg p.notPayable[offset]$
10:        $mutability \leftarrow "nonpayable"$
11:        if $abiFunction.payable$ then
12:            $mutability \leftarrow "payable"$
13:        else
14:            $hasStateChangingOps \leftarrow$ false
15:            $hasStateReadingOps \leftarrow$ false
16:            for $opcode$ in [SSTORE, CREATE, CREATE2] do
17:                if $opcode$ in $tags$ then
18:                    $hasStateChangingOps \leftarrow$ true
19:                    break
20:                end if
21:            end for
22:            if not $hasStateChangingOps$ then
23:                for $opcode$ in [SLOAD] do
24:                    if $opcode$ in $tags$ then
25:                        $hasStateReadingOps \leftarrow$ true
26:                        break
27:                    end if
28:                end for
29:                if $hasStateReadingOps$ then
30:                    $mutability \leftarrow$ "view"
31:                else
32:                    $mutability \leftarrow$ "pure"
33:                end if
34:            end if
35:        end if
36:        $abiFunction.stateMutability \leftarrow mutability$
37:        if $tags$ has $RETURN$ or $mutability = "view"$ then
38:            add output to $abiFunction.outputs$
39:        end if
40:        if $tags$ has $CALLDATA[LOAD/SIZE/COPY]$ then
41:            add input to $abiFunction.inputs$
42:        end if
43:        add $abiFunction$ to $abi$
44:    end for
45:    return $abi$
46: end procedure
```

The initial step involves the disassembly of the bytecode of the smart contract. This process, referred to as `DISASM`, decomposes the bytecode into a series of readable opcodes and associated data. Disassemblers (e. g., *pyevmasm* [123]) facilitate this step by translating the bytecode back into a form that represents the original instructions and operations defined within the smart contract.

Following disassembly, the algorithm initializes by creating an empty array intended to store the ABI and defining lists of opcodes that either change the state or read from the state of the blockchain. These opcodes include `SSTORE`, `CREATE`, `CREATE2` for state-changing operations and `SLOAD` for state-reading operations, reflecting the fundamental actions a smart contract on the EVM can perform [44].

The core of the algorithm iterates over selector/offset pairs within the disassembled bytecode. Selectors serve as identifiers for functions in the EVM, facilitating the mapping to the corresponding functionality. If a given offset does not match any destination within the program's destinations, the iteration skips to the next pair, ensuring only valid functions are considered. Upon finding a valid function destination, the algorithm retrieves the function definition and assigns tags based on its behavior. This tagging process involves analyzing the opcodes contained within the function and any related jump destinations. The purpose is to categorize functions according to how they alter the blockchain state, using a depth-first search algorithm to navigate through the function call graph.

An `AbiFunction` object is then created for each valid function, with its payable status determined inversely by the presence of a `notPayable` marker at the corresponding offset. The algorithm next assigns mutability attributes (`nonpayable`, `payable`, `view`, or `pure`) based on whether the function alters state, reads state, or neither. This classification is crucial for understanding how functions interact with the blockchain and their implications on transaction costs and permissions.

Finally, the algorithm decides on the inclusion of inputs and outputs in the function signature, informed by the presence of specific tags. For instance, tags indicating data retrieval or state mutation influence whether parameters are inputs or outputs.

Returning to the transaction example in Listing 6.1, with the ABI of the smart contract, it is now possible to gain additional context about the transaction by decoding:

- the input data of the transaction: `mintReaper(uint256 _times)`, which includes both the name of the function being called `mintReaper` and the name of the argument `_times` with its data type `uint256`.
- the input argument(s) that were passed as input data to the function `mintReaper` to determine the number of NFTs to be minted in a single transaction. In this instance, the quantity is determined by the `_times` argument, which is set to 1.

Combined with the previously known information that 0.08 ETH[1] were transferred and the recipient 0x0a5 is an NFT contract, dubbed Ether Reapers, this allows to describe the action of this transaction as "0xB24 minted 1 Ether Reapers NFT for 0.08 ETH."

[1] Transaction value is denoted in wei, the smallest denomination of ETH. 1 ETH = $1 \cdot 10^{18}$ wei.

### 6.4.2 Address Tagging

Although the exact identity of a real-world entity controlling a blockchain address is often times unknown, it can still be categorized and tagged based on prior blockchain activities. The *address tagging module* tags the sender and receiver addresses based on their extracted relationship from the preceding address relation extraction module. For instance, an EOA deploying a smart contract is tagged as deployer in case of a contract creation transaction. Likewise, if an EOA is sending Ether to an NFT contract $T$ via a contract function containing the word "mint," the EOA is tagged as NFT minter of $T$.

Listing 6.2 shows the processed transaction after the address relation extraction and address tagging steps. The transaction object has three additional properties: `action` describes the purpose of the transaction, `decodedInput` is the decoded input data, and `inputArgs` are the input arguments that are passed to a function call in the input data. Properties that are not stored in the KG (e. g., `gasPrice`) are removed from the transaction object. The processed transaction has two additional entries for sender and receiver, each with a contract, classes, and tags property. The `classes` property indicates the class according to the ontology, and `tags` are the address tags. In case of a smart contract, a `name` property for the contract name is added.

```
{
  "blockchain": "Ethereum",
  "transaction": {
        "action": "Mint NFT",
        "blockNumber": 13456446,
        "hash": "0x819",
        "sender": "0xB24",
        "recipient": "0x0a5",
        "nonce": 57,
        "value": 80000000000000000,
        "decodedInput": "mintReaper(uint256 _times)"
        "inputsArgs": [{ "_times": 1 }]
  },
  "sender": {
        "contract": false,
        "classes": ["Externally Owned Account"],
        "tags": ["Ether Reapers Minter"],
  },
  "recipient": {
        "contract": true,
        "name": "Ether Reapers",
        "classes": ["Token Contract", "ERC721"],
        "tags": []
  }
}
```

**Listing 6.2:** Processed transaction after address relation extraction and tagging

### 6.4.3 Blockchain Entity Resolution

The blockchain entity resolution step is the final module in the blockchain data-processing workflow. This module is responsible for identifying unique entities represented in the blockchain data. This involves distinguishing among different types of entities (e. g., blockchain accounts, transactions) and ensuring that each entity is uniquely identified within the KG. Further, it is responsible for linking blockchain addresses to either new or existing real-world entities in the KG based on their interactions on the blockchain or by mentions on social media.

Entity resolution begins with a request to the KG to receive RDF triples that are relevant to the resolution process based on the processed blockchain transaction, including:

- Blockchain account triples based on the blockchain addresses in the transaction.
  - Account tags Externally Owned Account $\xrightarrow{\text{hasAccountTag}}$ Account Tag based on past transaction behavior
  - Contract tags Contract Account $\xrightarrow{\text{hasContractTag}}$ Contract Tag for smart contracts
  - Real-World Entity $\xrightarrow{\text{owns}}$ BlockchainAccount to account for blockchain accounts that are already linked to a real-world entity
  - Externally Owned Account $\xrightarrow{\text{deployed}}$ Contract Account to determine if an EOA is a contract deployer
- Transaction triples for previous transactions made by the sender of the transaction. The transaction nonce can be used to determine how many transactions the address already made. The amount of transactions is limited to 100 to ensure the subgraph does not become too large.
  - Prior Value Transaction $\xrightarrow{\text{from}}$ Externally Owned Account of the sender of an account-based transaction to consider past interactions with other EOAs
  - Prior Call Transaction $\xrightarrow{\text{calls}}$ Contract Account by the sender of an account-based transaction to consider past smart contract interactions
  - Prior UTXO Transaction $\xrightarrow{\text{to}}$ UTXO Account spent by the sender of a UTXO
  - Amount/value Transaction $\xrightarrow{\text{value}}$ xsd:unsignedInteger of each transaction
  - Timestamp Transaction $\xrightarrow{\text{minedInBlock}}$ xsd:unsignedInteger of each transaction
- Social media triples provide additional information that can aid in linking blockchain accounts to real-world entities.
  - X Post $\xrightarrow{\text{mentions}}$ Blockchain Account for 𝕏 mentions of a blockchain address
  - X Account $\xrightarrow{\text{postedOnX}}$ X Post to get the author of the 𝕏 post mentioning the address of the blockchain account
  - Content X Post $\xrightarrow{\text{content}}$ xsd:string of each 𝕏 post
  - Timestamp X Post $\xrightarrow{\text{postedAt}}$ xsd:dateTime of each 𝕏 post

To identify potential duplicates, the system matches the entities from the RDF triples with the entities contained in the transaction object (e. g., listing 6.2), which is currently being processed. Once the entities are matched and potential duplicates are removed, the system continues with linking the blockchain addresses, that are involved in the transaction, to real-world entities. The linking is done in the following steps:

1. *Canonicalized real-world entities*: First, the system checks if the sender or recipient address, or both, is linked to a canonicalized real-world entity, identified through RDF triples that confirm the real-world entity owns the blockchain account.
2. *Social media-based linking*: If either none or one of the addresses is not yet linked to a real-world entity, the system checks if there are any social media mentions of that blockchain addresses that can be used to link the blockchain address to a real-world entity.
3. *Transaction-based linking*: If still none or one of the addresses is not yet linked to a real-world entity, clustering algorithms based on the heuristics outlined in section 4.2.1, are applied to determine which blockchain addresses in a transaction may belong to the same real-world entity. The clustering algorithms are selected based on which blockchain the transaction originated from. In the following, the implementation of transaction-based linking is described in detail.

**Account-based Transactions**

In a first step, it is important to distinguish between addresses that belong to centralized exchanges or individuals, since exchanges addresses can either be smart contract addresses or EOAs [99]. To achieve this, the deposit address reuse heuristic is utilized.

*Deposit Address Reuse*

Firstly, the algorithm takes as its input a graph where nodes represent addresses and edges represent transactions between them. The graph is examined to isolate subsets of exchange addresses and miner addresses, along with the parameter $a_{max}$, the maximum difference in Ether amount allowable for two linked transactions to be considered part of a deposit action. The algorithm begins by iterating through transaction paths that link two exchange addresses. The path is valid for the heuristic if it starts with an address that is not identified as an exchange or a miner and ends with an exchange address. For each path found, the algorithm checks if the transactions involved are of the same type (either both Ether transactions or both token transfers) and if the differences in the amount between the received and forwarded transactions fall within the specified maximums. When a path meets these criteria, the deposit address is recorded. This address is the intermediary through which Ether or tokens are received from a user address and forwarded to an exchange address. The algorithm builds a separate graph for exchange-related activities and another for user activities, adding the valid paths to the respective graphs. Finally, it refines the user address clusters by removing any addresses that have already been identified as belonging to exchanges. The result is two mappings: $M_e$, which associates addresses with exchanges, and $M_u$, which associates addresses with individual users.

*EVM Address Coexistence*

Recall from the use case in chapter 5 that the address 0xc3bc coexists as Ethereum EOA and Polygon EOA. It is reasonable to assume that Homer_eth owns both EOAs since a single address is invariably linked to a unique private key across all EVM-compatible blockchains. To the best of my knowledge, this heuristic is unexplored in academic studies. It capitalizes on the fact that EVM-compatible blockchains are designed to be interoperable with Ethereum, meaning they use the same algorithms to generate addresses. This compatibility is achieved in part through the use of the same cryptographic algorithms for generating private keys and addresses. In Ethereum and EVM-compatible blockchains, addresses are derived from the public key, which in turn is derived from the private key using the ECDSA on the secp256k1 curve [44]. The process for generating an address involves taking the Keccak-256 hash of the public key and then using the last 20 bytes of this hash to form the address [44]. Because this process is standardized across EVM-compatible chains, the same private key will generate the same public key and, consequently, the same address on all these blockchains. As a result, the control of an address on one chain implicitly confirms control over the same address on all other EVM-compatible chains, provided the real-world entity holds the corresponding private key. Wallet software like MetaMask allows users to switch between different EVM-compatible networks. When a user switches networks, the same account (i. e., the same set of private and public keys and thus the same address) is used to interact with the new network. Transactions and balances, however, are network-specific. For instance, an ETH balance will not carry over to Polygon, but the address through which a user accesses and manages assets on Polygon will be the same as on Ethereum, as shown in figure 6.9.

