Analysis and Patterns of Unknown Transactions in Bitcoin

Maurantonio Caprolu*, Matteo Pontecorvi†, Matteo Signorini†, Carlos Segarra‡ and Roberto Di Pietro*

*Division of Information and Computing Technology, College of Science and Engineering
Hamad Bin Khalifa University, Qatar Foundation - Doha, Qatar
†NOKIA Bell Labs - 91620 Nozay, France
‡Imperial College, London, UK

***Abstract*—Bitcoin (BTC) is probably the most transparent payment network in the world, thanks to the full history of transactions available to the public. Though, Bitcoin is not a fully anonymous environment, rather a pseudonymous one, accounting for a number of attempts to beat its anonymity using clustering techniques. There is, however, a recurring assumption in all the cited deanonymization techniques: that each transaction output has an address attached to it. That assumption is false. An evidence is that, as of block height 591,872, there are several millions transactions with at least one output for which the Bitcoin Core client cannot infer an address. In this paper, we present a novel approach based on sound graph theory for identifying transaction inputs and outputs. Our solution implements two simple yet innovative features: it does not rely on BTC addresses and explores all the transactions stored in the blockchain. All the other existing solutions fail with respect to one or both of the cited features. In detail, we first introduce the concept of Unknown Transaction and provide a new framework to parse the Bitcoin blockchain by taking them into account. Then, we introduce a theoretical model to detect, study, and classify, for the first time in the literature, unknown transaction patterns in the user network. Further, in an extensive experimental campaign, we apply our model to the Bitcoin network to uncover hidden transaction patterns within the Bitcoin user network. Results are striking: we discovered more than 30,000 unknown transaction DAGs representing money flows never observed before. To the best of our knowledge, the proposed framework is the only one that enables a complete study of the unknown transaction patterns, hence enabling further research in the fields, for which we provide some directions.**

1. Introduction

Known as the first successful virtual currency with the potential to disrupt the banking system and provide peer-to-peer payments, Bitcoin has been widely adopted in ransomware campaigns [9], such as *Wannacry* [19] and *NotPetya* [17], and many other illicit activities, such as botnet orchestration [12]. The main reasons being the relative diffusion of Bitcoin, as well as privacy [6]. Privacy is easily achievable with Bitcoin pseudonyms in the form of randomly generated addresses that can be used to send/receive money without being linked to any real identity [1]. Being organized into wallets, Bitcoin's addresses can be easily and freely self-generated by end-users (neither banks nor trusted third parties are needed) without any limitation on their number. Indeed, using each address for a single transaction is a strongly advised common practice in the Bitcoin community and has always been its key component in providing privacy, since the resulting transactions' network has always been assumed too complex to track money flows [7]. Such "privacy through complexity" approach has been enhanced in the last years by both academia [18] and online services, such as mixers and tumblers. Similar to the TOR project [10], aimed at concealing user's location and network activities from anyone conducting network surveillance or traffic analysis, mixers and tumblers try to conceal users' addresses that took part in some monetary transactions [8]. The functioning of such services is quite simple and, similarly to TOR, they require to bounce bitcoins through peers in order to make their tracking hard. Furthermore, the bitcoins that are bounced and returned to the user are not the same that were initially sent, since they come from other sources (i.e. other addresses). However, during the last few years, it has been shown that the above "privacy through complexity" approach can be attacked by clustering the addresses into groups that are likely to belong to the same entity (a user, a shop, a mixer etc.). In 2015, David Nick described [14] some of the most famous heuristics being used still to date for Bitcoin address clustering, such as the shadow, consumer, optimal-change, and multi-input. These heuristics, applied to transactions data previously extracted from the blockchain using parsing algorithms, produce in output the clustered user network. All such parsing algorithms suffer from the cognitive bias that Bitcoin transactions are just operations that link one address to another [13]. However, such address-based linkability is not enforced directly in the Bitcoin protocol:

the protocol only verifies that the locking and unlocking scripts do not produce any false statement [2]. This cited bias is also reinforced by the fact that most of the locking and unlocking scripts follow a specific pattern based on asymmetric encryption using randomly generated addresses. However, such a pattern is not mandatory. Consequently, transactions data parsed relying only on standard locking/unlocking scripts risk to be incomplete, and they often are. This incompleteness, in turn, causes a loss of reliability in all modern clustering and de-anonymization techniques. Moreover, intentionally crafted custom transactions could be used to hide illicit money flows, while being completely invisible to modern automatic parsers unable to decode output addresses. As an evidence, note that up to block with height 591,872, Bitcoin's blockchain contains more than 22 million transactions with at least one locking script (or output) not following any well-known locking/unlocking script. We will often refer to these outputs as *unknown* transaction outputs.

Contribution: At glance, we are the first, to the best of our knowledge, to provide a navigation tool within the Bitcoin blockchain that: does not rely on addresses and explores all the transactions stored in the blockchain. In detail, our contributions can be summarized as follow:

- we introduce the general concept of *unknown* transactions, subsuming the less rigorous definition of non-standard transactions provided by the Bitcoin protocol;
- we design a novel (theoretically sound) parsing methodology to study, and likely understand for the first time, *unknown* transaction patterns by using a specific class of graphs called T-DAGs;
- we show that T-DAGs can be efficiently compared via isomorphism, offering a new mechanism for clustering similar transaction patterns.
- we test our algorithms over the Bitcoin network, collecting and analysing all unknown transactions in the ledger from Bitcoin origins until block 591,872;
- to the best of our knowledge, this is the first study of the Bitcoin transaction network involving patterns generated by non-standard transactions.

With reference to the Bitcoin context, our contribution can be used to collect and analyze existing *unknown* transaction patterns (by parsing the ledger) and those generated in the