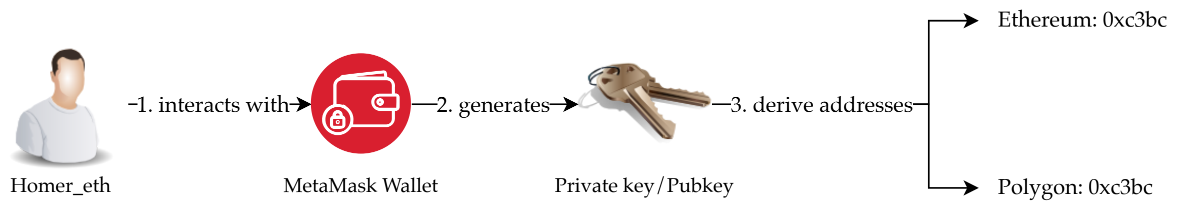


**Figure 6.9:** Illustration of address coexistence on EVM-compatible blockchains

### UTXO Transactions

Given a UTXO transaction with transaction inputs $I$ and outputs $O$, the set of inputs is $I = \{(a_i, v_i, s_i) | 1 \leq i \leq n\}$, where $a_i$ is the address unlocking the input, $v_i$ is the value being spent by that input, and $s_i$ is the script type defining the spending conditions for that input. The set of outputs is $O = \{(a_j, v_j, s_j) | 1 \leq j \leq m\}$, where $a_j$ is the address locking the output, $v_j$ is the value of the output, and $s_j$ is the script type specifying conditions for spending the output.

*1. Common-input-ownership*

The clustering process commences with the common-input-ownership heuristic that is based on the idea that all inputs of a UTXO transaction are likely controlled by the same real-world entity, since the initiator of a transaction must possess the private keys for all involved input addresses to spend the funds. This heuristic is implemented in the Kosmosis prototype as follows:

$$\forall (a_j, v_j), (a_k, v_k) \in I : a_j = a_k \Rightarrow \mathcal{RWE}(a_j) = \mathcal{RWE}(a_k)$$

where $a_j, a_k$ are the addresses that unlock inputs $(a_j, v_j)$ and $(a_k, v_k)$; and $\mathcal{RWE}(a_j)$, $\mathcal{RWE}(a_k)$ are the real-world entities controlling addresses $a_j, a_k$, respectively.

*2. Address Reuse*

Using the set of addresses from all previously observed transactions $A_{\text{prev}}$, the address reuse heuristic can be applied under the following conditions:

1. The transaction has more than one input address ($n > 1$).
2. No address is both in the input and output ($\forall a_i \in I, \forall a_j \in O, a_i \neq a_j$).
3. Exactly one output address is new ($\exists! a_k \in O$ such that $a_k \notin A_{\text{prev}}$ and $a_k \notin I$).
4. All other output addresses have been observed before ($\forall a_j \in O, j \neq k, a_j \in A_{\text{prev}}$).

*5. Equal-Output CoinJoin*

CoinJoin is a mechanism used to increase privacy by combining multiple payments from several users into a single transaction, making it more difficult to determine who paid whom. The goal of algorithm 2 is to cluster entities associated with outputs having the same value, assuming these outputs are controlled by different entities participating in a CoinJoin transaction. A transaction is considered to be a CoinJoin transaction if there exist at least two outputs in $O$ with equal values (i. e., $\exists v_j, v_k \in O$ where $v_j = v_k$ and $j \neq k$).

**Algorithm 2** Equal-Output CoinJoin

**Input:** A Bitcoin transaction with a set of outputs $O$.
**Output:** Number of real-world entities involved in the CoinJoin transaction.

```
1: $entityCount \leftarrow 0$
2: $outputValues \leftarrow$ a map of $v_j$ to count
3: for each $(a_j, v_j) \in O$ do
4:     if $v_j$ in $outputValues$ then
5:         $outputValues[v_j] \leftarrow outputValues[v_j] + 1$
6:     else
7:         $outputValues[v_j] \leftarrow 1$
8:     end if
9: end for
10: for each $value, count$ in $outputValues$ do
11:     if $count \geq 2$ then
12:         $entityCount \leftarrow entityCount + count$
13:     end if
14: end for
```

*2. Script Type*

A real-world entity tends to use a consistent script type (e. g., P2SH, P2PKH) for all inputs within a transaction. If there is an output address whose script type matches the uniform script type of all input addresses, this output address is likely the change address. It is especially indicative if this script type is unique or rare among the outputs. The heuristic for identifying a change address based on script type consistency can be formalized as follows:

- Consistency Check: Verify that all inputs in $I$ have the same script type. This is determined by checking if there exists a common script type $s^*$ for all inputs $(a_i, s_i) \in I$, $s_i = s^*$.
- Change Address Identification: An output address $(a_j, s_j) \in O$ is identified as the change address if $s_j$ matches the common script type $s^*$ of the inputs, and it is the only output or one of the few outputs with this script type, given that other outputs may have different script types.

The heuristic for identifying a change address based on script type consistency is implemented as follows:

$$(\forall (a_i, s_i), (a_k, s_k) \in I : s_i = s_k = s^*) \land (\exists! (a_j, s_j) \in O : s_j = s^*) \Rightarrow a_j \text{ is change}$$

*3. Round Numbers and Reduced Precision*

This heuristic is based on the observation that change addresses often have values with a greater number of decimal places compared to other outputs in a transaction, due to the nature of transaction fees and the tendency for users to send amounts with round numbers to other parties. This heuristic considers the following criteria:

- Search Criterion: Identify any output $(a_i, v_i) \in O$ for which the decimal length $D(v_i) > 7$. This output is a candidate for being the change address.
- Validation Criterion: Validate the candidate change address by ensuring that for all other outputs $(a_j, v_j) \in O$ where $j \neq i$, the decimal length $D(v_j) < 2$. If this condition is met, $a_i$ is accepted as a change address.

The heuristic for identifying a change address based on round numbers and reduced precision is implemented as follows:

$$\exists (a_i, v_i) \in O : D(v_i) > 7 \land \forall (a_j, v_j) \in O \setminus \{(a_i, v_i)\} : D(v_j) < 2 \Rightarrow a_i \text{ is change}$$

where $D(v_i)$ is the length of decimal places in the value $v_i$ of an output. This is a function that takes a value and returns the number of digits after the decimal point.

What is not considered in this implementation is that the reduced precision number could be in a different currency (e. g., US dollar). Hence, it would require the exchange rate at the time the transaction was mined. Further, it may not accurately identify change addresses in all cases, especially in transactions where users send amounts with unusual precision or where the change address does not follow the typical pattern of having a high number of decimal places.

*6. Optimal Change*

The *optimal change* heuristic is used to analyze Bitcoin transactions to infer which inputs and outputs belong to the same real-world entity, assuming that transactions are optimized to minimize change. This heuristic is based on the observation that when a transaction includes more inputs than necessary to meet the output amount, the extra inputs can give insights into the ownership of the inputs.

For the *optimal change* heuristic, algorithm 3 calculates the total values of inputs ($V_{\text{in}} = \sum_{i=1}^{n} v_i$) and outputs ($V_{\text{out}} = \sum_{j=1}^{m} v_j$), then iterates through the outputs to find the one that, when considered as change, results in the minimal overpayment by the inputs. This output $o_c \in O$ is tentatively identified as the change.

**Algorithm 3** Optimal Change Heuristic

**Input:** A Bitcoin transaction $T$ with sets of inputs $I$ and outputs $O$.
**Output:** The index of the output likely to be the change, or -1 if undetermined.

```
1: $totalInputValue \leftarrow 0$
2: $totalOutputValue \leftarrow 0$
3: $changeIndex \leftarrow -1$
4: $minimalDifference \leftarrow \infty$
5: for each $(a_i, v_i) \in I$ do
6:     $totalInputValue \leftarrow totalInputValue + v_i$
7: end for
8: for each $(a_j, v_j) \in O$ do
9:     $totalOutputValue \leftarrow totalOutputValue + v_j$
10: end for
11: for each $j$ in $1 \dots m$ do
12:     $v_j \leftarrow O[j].value$
13:     $difference \leftarrow totalInputValue - (totalOutputValue - v_j)$
14:     if $difference \geq 0$ and $difference < minimalDifference$ then
15:         $minimalDifference \leftarrow difference$
16:         $changeIndex \leftarrow j$
17:     end if
18: end for
```

### 6.4.4 RDF Mapping

Initially, the ontology, which defines the conceptual framework for interpreting the blockchain data, is loaded into an `RDFGraph` utilizing the `RDFLib` library. This foundational step ensures that the subsequent mapping aligns with the established semantic structure, enabling coherent interpretation within the KG. The core of the mapping process is encapsulated in the `dict_to_rdf` method. This function is designed to recursively traverse the processed transaction object, systematically converting each element into RDF triples. This recursive approach is pivotal as it allows for the comprehensive representation of nested structures inherent in transaction data, thereby preserving the relational intricacies of the data.

For each key-value pair within the transaction object, the mapping follows a structured approach. The URI of the main entity (e. g., sender, recipient) serves as the subject of the triple. To account for the coexistence of blockchain addresses and ensure the uniqueness of entities, a prefix is attached to distinguish between identical addresses or transaction hashes that may exist across different blockchains. For example, the prefixes `eth:`, `btc:`, and `matic:` are utilized to denote Ethereum, Bitcoin, and Polygon blockchain addresses, respectively. The predicate of the triple is derived from the ontology, describing the relationship between the subject and the object. The object data type varies depending on its value:

- If the value is a string literal (e. g., "Mint NFT"), it is represented as a literal with an appropriate datatype (e. g., `xsd:string`).
- If the value is another dictionary (e. g., representing a transaction), the function recursively converts this dictionary into RDF triples, treating it as a sub-entity within the graph.
- If the value is a list (e. g., `inputArgs`), the function iterates through the list, generating separate triples for each element, thereby accommodating multi-valued properties.

Listing 6.3 showcases the final RDF representation of the processed transaction object from the previous steps. This representation is subsequently loaded into the KG using the methods `store_entities` and `store_relations`.

```
<http://ontology.kosmosis.net#eth:0x0a5>
        rdf:type owl:NamedIndividual ,
                :Token_Contract ;
        :implements :ERC-721 ;
        :address "0x0a5" ;
        :tokenName "Ether Reapers" .

<http://ontology.kosmosis.net#eth:0x819>
        rdf:type owl:NamedIndividual ,
                :Call_Transaction ;
        :minedOn :Ethereum ;
        :minedInBlock "13456446"^^xsd:unsignedInt ;
        :from <http://ontology.kosmosis.net#eth:0xB24> ;
        :to <http://ontology.kosmosis.net#eth:0x0a5> ;
        :action "Mint 1 Ether Reapers NFT" ;
        :value "80000000000000000"^^xsd:unsignedInt .

<http://ontology.kosmosis.net#eth:0xB24>
        rdf:type owl:NamedIndividual ,
                :Externally_Owned_Account ;
        :hasAccountTag :Ether_Reapers_Minter .
```

**Listing 6.3:** The final processing result after entity resolution

## 6.5 Text-Processing Workflow

The workflow starts with the input of unstructured data from the 𝕏 Filtered stream API, which is incrementally streamed and parsed via a long-lived HTTP request into the pipeline for processing. The filtered stream allows the use of custom rules per stream. To identify mentions or announcements of new tokens, the following rules were defined to target keywords and common phrases that are likely to appear in posts announcing or discussing new tokens (words are placed within quotation marks to filter for exact matches):

“new token” AND “Ethereum” OR “Polygon”

“new contract” AND “Ethereum” OR “Polygon”

“token” AND “live now”

Since the system must be able to ingest data from possibly multiple EVM-compatible blockchains that follow the same address patterns, it is important to further distinguish possibly identified addresses using hashtags, which are likely being used in conjunction with the aforementioned keywords: #eth OR #ethereum, #matic OR #polygon.

Regular expressions are applied using the `blockchain_address_recognition` method on parsing a new post, because the filtered stream does not support them. Table 6.1 lists the regular expressions that are applied to extract the desired type of address.

| **Address Type** | **Regular Expression** |
|---|---|
| EVM | `0x[a-fA-F0-9]{40}` |
| P2PKH | `[13][a-km-zA-HJ-NP-Z1-9]{25,34}` |
| P2SH | `3[a-km-zA-HJ-NP-Z1-9]{25,34}` |
| Bech32 (SegWit) | `bc1[a-zA-HJ-NP-Z0-9]{25,59}` (adopted from Cyble [124]) |

**Table 6.1:** Regular expressions for extracting blockchain addresses

Figure 6.10 depicts the classes responsible for extracting knowledge from 𝕏 posts, or more generally, text. The first step in processing Text is the removal of superfluous whitespaces and the expansion of contractions to their full forms, thereby standardizing the text format for consistent processing. Following this preliminary cleansing, Spacy is used to generate a tokenized `Doc` object, representative of the processed text. This object then undergoes further processing through a sequence of steps, including tagging, parsing, and entity recognition. The tagger assigns POS tags to each text token, thereby elucidating the grammatical roles of words within the context of sentences. Subsequently, the parser assigns dependency labels, which establish the relationships between tokens, thereby constructing a structured representation of the sentence. Lastly, NER identifies and classifies named entities present in the text into predefined categories, such as the names of persons, organizations, and locations. The next step is relation extraction. This process involves identifying and extracting relationships between the named entities that were previously recognized. For instance, it could determine that a person named “Alice” works for a company named “Acme.”

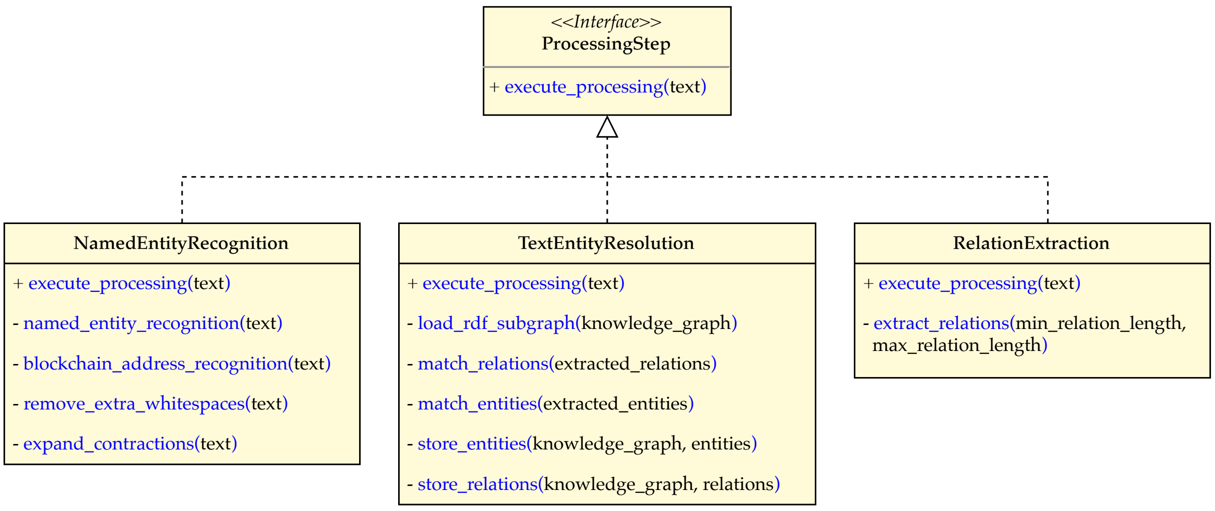

<<Interface>>
ProcessingStep
+ execute_processing(text)
NamedEntityRecognition
+ execute_processing(text)
- named_entity_recognition(text)
- blockchain_address_recognition(text)
- remove_extra_whitespaces(text)
- expand_contractions(text)
TextEntityResolution
+ execute_processing(text)
- load_rdf_subgraph(knowledge_graph)
- match_relations(extracted_relations)
- match_entities(extracted_entities)
- store_entities(knowledge_graph, entities)
- store_relations(knowledge_graph, relations)
RelationExtraction
+ execute_processing(text)
- extract_relations(min_relation_length, max_relation_length)

**Figure 6.10:** UML class diagram for the text-data processing workflow

Furthermore, `relextract` from the NLTK, facilitates the extraction of relational triples from the text. These triples, which consist of entities and the relationships between them, are crucial for the identification and analysis of the underlying semantic networks within the document, thus providing insights into its content and context. The final step in the text-processing workflow is entity resolution, achieved through blocking and matching. For each new entity, the system identifies all other entities within the KG that must be considered for matching. Due to the growing size of the KG, through the incremental updates, it is important to limit the matching process to as few candidates as possible [4]. The method of limiting candidates is known as blocking, which confines the matching process to entities of the same or most similar entity type. Following the blocking that serves as a preliminary filtering step, the matching is performed. This involves a pairwise comparison of the new entities with those existing entities in the KG identified during the blocking phase. Finally, the result is mapped to an RDF format at the end, before it is loaded into the KG. Figure 6.11 illustrates the text-processing procedure from pre-processing to entity resolution for the example of the Ether Reapers announcement post "Ether Reapers NFT mints today! Contract: 0x0a5." The "today" value is resolved to "10/20/2021" using the post metadata that contains the date.

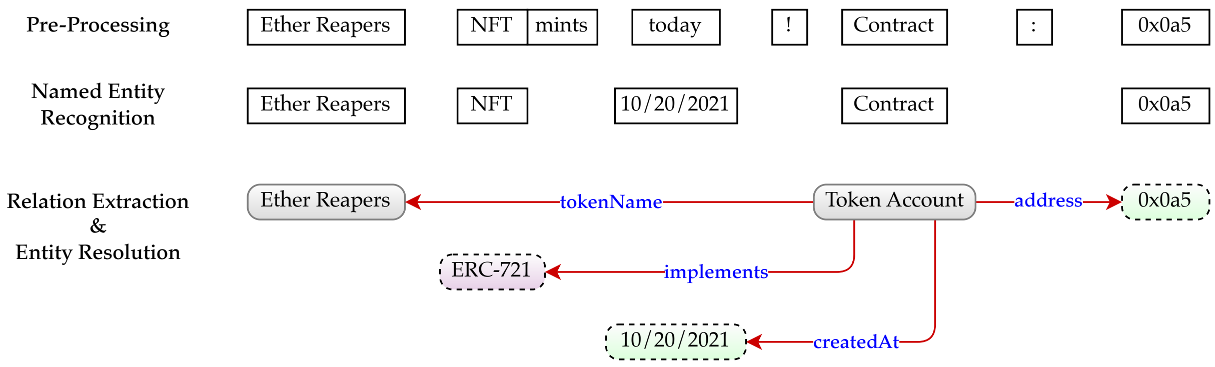


**Figure 6.11:** Text-processing example of the Ether Reapers announcement post

## 6.6 Enrichment Data-Processing Workflows

Data enrichment enhances the KG by adding missing attributes or relationships that may not have been evident or available in the primary data sources. Further, it helps to correct inaccuracies, thereby improving the overall quality of the data in the KG. The first workflow focuses on integration of attributions data that is sourced from community-curated datasets of tagged blockchain addresses to real-world entities based on their known activities or associations to real-world entities. The second workflow is for external knowledge bases, aiming to augment the real-world entities integrated within the KG with detailed descriptions.

### 6.6.1 Attributions

Attributions involve the mapping of blockchain addresses to their corresponding real-world entities. Table 6.2 shows an excerpt from the attributions database. This task is largely dependent on data sourced from a network of experts, such as team members from blockchain projects. The input data for the attribution process is typically not consistent in its timing as it depends on when the experts provide updates or when new information becomes available. As a result, the enrichment data-processing workflow is designed to operate at regular intervals, ensuring that the KG is updated systematically and remains as current as possible with the latest available data.

| Id | Blockchain | Blockchain Address | Address Label (Tag) | Real-World Entity |
|---|---|---|---|---|
| 1 | Ethereum | 0x503 ... 23da | Hot Wallet 3 | Coinbase |
| 2 | Ethereum | 0xBE0 ... 3E8 | Cold Wallet 7 | Binance |
| 3 | Ethereum | 0x94A ... daf | Coin Mixer | Tornado Cash |
| 4 | Ethereum | 0xA0c... C77 | Matic Bridge | Polygon |

**Table 6.2:** Attribution examples for Ethereum blockchain addresses

### 6.6.2 External Knowledge Base

To further enrich the KG, data from an external knowledge base is integrated. In this case, the Golden Knowledge Graph is utilized due to its concentrated information on tech startups and cryptocurrencies. It offers a significant amount of information about crypto projects, including details about their founders and project descriptions.

The workflow for integrating knowledge from an external KG is event-driven, activated once the knowledge storage indicates the addition of new entities from the social media platform 𝕏. Then the workflow triggers a process to pull in additional background information from the *Golden Facts API* [23]. It uses the 𝕏 username of a real-world entity in the KG as a unique identifier to fetch the enrichment data. Listing 6.4 is an illustrative response in JSON-LD format for an organization called Coinbase, proving additional context about the type of real-world entity (organization), the founders, homepage URL, and a description.

```json
{
  "json_ld": {
    "@context": "http://schema.org",
    "@type": "Organization",
    "name": "Coinbase",
    "founder": [
      {
        "@type": "Person",
        "name": "Fred Ehrsam",
        "url": "https://golden.com/wiki/Fred_Ehrsam"
      },
      {
        "@type": "Person",
        "name": "Brian Armstrong",
        "url": "https://golden.com/wiki/Brian_Armstrong"
      }
    ],
    "legalName": "Coinbase, Inc.",
    "url": "https://coinbase.com",
    "text": "Coinbase is a digital asset exchange company."
  }
}
```

**Listing 6.4:** Golden KG response for the real-world entity Coinbase

## 6.7 Summary

This chapter introduced the Kosmosis approach for the incremental construction of a KG that can be utilized to investigate fraudulent activities with crypto assets. Kosmosis is a three-stage data pipeline architecture comprising data ingestion, data processing, and knowledge storage. The data ingestion stage is characterized by its diverse frequency in data acquisition, which encompasses continuous, incremental, periodic, and event-based strategies. This variety ensures a timely collection of relevant data. The data-processing stage is composed of multiple processing workflows tailored to the specific nature of each data source. These sources include semi-structured blockchain data, unstructured text data, and structured enrichment data. Enrichment data plays a pivotal role in this architecture, incorporating attributions for blockchain addresses and leveraging the Golden KG as an external knowledge base. This external base sources descriptions for the real-world entities represented within the KG. A domain-specific ontology was also created to organize and interpret the data within the KG. This ontology is built around four core concepts: blockchain account, blockchain transaction, social media account, and real-world entity.

# Chapter 7

# Evaluation

This chapter provides a technical evaluation of the implemented Kosmosis prototype. The first section provides a comparison between the KG constructed with Kosmosis and the transaction graphs presented in chapter 4. Subsequently, section 7.1 discusses the fitness for use based on the rug pull prevention use case, described in chapter 5. Finally, section 7.3 discusses current the system limitations of the Kosmosis implementation.

## 7.1 Fitness for Use

Written in Python and deployed within a Python 3.11 environment on a Linux Ubuntu OS (22.04), Kosmosis was operated on hardware equipped with a 3.4 GHz Intel i7-13700K CPU and 32 GB RAM. The pipeline ingested blockchain data from Bitcoin, Ethereum, and Polygon through Quicknode archive nodes. Posts from the social media platform 𝕏 were ingested via the full-archive search endpoint, rather than the filtered stream endpoint, because historical data was required to evaluate the prototype against the CQs. The CQs were answered by executing SPARQL queries against the SPARQL API endpoint of the triplestore where the constructed KG was stored.

Table 7.1 summarizes the results of the CQs, that were defined in section 5.4.1, against which the final KG was evaluated. The first CQ confirms that the address 0x872d is indeed associated with the real-world entity Homer_eth. The second question verifies that the address 0xc8a6 deployed the NFT account Ether Reapers. For the third question, a list of Ethereum addresses linked to Homer_eth is provided, however, it is incomplete because 0xf580 is not recognized to belong to Homer_eth. The fourth question, regarding the linkage of blockchain transactions to social media activity, remains partially unfulfilled because it is not possible for the system to distinguish between mentions and announcements. The fifth question confirms the ability to identify an abnormal pattern of fund diversion related to the Ether Reapers contract. The sixth confirms that Homer_eth deployed six NFT token accounts on Ethereum. Lastly, the seventh question confirms that 397 Ethereum addresses minted an Ether Reapers NFT. Overall, most of the competency questions were fulfilled satisfactorily, except for one incomplete and one not fulfilled.

| | Question | Expected | Response | Fulfilled |
|---|---|---|---|---|
| CQ1 | Does the address 0x872d belong to the real-world entity Homer_eth? | true | true | ● |
| CQ2 | Did 0xc8a6 deploy the NFT account Ether Reapers? | true | true | ● |
| CQ3 | Which blockchain addresses belong to Homer_eth on the Ethereum blockchain? | 0x872d, 0x2bdd, 0xc3bc, 0xc8a6, 0xe396, 0xf580 | 0xc3bc, 0x2bdd, 0x872d, 0xc8a6, 0xe396 | ◗ |
| CQ4 | How are the blockchain transactions of Homer_eth linked to his social media accounts and posts on the social media platform 𝕏? | announces | mentions | ◗ |
| CQ5 | Can the abnormal transaction pattern of fund diversion from the Ether Reapers contract to its deployer address be identified? | yes | yes (Transfer of ETH via mintReaper from minters to contract to 0xc8a6) | ● |
| CQ6 | How many NFT token accounts did Homer_eth deploy on Ethereum? | 6 | 6 | ● |
| CQ7 | How many Ethereum addresses minted an Ether Reapers NFT? | 397 | 397 | ● |

**Table 7.1:** Competency question evaluation results. ● - Fulfilled; ◗ - Partially fulfilled

## 7.2 Comparison with Transaction Graphs

Address-transaction graphs illustrate the movement of an asset across transactions and addresses, providing a fundamental view of transaction flows on the blockchain. These graphs are essential for tracing asset movement and identifying patterns within the transactional ecosystem of a blockchain.

User-entity graphs build on address-transaction graphs by clustering them for the potential linking of addresses controlled by the same user or real-world entity. This clustering aims to de-anonymize identities and purpose (cf. section 4.2), facilitating a better understanding of the blockchain usage in practical, real-world contexts.

The money-flow transaction graph, while technically a form of transaction graph, is not included in this comparison, because the money flow transaction graph primarily focuses on the financial aspects of transactions. Its primary aim is to visualize the transfer of value across the network, which, while valuable, offers a narrower scope of analysis compared with the broad and multifaceted insights provided by KGs.

The KG constructed with Kosmosis exhibits three significant advancements compared with address-transaction and user-entity graphs (summarized in table 7.2):

1. **Integration of On- and Off-Chain Data**: Unlike transaction graphs that are limited to on-chain data, KGs incorporate data from both on-chain and off-chain sources. This broader data integration offers a more comprehensive view of entities and their interactions, enriching the analysis with context that extends beyond the blockchain itself.
2. **Extension of User-Entity Representation**: KGs not only link addresses belonging to the same user but also represent users as real-world entities with identifiable names. Moreover, KGs encompass addresses, transactions, social media profiles, posts, and possibly descriptions. This extended representation facilitates a deeper understanding of the users’ roles and activities within and beyond the blockchain ecosystem.
3. **Semantic Data Model**: The KG constructed with Kosmosis utilizes the RDF data model, which semantically describes entities and relations using an ontology. This semantic foundation enables reasoning and inference over the facts contained within the KG for downstream tasks.

| | On-Chain Data | Off-Chain Data | User-Entity | Semantics |
|---|---|---|---|---|
| Transaction Graph | ✓ | | | |
| User Entity Graph | ✓ | | ✓ | |
| Knowledge Graph | ✓ | ✓ | ✓ | ✓ |

**Table 7.2:** Comparison between the constructed knowledge graph and transaction graphs

## 7.3 Limitations

During the development of Kosmosis, the following limitations have emerged. In the context of address de-anonymization, linking addresses across blockchains that use different methodologies for generating blockchain addresses is currently not possible. Specifically, the methodologies for address de-anonymization detailed in the state-of-the-art chapter, including heuristics proposed by Yousaf *et al.* [101] for linking addresses that are owned by the same real-world entity across different blockchains, the single deposit-withdraw and multi-deposit and multi-withdraw coin mixing heuristics proposed by Tang *et al.* [72] for addresses that have interacted with coin mixing services, and the airdrop multi-participation heuristic by Victor [99] could not be applied for Kosmosis. The root of these limitations lies in the prerequisite for these methods to have access to a comprehensive historical dataset blockchain data and for some methods to be based on temporal transaction patterns. Kosmosis is designed to integrate data on a per-transaction basis, consequently, facing challenges in adopting these methods. For instance, transactions to link the addresses may not yet be available at the time of processing the transaction and are therefore prone to inaccuracies as the linkage is based on incomplete or delayed transaction information. Moreover, the investigation into the utility of off-chain data, particularly from social media platforms, has revealed no significant benefits in circumventing these limitations. This is attributed to the deliberate use of obfuscation techniques by malicious actors, who avoid disclosing destination addresses on social media platforms, thereby nullifying the potential for off-chain data to contribute to deanonymization efforts in these contexts.

The implemented ingestion strategy for text data from the social media platform 𝕏 depends on the presence of direct links to blockchain addresses in social media posts. For instance, the ability to link the user Homer_eth with the *Ether Reapers* smart contract (section 5.3) was solely facilitated by the explicit mention of the smart contract address in Homer_eth's announcement post on 𝕏. This example underscores the limitations of the current approach, which may overlook relevant connections in the absence of direct references in social media posts. Therefore, a more sophisticated approach is required to ensure a broader and still relevant dataset is captured to associate 𝕏 users with their respective blockchain addresses. Furthermore, the implemented prototype is confined to processing text content exclusively in the English language. This linguistic constraint narrows the scope of data that is ingested and processed with Kosmosis. As a result, potentially valuable insights from non-English-speaking users on social media platforms are not captured.

Besides the limitation on the data ingestion end, the text-processing workflow does not yet cluster matched entities in the entity fusion step. This type of clustering can enhance the accuracy of entity matching and helps with the subsequent process of knowledge fusion, where matched entities are fused into a single representative entity for the KG.

## 7.4 Summary

The evaluation highlights the development and comparison of various graph models used in blockchain analysis, including address-transaction graphs, user-entity graphs, and the KG. Address-transaction graphs offer a fundamental view of transaction flows, while user-entity graphs aim to link addresses controlled by the same entity, enhancing the understanding of blockchain usage. The KG constructed with Kosmosis, surpasses these models by integrating on- and off-chain data, extending user entity representation, and employing a semantic data model for deeper insights. However, there are limitations in linking blockchain addresses to real-world entities, particularly across different blockchain methodologies and in incorporating off-chain data from social media due to obfuscation techniques and linguistic constraints. Additionally, challenges in entity matching and knowledge fusion processes hinder the effectiveness of the KG in providing a comprehensive blockchain analysis. Overall, most of the competency questions were fulfilled satisfactorily, except for one incomplete and one not fulfilled. The limitations outlined in this chapter highlight areas for future research and development to enhance.

# Chapter 8

# Future Work

While designing Kosmosis, many points of extensibility were discovered. This chapter provides an overview of future work motivated by completing the KG development life cycle in section 8.1, and the rug pull prevention use case in section 8.2.

## 8.1 Implementational Features

This thesis focused on data acquisition, knowledge extraction and processing as well as the development of an initial domain-specific ontology. Future work should focus on completing the KG development life cycle with an implementation for ontology learning (section 8.1.1) and quality assurance (section 8.1.2).

### 8.1.1 Ontology Learning

The current design of the prototype exhibits a constrained extensibility for new data sources due to the static nature of the ontology. While it is possible to add new data sources to the pipeline, each new addition requires a manual integration process to align with the ontology that is applied to the KG. To overcome this constraint, the pipeline should implement methods that assist in evolving the ontology by:

1. **Learning from text data**: The objective here is to identify key concepts and their relationships within a collection of text documents. This can be achieved through linguistic approaches using NLP methods (e. g., syntactic structure analysis) or machine learning approaches (e. g., utilizing co-occurrence analysis) [125]. Another direction of research are topic-modeling algorithms like Latent Dirichlet Allocation [126] that can be used to discover topics within the text. This method can help in grouping documents that discuss similar concepts and discover emerging topics from social media posts.
2. **Learning from blockchain data**: Blockchain account tags are statically defined as concrete instances within the enumerated "Tags" class, providing a structured method for categorizing blockchain accounts according to their roles or behaviors.

However, given the dynamic and ever-evolving nature of the blockchain sector, this static classification system faces significant challenges, specifically with the emergence of new blockchain functionalities and the evolution of existing ones. The rigidity of statically defined tags may lead to scenarios where certain account tags become obsolete or, in a worst-case scenario, the potential to inadequately describe the role or behavior of a blockchain account, stemming from an absence of corresponding tags within the ontology.

### 8.1.2 Quality Assurance

In the realm of crypto asset fraud investigations, the quality of the KG is fundamentally important for it to be considered as a credible source of information. Quality assurance encompasses a spectrum of tasks aimed at both evaluating and improving the KG amid its continuous evolution, including tasks such as quality evaluation, quality improvement and knowledge completion.

To begin, quality evaluation constitutes the preliminary step for quality assurance. It involves a systematic assessment of the KG to identify discrepancies, inaccuracies, or areas that are incomplete. This phase sets the benchmark against which the quality of the KG can be measured and improved. Upon identifying the quality issues within the KG, the focus shifts to quality improvement. This step is dedicated to rectifying the identified problems through the refinement of the KG. Efforts in quality improvement encompass the addition of missing knowledge and the correction of erroneous information. Additionally, data-cleaning techniques can play a significant role in improving the quality of the KG. Identifying outliers or contradictions within the dataset helps in mitigating errors and inconsistencies. Another avenue for quality improvement is to validate the semantic correctness and ensure data integrity within the KG by utilizing solutions such as SHACL or ShEx. These solutions can be used to enforce data integrity and shape constraints. Finally, KG completion aims to add missing entries to the KG based on existing triples. This task can range from determining missing type information and missing relations to missing literals. The conventional method to identify missing type information relies on logical reasoning. Recent studies have proposed statistical approaches that leverage the distribution of relations between entities to predict missing type information. For instance, *StaTIX* [127] could be utilized, which relies on weighted statistical data from various attributes of entities as the basis to predict missing type information. To predict missing relations between two entities, distant supervision [128] can be used to link entities of the KG to a text corpus (e. g., social media posts mentioning crypto assets) using NLP approaches and then try to find patterns in the text between entities.

## 8.2 Rug Pull Detection and Prevention

The incremental construction of a knowledge graph lays the foundation for a variety of applications and downstream tasks (many of which are beyond the scope of this thesis), including the detection of illicit activities, specifically the detection of rug pulls.

### 8.2.1 Rug Pull Detection Algorithms

The constructed KG within this thesis serves as a first step toward the identification of rug pull schemes within the crypto asset sector. However, the generalization, from the exemplary use case to a sophisticated general rug pull classification method, covering various data patterns in the KG, is open research. Subsequent endeavors involve the development of an algorithm capable of discerning rug pull warnings at varying confidence levels. This pursuit commences with the formulation of an intricate SPARQL query. Further exploration in this domain could extend beyond the realm of rug pulls, encompassing a wide array of fraud scenarios prevalent within digital transactions and cryptocurrency exchanges. The ambition to generalize detection algorithms calls for the creation of a dedicated library for fraud detection algorithms. Such a library would encapsulate a variety of methods and models tailored to recognize an extensive spectrum of illicit activities.

### 8.2.2 Rug Pull Prevention Methods

Building on the foundation laid by detection algorithms, the next phase of development focuses on the implementation of proactive measures to prevent engagement with fraudulent actors. This includes an alerting system for crypto wallets and a browser extension for the social media platform 𝕏.

#### Alerting System for Crypto Wallets

The conceptualization of an alerting system represents a pivotal advancement in rug pull prevention. By integrating this system directly with cryptocurrency wallets, users can be furnished with real-time warnings prior to interacting with potential rug pull token contracts. The warning notification includes an assessment of the risk level, based on a risk assessment model that considers various factors such as transaction history, token contract characteristics, and known associations with fraudulent activities. This integration not only facilitates an immediate dissemination of critical information but also empowers users to make informed decisions, thereby safeguarding their investments against fraudulent schemes.

#### Browser Extension for the Social Media Platform 𝕏

Recognizing the significant role social media plays in the promotion and proliferation of cryptocurrency projects, specifically on the social media platform 𝕏, the development of a browser extension emerges as another preventive tool. This extension would display a score indicative of fraudulent crypto behavior adjacent to user profiles, offering a clear and immediate assessment of the trustworthiness of social media endorsements. By providing users with a tangible metric to gauge the legitimacy of cryptocurrency projects and their proponents, this extension aims to significantly reduce the risk of engagement with fraudulent entities.

# Chapter 9

# Conclusion

In the evolving crypto assets sector, fraud detection and prevention remain significant challenges. Knowledge graphs offer a solution by integrating data from diverse sources to identify patterns and hidden connections indicative of fraud, overcoming the limitations of traditional blockchain transaction analysis.

This thesis proposes Kosmosis, an incremental knowledge graph construction pipeline, which updates the knowledge graph with new data from the blockchain (on-chain) and social media (off-chain) without rebuilding the entire knowledge graph. This thesis enhances fraud detection for crypto asset fraud investigations by leveraging knowledge graphs for a more comprehensive analysis, addressing the challenges traditional methods face in linking blockchain addresses to real-world entities and understanding the context of transactions. The research conducted for the Kosmosis prototype was structured around three objectives:

**O1: How to incrementally construct a knowledge graph from on-chain data and off-chain data?**
The primary objective was to devise a solution to construct a knowledge graph as new on-chain and off-chain data becomes available. This was achieved through the design and implementation of a software prototype. The pipeline has the following three stages:
(1) *Data ingestion*, a stage where raw data from various sources is collected and prepared for further processing. The frequency of data acquisition can be (i) continuous to capture real-time updates from sources such as blockchain nodes, (ii) incremental for new posts via the 𝕏 Filtered stream API, (iii) periodic to capture new entries in structured data sources like relational databases at regular intervals, or (iv) event-based, responding to events that are emitted upon new entity additions to the knowledge graph; (2) *Data processing*, a stage where relevant knowledge from the pre-processed data is identified, extracted, and finally fused with the existing knowledge graph. It is partitioned into distinct data-processing workflows tailored to handle each type of ingested data; (3) *Knowledge storage* is the final stage where new knowledge gets loaded into the knowledge graph, which persists in a triplestore.

**O2: How to extract the semantics contained within blockchain transactions?**
The extraction of semantics in blockchain transactions provides additional context for understanding crypto asset flows, specifically for smart contract platforms such as Ethereum. Transactions on smart contract platforms can be broadly categorized into: *value* transactions, which solely transfer the native blockchain currency (e. g., Ether); *contract creation* transactions, wherein a new smart contract or token contract is deployed onto the blockchain by an externally owned account; and *call* transactions, which trigger the execution of a smart contract, including actions like minting or swapping tokens on a blockchain.

To achieve the extraction of the semantics in blockchain transactions, the ABI of smart contracts was utilized. However, this task presented a significant challenge due to the public availability of the bytecode of a smart contract, contrasted with the inaccessibility of its ABI. Therefore, the system attempts to fetch the ABI from web sources such as blockchain explorers at first. If this does not yield a result, the available bytecode is used to reconstruct the ABI. This enables to extract the semantics even if the ABI is not available elsewhere. It is important to note that this approach is specific to smart contract platforms and does not apply to UTXO-based blockchains, like Bitcoin, where transactions are seen as transfer transactions without the layered complexity of smart contract platforms.

**O3: How to automatically link blockchain addresses to their real-world entities during knowledge graph construction?**
Automatically mapping blockchain addresses to real-world entities is a critical process in the investigation of crypto asset fraud. This step is paramount because it allows investigators to peel back the layers of anonymity inherent in blockchain technology, providing a clearer view of the individuals or organizations behind suspicious activities. By utilizing a combination of off-chain information sources, such as social media profiles, attributions from community-curated address labels, and external knowledge bases, alongside clustering heuristics, this process can effectively link blockchain transactions to their real-world counterparts. The clustering heuristics employed include *common-input-ownership*, which assumes control over multiple addresses if they are inputs to the same transaction; *address reuse*, where repeated use of the same address suggests ownership; *equal-output CoinJoin*, identifying transactions intended to obfuscate ownership; *script type*, which classifies addresses based on their script patterns; and *optimal change* to distinguish the change addresses in transactions of UTXO-based blockchains. For account-based blockchains, *deposit address reuse* was found to be an important heuristic for identifying addresses that belong to centralized exchanges. For EVM-compatible blockchains (a subset of account-based blockchains), the *EVM address coexistence* heuristic was described to link coexisting EVM addresses to a real-world entity.

Kosmosis becomes the basis for semantic querying and reasoning over the constructed knowledge graph, facilitating analyses for cybercrime and fraud prevention, with the current focus on rug pulls as a major fraud scheme. The feasibility of the software prototype has been demonstrated by the use case of rug pull prevention, for which competency questions were derived from a user story based on a real rug pull series from 2021. The constructed knowledge graph can serve as a knowledge base for various other downstream tasks such as transaction pattern recognition to track and observe transactions from specific users, or the detection of illicit activities on the blockchain network such as money laundering.

# Appendix A

# Supplementary Material

```
{
  "jsonrpc": "2.0",
  "method": "eth_subscription",
  "params": {
    "result": {
      "transaction": {
        "blockHash": "0x6a5...636",
        "blockNumber": 14304371,
        "from": "0x1a8...D66",
        "gas": 5413447,
        "gasPrice": 56772873536,
        "maxFeePerGas": 63527685620,
        "maxPriorityFeePerGas": 1500000000,
        "hash": "0x2bb...4d8",
        "input": "0x60e06040...",    → Has input data
        "nonce": 2,
        "to": null,                  → No receiver address
        "transactionIndex": 102,
        "value": 0,
        "type": 2,
        "accessList": [],
        "chainId": 1,
        "v": 0,
        "r": "0xac9...0ab",
        "s": "0x75d...872",
      }
    }
  }
}
```

**Listing A.1:** Contract creation transaction on the Ethereum blockchain

```
{
  "jsonrpc": "2.0",
  "method": "get_transaction_receipt",
  "params": {
    "result": {
      "receipt": {
        "blockHash": "0x6a5...636",
        "blockNumber": 14304371,
        "contractAddress": "0x439...094",  → New contract
        "cumulativeGasUsed": 12057812,
        "effectiveGasPrice": 56772873536,
        "from": "0x1a8...D66",
        "gasUsed": 5413447,
        "logs": [
          {
            "address": "0x439...094",
            "topics": [ ... ],
          }
        ],
        "logsBloom": "0x0000..."
        "status": 1,
        "to": null,
        "transactionHash": "0x2bb...4d8",
        "transactionIndex": 102,
        "type": 2
      }
    }
  }
}
```

**Listing A.2:** Transaction receipt for the contract creation transaction

```
{
  "txid": "6d4...a2c",
  "size": 224,
  "version": 1,
  "locktime": 0,
  "fee": 16950,
  "inputs": [
    {
      "coinbase": false,
      "txid": "c69...65f",
      "output": 0,
      "sigscript": "483...db8",
      "sequence": 4294967295,
      "pkscript": "76a...88ac",
      "value": 29885600,
      "address": "1HZxxLkrATEcDBmdXqNLxxUXUhCSy97E2T",
      "witness": []
    }
  ],
  "outputs": [
    {
      "address": "3BuQAXATUC64PGFpFbGebzxz5DyspQT1d0",
      "pkscript": "a91...687",
      "value": 287700,
      "spent": false,
      "spender": null,
    },
    {
      "address": "1HZxxLkrATEcDBmdXqNLxxUXUhCSy97E2T",
      "pkscript": "76a...8ac",
      "value": 29580950,
      "spent": true,
      "spender": {
        "txid" : "C7ad2c1a041f...f2a5" ,
        "input": 0
      }
    }
  ]
}
```

**Listing A.3:** UTXO transaction with one input and two outputs

# Bibliography

[1] K. Grauer, E. Jardine, E. Leosz, and H. Updegrave, "The 2023 Crypto Crime Report," Chainalysis, Annual Report, 2023.

[2] X. Zhu, X. Ao, Z. Qin, *et al.*, "Intelligent financial fraud detection practices in post-pandemic era," *The Innovation*, vol. 2, no. 4, p. 100 176, Nov. 2021, ISSN: 26666758. DOI: `10.1016/j.xinn.2021.100176`.

[3] C. Feilmayr and W. Wöß, "An Analysis of Ontologies and their Success Factors for Application to Business," *Data & Knowledge Engineering*, vol. 101, pp. 1–23, 2016, ISSN: 0169-023X. DOI: `10.1016/j.datak.2015.11.003`.

[4] M. Hofer, D. Obraczka, A. Saeedi, H. Köpcke, and E. Rahm. "Construction of Knowledge Graphs: State and Challenges." arXiv: `2302.11509 [cs.AI]`. (Oct. 11, 2023).

[5] A. Khan, "Graph Analysis of the Ethereum Blockchain Data: A Survey of Datasets, Methods, and Future Work," in *2022 IEEE International Conference on Blockchain (Blockchain)*, Espoo, Finland: IEEE, Aug. 2022, pp. 250–257, ISBN: 978-1-66546-104-7. DOI: `10.1109/Blockchain55522.2022.00042`.

[6] F. Béres, I. A. Seres, A. A. Benczúr, and M. Quintyne-Collins, "Blockchain is Watching You: Profiling and Deanonymizing Ethereum Users," in *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, IEEE, 2021, pp. 69–78.

[7] H. Huang, W. Kong, S. Zhou, Z. Zheng, and S. Guo, "A Survey of State-of-the-Art on Blockchains: Theories, Modelings, and Tools," *ACM Computing Surveys*, vol. 54, no. 2, pp. 1–42, Mar. 12, 2021, ISSN: 0360-0300. DOI: `10.1145/3441692`.

[8] M. Fleder, M. S. Kester, and S. Pillai. "Bitcoin Transaction Graph Analysis." arXiv: `1502.01657 [cs.CR]`. (Feb. 5, 2015).

[9] A. R. Hevner, "A Three Cycle View of Design Science Research," *Scandinavian journal of information systems*, vol. 19, no. 2, p. 4, 2007.

[10] C. Bezerra, F. Freitas, and F. Santana da Silva, *Evaluating Ontologies with Competency Questions*. Nov. 1, 2013, p. 285, 284 pp., ISBN: 978-1-4799-2902-3. DOI: `10.1109/WI-IAT.2013.199`.

[11] Y. Ren, A. Parvizi, C. Mellish, J. Z. Pan, K. Van Deemter, and R. Stevens, "Towards Competency Question-Driven Ontology Authoring," in *The Semantic Web: Trends and Challenges*, V. Presutti, C. d'Amato, F. Gandon, M. d'Aquin, S. Staab, and A. Tordai, Eds., red. by D. Hutchison, T. Kanade, J. Kittler, *et al.*, vol. 8465, Cham: Springer International Publishing, 2014, pp. 752–767, ISBN: 978-3-319-07443-6. DOI: `10.1007/978-3-319-07443-6_50`.

[12] Stanford University. "Protégé." (2023), [Online]. Available: `https://protege.stanford.edu/` (visited on 12/03/2023).

[13] N. Noy and D. Mcguinness, "Ontology Development 101: A Guide to Creating Your First Ontology," *Knowledge Systems Laboratory*, vol. 32, Jan. 1, 2001.

[14] D. Krech *et al.* "RDFLib." (2002), [Online]. Available: `https://github.com/RDFLib/rdflib` (visited on 12/12/2023).

[15] X. Schmitt, S. Kubler, J. Robert, M. Papadakis, and Y. LeTraon, *A Replicable Comparison Study of NER Software: StanfordNLP, NLTK, OpenNLP, SpaCy, Gate*. Oct. 1, 2019, p. 343, 338 pp. DOI: `10.1109/SNAMS.2019.8931850`.

[16] S. Bird, "NLTK: The Natural Language Toolkit," in *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, J. Curran, Ed., Sydney, Australia: Association for Computational Linguistics, Jul. 2006, pp. 69–72. DOI: `10.3115/1225403.1225421`.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson International, Oct. 31, 1994, ISBN: 978-0-321-70074-2.

[18] B. Okken, *Python Testing with Pytest*. Pragmatic Bookshelf, 2022.

[19] QuickNode. "QuickNode - Blockchain Infrastructure Powering Secure, Decentralized Innovation." (2024), [Online]. Available: `https://www.quicknode.com/` (visited on 01/28/2024).

[20] X Corp. "Filtered stream introduction," Twitter API Documentation. (2024), [Online]. Available: `https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/introduction` (visited on 02/06/2024).

[21] Etherscan. "Etherscan: The Ethereum Blockchain Explorer." (2023), [Online]. Available: `https://etherscan.io/` (visited on 12/07/2023).

[22] R. O. Obe and L. S. Hsu, *PostgreSQL: Up and Running, A Practical Guide to the Advanced Open Source Database*, 3rd ed. O'Reilly Media, Inc., Oct. 2017, ISBN: 978-1-4919-6336-4.

[23] Golden. "Golden Facts API - Keep Your Data Fresh." (2024), [Online]. Available: `https://golden.com/product/api` (visited on 02/06/2024).

[24] PolygonScan. "Polygon PoS Chain (MATIC) Blockchain Explorer," Polygon (MATIC) Blockchain Explorer. (2023), [Online]. Available: `http://polygonscan.com/` (visited on 12/02/2023).

[25] J. Lehmann, R. Isele, M. Jakob, *et al.*, "DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia," *Semantic Web Journal*, vol. 6, Jan. 1, 2014. DOI: `10.3233/SW-140134`.

[26] A. Singhal. "Introducing the Knowledge Graph: Things, Not Strings," Google. (May 16, 2012), [Online]. Available: `https://blog.google/products/search/introducing-knowledge-graph-things-not/` (visited on 12/15/2023).

[27] B. Abu-Salih, "Domain-specific knowledge graphs: A survey," *Journal of Network and Computer Applications*, vol. 185, p. 103 076, Jul. 2021, ISSN: 10848045.

[28] M. Bergman. "A Common Sense View of Knowledge Graphs," AI3:::Adaptive Information. (Jul. 1, 2019), [Online]. Available: `https://www.mkbergman.com/2244/a-common-sense-view-of-knowledge-graphs/` (visited on 12/15/2023).

[29] P. A. Bonatti, S. Decker, A. Polleres, and V. Presutti, "Knowledge Graphs: New Directions for Knowledge Representation on the Semantic Web," *Dagstuhl Reports*, vol. 8, no. 9, P. A. Bonatti, S. Decker, A. Polleres, and V. Presutti, Eds., pp. 29–111, 2019, ISSN: 2192-5283. DOI: 10.4230/DagRep.8.9.29.

[30] L. Ehrlinger and W. Wöß, "Towards a Definition of Knowledge Graphs," *SEMANTiCS (Posters, Demos, SuCCESS)*, vol. 48, no. 1-4, p. 2, 2016.

[31] A. Hogan, E. Blomqvist, M. Cochez, *et al.*, "Knowledge Graphs," *ACM Computing Surveys*, vol. 54, no. 4, pp. 1–37, Jul. 2, 2021, ISSN: 0360-0300. DOI: 10.1145/3447772.

[32] M. Lal, *Neo4j Graph Data Modeling*. Packt, Jul. 2015, ISBN: 978-1-78439-344-1.

[33] L. Ora, "Resource Description Framework (RDF) Model and Syntax Specification," *http://www. w3. org/TR/REC-rdf-syntax/*, 1999.

[34] N. Francis, A. Green, P. Guagliardo, *et al.*, "Cypher: An Evolving Query Language for Property Graphs," in *SIGMOD'18 Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18, Houston, TX, USA: Association for Computing Machinery, 2018, p. 1433. DOI: 10.1145/3183713.3190657.

[35] J. Pokorný, "Graph Databases: Their Power and Limitations," in *Computer Information Systems and Industrial Management*, K. Saeed and W. Homenda, Eds., vol. 9339, Cham: Springer International Publishing, 2015, pp. 58–69, ISBN: 978-3-319-24369-6. DOI: 10.1007/978-3-319-24369-6_5.

[36] D. Alocci, J. Mariethoz, O. Horlacher, J. T. Bolleman, M. P. Campbell, and F. Lisacek, "Property Graph vs RDF Triple Store: A Comparison on Glycan Substructure Search," *PLOS ONE*, vol. 10, no. 12, Dec. 14, 2015, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0144578. pmid: 26656740.

[37] M. Färber, F. Bartscherer, C. Menne, and A. Rettinger, "Linked Data Quality of Dbpedia, Freebase, Opencyc, Wikidata, and Yago," *Semantic Web*, vol. 9, no. 1, pp. 77–129, 2018.

[38] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[39] M. Arenas and J. Pérez, "Querying Semantic Web Data with SPARQL," in *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Athens Greece: Association for Computing Machinery, Jun. 13, 2011, pp. 305–316, ISBN: 978-1-4503-0660-7. DOI: 10.1145/1989284.1989312.

[40] H. Knublauch and D. Kontokostas, "Shapes Constraint Language (SHACL)," *W3C Candidate Recommendation*, vol. 11, no. 8, p. 1, 2017.

[41] E. Prud'hommeaux, J. E. Labra Gayo, and H. Solbrig, "Shape Expressions: An RDF validation and transformation language," in *Proceedings of the 10th International Conference on Semantic Systems*, 2014, pp. 32–40.

[42] J. E. L. Gayo, E. Prud'Hommeaux, I. Boneva, and D. Kontokostas, *Validating RDF Data*. Morgan & Claypool Publishers, 2017. [Online]. Available: https://book.validatingrdf.com/ (visited on 12/16/2023).

[43] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," Bitcoin.org, White Paper, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf (visited on 12/03/2023).

[44] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," Ethereum Foundation, Yellow Paper c74b55f, 2024. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf (visited on 01/29/2024).

[45] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0.8.13," Solana Labs, White Paper, 2018. [Online]. Available: https://coincode-live.github.io/static/whitepaper/source001/10608577.pdf (visited on 11/29/2023).

[46] L. Breidenbach, C. Cachin, A. Coventry, *et al.*, "Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks," Chainlink Labs, White Paper, 2021. [Online]. Available: https://research.chain.link/whitepaper-v2.pdf (visited on 01/29/2024).

[47] B. Mizrach. "Stablecoins: Survivorship, Transactions Costs and Exchange Microstructure." arXiv: 2201.01392 [q-fin.TR]. (Feb. 26, 2023).

[48] M. Hall and J. Watkinson. "CryptoPunks." (2017), [Online]. Available: https://cryptopunks.app/ (visited on 11/19/2023).

[49] S. Alizadeh, A. Setayesh, A. Mohamadpour, and B. Bahrak, "A Network Analysis of the Non-Fungible Token (NFT) Market: Structural Characteristics, Evolution, and Interactions," *Applied Network Science*, vol. 8, no. 1, p. 38, 2023.

[50] Coinbase Inc. "Coinbase - Buy and Sell Bitcoin, Ethereum, and More with Trust." (2024), [Online]. Available: https://www.coinbase.com/ (visited on 01/19/2024).

[51] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson, "Uniswap v3 Core," Uniswap Labs, White Paper, Mar. 2021. [Online]. Available: https://uniswap.org/whitepaper-v3.pdf (visited on 12/19/2023).

[52] B. White, A. Mahanti, and K. Passi, "Characterizing the OpenSea NFT marketplace," in *Companion Proceedings of the Web Conference 2022*, 2022, pp. 488–496.

[53] K. Grauer and E. Jardine, "Cryptocurrencies and Drugs: Analysis of Cryptocurrency Use on Darknet Markets in the EU and Neighbouring Countries," *The European Monitoring Centre for Drugs and Drug Addiction (EMCDDA)*, 2022.

[54] M. Bartoletti, S. Lande, A. Loddo, L. Pompianu, and S. Serusi, "Cryptocurrency Scams: Analysis and Perspectives," *IEEE Access*, vol. 9, pp. 148 353–148 373, 2021, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3123894.

[55] B. Mazorra, V. Adan, and V. Daza. "Do Not Rug on Me: Zero-dimensional Scam Detection." arXiv: 2201.07220 [cs.CR]. (Jan. 16, 2022).

[56] T. Sharma, R. Agarwal, and S. K. Shukla, "Understanding Rug Pulls: An In-depth Behavioral Analysis of Fraudulent NFT Creators," *ACM Transactions on the Web*, vol. 18, no. 1, pp. 1–39, Feb. 29, 2024, ISSN: 1559-1131. DOI: 10.1145/3623376.

[57] M. H. Nguyen, P. D. Huynh, S. H. Dau, and X. Li, "Rug-pull malicious token detection on blockchain using supervised learning with feature engineering," in *2023 Australasian Computer Science Week*, Melbourne, VIC, Australia: Association for Computing Machinery, Jan. 30, 2023, pp. 72–81. DOI: 10.1145/3579375.3579385.

[58] F. Cernera, M. La Morgia, A. Mei, and F. Sassi, "Token Spammers, Rug Pulls, and Sniper Bots: An Analysis of the Ecosystem of Tokens in Ethereum and in the Binance Smart Chain (BNB)," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 3349–3366.

[59] S. Wang, D. Li, Y. Zhang, and J. Chen, "Smart Contract-Based Product Traceability System in the Supply Chain Scenario," *IEEE Access*, vol. 7, pp. 115 122–115 133, 2019. DOI: 10.1109/ACCESS.2019.2935873.

[60] P. Stangl, "Design and Implementation of a Heterogeneous Blockchain Consortium for a Food Supply Chain Network," Bachelor's Thesis, Ostbayerische Technische Hochschule Amberg-Weiden, Jan. 2022. [Online]. Available: https://www.cyberlytics.eu/theses/all/OTH-AW/BT_2022_Stangl_Philipp_Thesis/BT_2022_Stangl_Philipp_Thesis.pdf.

[61] P. Stangl and C. P. Neumann, "FoodFresh: Multi-Chain Design for an Inter-Institutional Food Supply Chain Network," in *IARIA Cloud Computing 2023: 14th International Conference on Cloud Computing, GRIDs, and Virtualization*, Nice, France, Jun. 2023, pp. 41–46.

[62] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," in *2017 IEEE International Congress on Big Data (BigData Congress)*, Honolulu, HI, USA: IEEE, Jun. 2017, pp. 557–564, ISBN: 978-1-5386-1996-4. DOI: 10.1109/BigDataCongress.2017.85.

[63] A. M. Antonopoulos and D. A. Harding, "Keys, Addresses, Wallets," in *Mastering Bitcoin: Programming the Open Blockchain*, M. Smith, A. Rufino, C. Laylock, and K. Cofer, Eds., 3rd ed. O'Reilly Media, Inc., Nov. 30, 2023, ch. 4, ISBN: 978-1-09-815009-9.

[64] B. Werkheiser. "Archive Nodes - Everything You Need to Know," Alchemy. (Jun. 21, 2022), [Online]. Available: https://www.alchemy.com/overviews/archive-nodes (visited on 02/19/2024).

[65] O. Marin, T. Cioara, L. Toderean, D. Mitrea, and I. Anghel, "Review of Blockchain Tokens Creation and Valuation," *Future Internet*, vol. 15, no. 12, p. 382, Nov. 27, 2023, ISSN: 1999-5903. DOI: 10.3390/fi15120382.

[66] F. Vogelsteller and V. Buterin. "ERC-20: Token Standard," Ethereum Foundation. (Nov. 19, 2015), [Online]. Available: https://eips.ethereum.org/EIPS/eip-20 (visited on 12/16/2023).

[67] W. Entriken, D. Shirley, J. Evans, and N. Sachs. "ERC-721: Non-Fungible Token Standard," Ethereum Foundation. (Jan. 24, 2018), [Online]. Available: https://eips.ethereum.org/EIPS/eip-721 (visited on 12/16/2023).

[68] E. Takeuchi. "Explaining Ethereum Contract ABI & EVM Bytecode." (Jul. 16, 2019), [Online]. Available: https://medium.com/@eiki1212/explaining-ethereum-contract-abi-evm-bytecode-6afa6e917c3b (visited on 12/07/2023).

[69] R. Zhang, R. Xue, and L. Liu, "Security and Privacy on Blockchain," *ACM Computing Surveys*, vol. 52, no. 3, pp. 1–34, May 31, 2020, ISSN: 0360-0300. DOI: 10.1145/3316481.

[70] S. Meiklejohn, M. Pomarole, G. Jordan, *et al.*, "A Fistful of Bitcoins: Characterizing Payments Among Men with No Names," *Communications of the ACM*, vol. 59, no. 4, pp. 86–93, Mar. 2016, ISSN: 0001-0782. DOI: 10.1145/2896384.

[71] A. Narayanan and M. Möser. "Obfuscation in Bitcoin: Techniques and Politics." arXiv: 1706.05432 [cs.CY]. (Jun. 29, 2017).

[72] Y. Tang, C. Xu, C. Zhang, Y. Wu, and L. Zhu, "Analysis of Address Linkability in Tornado Cash on Ethereum," in *Cyber Security*, W. Lu, Y. Zhang, W. Wen, H. Yan, and C. Li, Eds., Springer Nature Singapore, 2022, pp. 39–50, ISBN: 978-981-16-9229-1.

[73] T. Chen, H. Lu, T. Kunpittaya, and A. Luo. "A Review of zk-SNARKs." arXiv: 2202.06877 [cs.CR]. (Oct. 25, 2023).

[74] G. Maxwell. "CoinJoin: Bitcoin Privacy for the Real World." (2013), [Online]. Available: https://bitcointalk.org/index.php?topic=279249.0 (visited on 01/14/2024).

[75] D. Chaum, "Blind Signatures for Untraceable Payments," in *Advances in Cryptology: Proceedings of CRYPTO '82*, Santa Barbara, CA, USA: Springer, 1998, pp. 199–203.

[76] Á. Ficsór, I. A. Seres, Y. Kogman, and L. Ontivero, "Wabisabi: Centrally Coordinated CoinJoins with Variable Amounts," *Cryptology ePrint Archive*, 2021.

[77] K. Kaupe *et al.* "JoinMarket." (2017), [Online]. Available: https://github.com/JoinMarket-Org/joinmarket-clientserver (visited on 12/07/2023).

[78] Á. Ficsór *et al.* "ZeroLink." (2017), [Online]. Available: https://github.com/nopara73/ZeroLink (visited on 12/11/2023).

[79] Samourai Open Source Development Team. "Samourai Whirlpool." (2017), [Online]. Available: https://code.samourai.io/whirlpool (visited on 12/11/2023).

[80] M. Möser and R. Böhme, "Anonymous Alone? Measuring Bitcoin's Second-Generation Anonymization Techniques," in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, Paris: IEEE, Apr. 2017, pp. 32–41, ISBN: 978-1-5386-2244-5. DOI: 10.1109/EuroSPW.2017.48.

[81] G. Tamašauskaitė and P. Groth, "Defining a Knowledge Graph Development Process Through a Systematic Review," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 1, pp. 1–40, Jan. 31, 2023, ISSN: 1049-331X. DOI: 10.1145/3522586.

[82] Z. Nasar, S. W. Jaffry, and M. K. Malik, "Named Entity Recognition and Relation Extraction: State-of-the-Art," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–39, Feb. 2021.

[83] D. Nadeau and S. Sekine, "A Survey of Named Entity Recognition and Classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[84] J. Li, A. Sun, J. Han, and C. Li, "A Survey on Deep Learning for Named Entity Recognition," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 50–70, 2020.

[85] N. Nakashole, T. Tylenda, and G. Weikum, "Fine-Grained Semantic Typing of Emerging Entities," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2013, pp. 1488–1497.

[86] L. Chiticariu, M. Danilevsky, Y. Li, F. Reiss, and H. Zhu, "SystemT: Declarative Text Understanding for Enterprise," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, 2018, pp. 76–83.

[87] M. Atzmueller, P. Kluegl, and F. Puppe, "Rule-Based Information Extraction for Structured Data Acquisition using TextMarker," in *LWA*, vol. 8, 2008.

[88] A. Toral and R. Muñoz, "A Proposal to Automatically Build and Maintain Gazetteers for Named Entity Recognition by Using Wikipedia," in *Proceedings of the Workshop on NEW TEXT Wikis and Blogs and Other Dynamic Text Sources*, 2006.

[89] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[90] A. Saeedi, E. Peukert, and E. Rahm, "Incremental Multi-source Entity Resolution for Knowledge Graph Completion," in *The Semantic Web*, A. Harth, S. Kirrane, A.-C. Ngonga Ngomo, *et al.*, Eds., Springer, Cham, 2020, pp. 393–408, ISBN: 978-3-030-49461-2. DOI: 10.1007/978-3-030-49461-2_23.

[91] X. L. Dong, E. Gabrilovich, G. Heitz, *et al.*, "From Data Fusion to Knowledge Fusion," *Proceedings of the VLDB Endowment*, vol. 7, no. 10, pp. 881–892, Jun. 2014, ISSN: 2150-8097. DOI: 10.14778/2732951.2732962.

[92] J. Bleiholder and F. Naumann, "Data fusion," *ACM Computing Surveys*, vol. 41, no. 1, pp. 1–41, Jan. 15, 2009, ISSN: 0360-0300. DOI: 10.1145/1456650.1456651.

[93] Y. Elmougy and L. Liu, "Demystifying Fraudulent Transactions and Illicit Nodes in the Bitcoin Network for Financial Forensics," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Long Beach CA USA: Association for Computing Machinery, Aug. 6, 2023, pp. 3979–3990. DOI: 10.1145/3580305.3599803.

[94] J. Wu, J. Liu, Y. Zhao, and Z. Zheng, "Analysis of Cryptocurrency Transactions from a Network Perspective: An Overview," *Journal of Network and Computer Applications*, vol. 190, p. 103 139, Sep. 15, 2021, ISSN: 1084-8045. DOI: 10.1016/j.jnca.2021.103139.

[95] D. Ron and A. Shamir. "Quantitative Analysis of the Full Bitcoin Transaction Graph." (2012), [Online]. Available: https://eprint.iacr.org/2012/584 (visited on 03/05/2024).

[96] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun, "Evaluating User Privacy in Bitcoin," in *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*, Springer, 2013, pp. 34–51.

[97] J. D. Nick, "Data-Driven De-Anonymization in Bitcoin," M.S. thesis, ETH-Zürich, 2015. DOI: 10.3929/ETHZ-A-010541254.

[98] Y. Zhang, J. Wang, and J. Luo, "Heuristic-Based Address Clustering in Bitcoin," *IEEE Access*, vol. 8, pp. 210 582–210 591, 2020, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3039570. [Online]. Available: https://ieeexplore.ieee.org/document/9265226/ (visited on 12/03/2023).

[99] F. Victor, "Address Clustering Heuristics for Ethereum," in *Financial Cryptography and Data Security*, J. Bonneau and N. Heninger, Eds., vol. 12059, Cham:

Springer International Publishing, 2020, pp. 617–633, ISBN: 978-3-030-51280-4. DOI: 10.1007/978-3-030-51280-4_33.

[100] M. Wang, H. Ichijo, and B. Xiao. "Cryptocurrency Address Clustering and Labeling." arXiv: 2003.13399 [cs.CR]. (Mar. 30, 2020).

[101] H. Yousaf, G. Kappos, and S. Meiklejohn, "Tracing Transactions Across Cryptocurrency Ledgers," in *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA, Aug. 2019, pp. 837–850, ISBN: 978-1-939133-06-9.

[102] E. Voorhees. "Shapeshift: An open source platform to trade, track, buy, and earn.," ShapeShift. (2023), [Online]. Available: https://shapeshift.com/ (visited on 12/21/2023).

[103] K. Gladych. "Changelly: A Cryptocurrency Exchange Platform." (2023), [Online]. Available: https://changelly.com/ (visited on 12/21/2023).

[104] Arkham Intelligence, "Arkham: A Platform for Deanonymizing the Blockchain," Arkham Intelligence, White Paper, Jul. 4, 2023. [Online]. Available: https://assets-global.website-files.com/62879326fd745f7489b43224/64abc4471879916bc4e2aeb0_Arkham_Whitepaper_FINAL.pdf (visited on 03/15/2024).

[105] T. Pham and S. Lee. "Anomaly Detection in the Bitcoin System - A Network Perspective." arXiv: 1611.03942 [cs.SI]. (Nov. 12, 2016).

[106] W. Chen, Z. Zheng, J. Cui, E. Ngai, P. Zheng, and Y. Zhou, "Detecting Ponzi Schemes on Ethereum: Towards Healthier Blockchain Technology," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18, Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, pp. 1409–1418, ISBN: 978-1-4503-5639-8. DOI: 10.1145/3178876.3186046. [Online]. Available: https://doi.org/10.1145/3178876.3186046.

[107] L. Chen, J. Peng, Y. Liu, J. Li, F. Xie, and Z. Zheng, "Phishing Scams Detection in Ethereum Transaction Network," *ACM Transactions on Internet Technology*, vol. 21, no. 1, pp. 1–16, Feb. 28, 2021, ISSN: 1533-5399. DOI: 10.1145/3398071.

[108] D. S. H. Tam, W. C. Lau, B. Hu, Q. F. Ying, D. M. Chiu, and H. Liu. "Identifying Illicit Accounts in Large Scale E-payment Networks–A Graph Representation Learning Approach." arXiv: 1906.05546 [cs.SI]. (Jun. 13, 2019).

[109] D. Lin, J. Wu, Q. Yuan, and Z. Zheng, "T-Edge: Temporal Weighted Multidigraph Embedding for Ethereum Transaction Network Analysis," *Frontiers in Physics*, vol. 8, p. 204, 2020.

[110] Y. Hu, S. Seneviratne, K. Thilakarathna, K. Fukuda, and A. Seneviratne. "Characterizing and Detecting Money Laundering Activities on the Bitcoin Network." arXiv: 1912.12060 [cs.SI]. (Dec. 27, 2019).

[111] C. G. Akcora, Y. Li, Y. R. Gel, and M. Kantarcioglu, "BitcoinHeist: Topological Data Analysis for Ransomware Prediction on the Bitcoin Blockchain," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed., Special Track on AI in FinTech, International Joint Conferences on Artificial Intelligence Organization, Jul. 2020, pp. 4439–4445. DOI: 10.24963/ijcai.2020/612. [Online]. Available: https://doi.org/10.24963/ijcai.2020/612.

[112] R. Moody. "Worldwide crypto & NFT Rug Pulls and scams tracker." (Nov. 2023), [Online]. Available: `https://www.comparitech.com/crypto/cryptocurrency-scams/` (visited on 11/19/2023).

[113] C. P. Neumann and R. Lenz, "The alpha-Flow Use-Case of Breast Cancer Treatment – Modeling Inter-Institutional Healthcare Workflows by Active Documents," in *Proc of the 19th Int'l Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2010)*, Larissa, GR, Jun. 2010, pp. 12–22. DOI: `10.1109/WETICE.2010.8`.

[114] C. P. Neumann, "Distributed Document-Oriented Process Management in Healthcare," Ph.D. dissertation, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Nov. 2012. DOI: `10.13140/RG.2.2.14719.79521`. [Online]. Available: `https://nbn-resolving.org/urn:nbn:de:bvb:29-opus-39070`.

[115] Zachxbt. "@homer_eth Rug Pull Analysis," X (formerly Twitter). (May 26, 2022), [Online]. Available: `https://x.com/zachxbt/status/1529973318563946496` (visited on 12/05/2023).

[116] Consensys. "MetaMask: A Crypto Wallet & Gateway to Blockchain Apps." (2023), [Online]. Available: `https://metamask.io/` (visited on 12/15/2023).

[117] P. Stangl and C. P. Neumann, "The Kosmosis Use-Case of Crypto Rug Pull Detection and Prevention," Ostbayerische Technische Hochschule Amberg-Weiden, Technical Report CL-2024-01, Feb. 2024.

[118] The Apache Software Foundation. "Apache Jena - TDB," The Apache Software Foundation. (2024), [Online]. Available: `https://jena.apache.org/documentation/tdb/index.html` (visited on 01/04/2024).

[119] Ontotext. "Ontotext GraphDB," Ontotext. (2024), [Online]. Available: `https://www.ontotext.com/products/graphdb/` (visited on 01/10/2024).

[120] O. Erling, "Virtuoso, a Hybrid RDBMS/Graph Column Store," *IEEE Data Engineering Bulletin*, vol. 35, no. 1, pp. 3–8, 2012.

[121] A. Chernysheva, J. Götz, A. Imeraj, P. Korinth, P. Stangl, and C. P. Neumann, "SGDb Semantic Video Game Database: Svelte- und Ontotext-basierte Webanwendung mit einer Graphen-Suche für Videospiele," Ostbayerische Technische Hochschule Amberg-Weiden, Technical Report, Mar. 1, 2023. DOI: `10.13140/RG.2.2.11272.60160`.

[122] Sourcify. "Sourcify: Source-verified smart contracts for transparency and better UX in web3." (2023), [Online]. Available: `https://sourcify.dev/` (visited on 12/07/2023).

[123] F. A. Manzano and J. Little. "Pyevmasm: Ethereum Virtual Machine Disassembler and Assembler." (2024), [Online]. Available: `https://github.com/crytic/pyevmasm` (visited on 01/25/2024).

[124] Cyble. "New Laplas Clipper Distributed via SmokeLoader." (Nov. 2, 2022), [Online]. Available: `https://cyble.com/blog/new-laplas-clipper-distributed-by-smokeloader/` (visited on 12/18/2023).

[125] F. N. Al-Aswadi, H. Y. Chan, and K. H. Gan, "Automatic ontology construction from text: A review from shallow to deep learning trend," *Artificial Intelligence Review*, vol. 53, no. 6, pp. 3901–3928, 2020.

[126] H. Jelodar, Y. Wang, C. Yuan, *et al.*, "Latent Dirichlet allocation (LDA) and topic modeling: Models, applications, a survey," *Multimedia Tools and Applications*, vol. 78, pp. 15 169–15 211, Nov. 28, 2019. DOI: 10.1007/s11042-018-6894-4.

[127] A. Lutov, S. Roshankish, M. Khayati, and P. Cudré-Mauroux, "StaTIX - Statistical Type Inference on Linked Data," in *2018 IEEE International Conference on Big Data (Big Data)*, IEEE, 2018, pp. 2253–2262.

[128] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, Aug. 2009, pp. 1003–1011.

# Glossary

| Name | Description | Symbol | Def |
|---|---|---|---|
| ABI | *Application Binary Interface* – Interface between the high-level code of the smart contract and the low-level binary interaction on the blockchain. It defines how data is formatted and how to call smart contract functions. | | 3.3 |
| API | *Application Programming Interface* – Defines the methods and data formats for requests and responses, enabling developers to access and use functionalities of external software. | | 2.2 |
| CQ | *Competency Question* – Specific questions that define the scope and requirements that a knowledge graph should be able to answer or address. | | 2.1 |
| DeFi | *Decentralized Finance* – A financial system that operates without centralized intermediaries such as banks. Instead, individuals engage in peer-to-peer financial transactions. | | 3.2 |
| ECDSA | *Elliptic Curve Digital Signature Algorithm* – A cryptographic algorithm used to generate and verify digital signatures based on elliptic curve cryptography. | | 3.3 |
| EOA | *Externally Owned Account* – A type of blockchain account that can send transactions on a blockchain and is managed by its owner, who holds the private key. | | 3.4 |
| EVM | *Ethereum Virtual Machine* – A decentralized virtual computer for executing smart contracts on the blockchain. | | 3.3 |

| **Name** | **Description** | **Symbol** | **Def** |
|---|---|---|---|
| GCN | *Graph Convolutional Network* – A type of neural network designed to work with graph data. It generalizes the convolutional operation from grid data (like images) to graph data, enabling feature learning directly on graphs. | | 4.2 |
| KG | *Knowledge Graph* – A graph that consists of semantically described entities, each with a unique identifier, and relations among those entities using an ontological representation. | | 1.0 |
| Kosmosis | A portmanteau of the words "knowledge" and "osmosis." *Kosmosis* incrementally fuses knowledge from heterogeneous data sources into a knowledge graph. | | 1.0 |
| LPG | *Labeled Property Graph* – A data structure used in graph databases where each node (entity) and edge (relationship) can have zero or more labels that categorize their types as well as properties in the form of key-value pairs. | | 3.1 |
| NER | *Named Entity Recognition* – Identifies mentions of named entities in an input text, demarcating mentions of people, organizations, locations, and other types of entities. | | 4.1 |
| NFT | *Non-Fungible Token* – A unique digital asset that proves ownership and authenticity of a digital or real-world asset. Each NFT has a distinct value and cannot be exchanged on a one-to-one basis with other tokens. | | 3.2 |
| Object | The *object* in an RDF triple is the value or resource that is related to the subject by the predicate. The object can be another resource, or it can be a literal value. | $o$ | 3.1 |
| OOD | *Object-Oriented Design* – Planning a system of interacting objects for the purpose of solving a software problem. | | 2.2 |
| OOP | *Object-Oriented Programming* – Programming paradigm based on the concept of objects, which can contain data and code. | | 2.2 |
| OWL | *Web Ontology Language* – Designed by the W3C. A semantic web language for defining and instantiating web ontologies. | | 3.1 |

| **Name** | **Description** | **Symbol** | **Def** |
|---|---|---|---|
| P2PKH | *Pay-to-PubKey Hash* – A type of ScriptPubKey which locks UTXOs to the hash of a public key, instead of the public key itself. | | 4.2 |
| P2SH | *Pay-to-Script Hash* – Locking script that allows to send a UTXO to an address and lock them using a custom locking script. | | 4.2 |
| POS | *Part-of-Speech* tagging – Marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context. | | 4.1 |
| Predicate | The *predicate* in an RDF triple represents the relationship that connects the subject to the object. | $p$ | 3.1 |
| RDF | *Resource Description Framework* – The data exchange standard developed by the World Wide Web Consortium (W3C) for describing resources on the world wide web. | | 2.2 |
| RDFS | *RDF Schema* – A semantic extension of RDF that provides mechanisms for describing groups of related resources and the relationships between these resources. | | 3.1 |
| RWE | *Real-World Entity* – A person or organization of the real world. | $\mathcal{RWE}$ | 1.1 |
| SHA | *Secure Hash Algorithm* – A cryptographic hash function that takes an input (or message) and returns a fixed-size string of bytes. It should be practically impossible to generate the same return value with a different input message (collision resistance). | | 3.3 |
| SHACL | *Shape Constraint Language* – Schema language to set conditions an RDF graph must fulfill. | | 3.1 |
| ShEx | *Shape Expressions* – A high-level language for validating and describing RDF data structures. | | 3.1 |
| SPARQL | *SPARQL Protocol And RDF Query Language* – It is used to retrieve and manipulate data stored in RDF format. | | 3.1 |
| Subject | The *subject* in an RDF triple is the entity or resource. It is typically represented by a URI that uniquely identifies a resource. | $s$ | 3.1 |

| **Name** | **Description** | **Symbol** | **Def** |
|---|---|---|---|
| TDD | *Test-Driven Development* – Development and design paradigm where test cases are defined before the actual code is written to ensure code quality and reliability. | | 2.2 |
| UML | *Unified Modeling Language* – A standardized modeling language used to visualize, specify, and document software system artifacts. | | 2.2 |
| URI | *Uniform Resource Identifier* – An identifier that is a unique sequence of characters used to identify a resource. | | 3.1 |
| UTXO | *Unspent Transaction Output* - An accounting model used in blockchains like Bitcoin in which each unit of currency is treated as an individual UTXO, similar to cash. When a transaction is executed, it consumes one or more existing UTXOs as inputs and creates new UTXOs as outputs. | | 3.4 |

# List of Figures

1.1 Total cryptocurrency value received by illicit addresses from 2017 to 2022. Adopted from Grauer *et al.* [1] . . . . . . . . . . . . . . . . . . . . 2
1.2 Thesis structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.1 UML class diagram of the singleton pattern . . . . . . . . . . . . . . . . 7
2.2 UML class diagram of the factory method pattern . . . . . . . . . . . . . 8
2.3 UML class diagram of the strategy pattern . . . . . . . . . . . . . . . . 8

3.1 John Doe bank account example modeled as LPG . . . . . . . . . . . . . 12
3.2 John Doe bank account example modeled as RDF graph . . . . . . . . . 14
3.3 Overview of different types of crypto assets . . . . . . . . . . . . . . . 15
3.4 Blockchain datastructure. Adapted from Zheng *et al.* [62] . . . . . . . . . 17
3.5 Schematic representation of deploying and reading from smart contracts. Adapted from Takeuchi [68] . . . . . . . . . . . . . . . . . . . . . . . . 18
3.6 Transaction between Alice and Bob using ETH within the account-based model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
3.7 Transaction between Alice and Bob using BTC within the UTXO model 20
3.8 Process of Tornado Cash coin mixing. Adopted from Tang *et al.* [72] . . 22
3.9 Two individual transactions on the left are combined into a CoinJoin transaction on the right. Adopted from Möser and Böhme [80] . . . . . 23

4.1 Illustration of the named entity recognition task. Adapted from Li *et al.* [84] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
4.2 Illustration of the relation extraction task . . . . . . . . . . . . . . . . . 26
4.3 Illustration of the entity resolution task. Adopted from Hofer *et al.* [4] . 28
4.4 Overview of the different types of blockchain graph models . . . . . . . 29
4.5 Illustration of the common-input-ownership heuristic . . . . . . . . . . . 30
4.6 Illustration of the equal-output CoinJoin heuristic . . . . . . . . . . . . . 31
4.7 An illustration of the script type heuristic . . . . . . . . . . . . . . . . . 32
4.8 Illustration of the deposit address reuse heuristic, adopted from Victor [99], and extended with the fund-gathering pattern by Wang *et al.* [100]. Colors indicate the same real-world entity. Arrows indicate the flow of crypto assets between blockchain addresses. . . . . . . . . . . . 34
4.9 Illustration of the airdrop multi-participation heuristic. Colors indicate the same real-world entity. Adopted from Victor [99] . . . . . . . . . . . 35
4.10 Cross-chain transaction patterns. Adopted from Yousaf *et al.* [101] . . . 37

4.11 Reactor blockchain forensic software by Chainalysis . . . . . . . . . . . . 38

5.1 Rug pulls and scams since 2017, using data from comparitech [112] . . 41
5.2 Simplified transaction graph of Homer_eth's NFT rug pulls, showing only transactions that are directly related to the rug pulls . . . . . . . . 42
5.3 KG of Homer_eth's NFT rug pulls . . . . . . . . . . . . . . . . . . . . . . 44

6.1 Overview of Kosmosis, the incremental KG construction pipeline . . . . 49
6.2 Illustration of the blockchain account concept . . . . . . . . . . . . . . . . 50
6.3 Illustration of the blockchain transaction concept . . . . . . . . . . . . . 52
6.4 Illustration of the social media account concept . . . . . . . . . . . . . . 53
6.5 Illustration of the real-world entity concept . . . . . . . . . . . . . . . . . 54
6.6 UML class diagram for the pipeline . . . . . . . . . . . . . . . . . . . . . . 55
6.7 UML class diagram of the concrete ingestion strategies . . . . . . . . . . 55
6.8 UML class diagram of the blockchain data-processing workflow . . . . 56
6.9 Illustration of address coexistence on EVM-compatible blockchains . . . 63
6.10 UML class diagram for the text-data processing workflow . . . . . . . . 69
6.11 Text-processing example of the Ether Reapers announcement post . . . 69

# List of Listings

6.1 Transaction on the Ethereum blockchain . . . . . . . . . . . . . . . . . . . . 57
6.2 Processed transaction after address relation extraction and tagging . . . 60
6.3 The final processing result after entity resolution . . . . . . . . . . . . . . . 67
6.4 Golden KG response for the real-world entity Coinbase . . . . . . . . . . 71

A.1 Contract creation transaction on the Ethereum blockchain . . . . . . . . 83
A.2 Transaction receipt for the contract creation transaction . . . . . . . . . . 84
A.3 UTXO transaction with one input and two outputs . . . . . . . . . . . . 85

# List of Tables

2.1 Blockchain dataset used for evaluation . . . . . . . . . . . . . . . . . . . . 10

3.1 Overview of crypto asset fraud categories . . . . . . . . . . . . . . . . . . . 16

5.1 Rug pull projects by Homer_eth . . . . . . . . . . . . . . . . . . . . . . . 41

6.1 Regular expressions for extracting blockchain addresses . . . . . . . . . 68
6.2 Attribution examples for Ethereum blockchain addresses . . . . . . . . . 70

7.1 Competency question evaluation results. ● - Fulfilled; ◗ - Partially fulfilled 73
7.2 Comparison between the constructed knowledge graph and transaction graphs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74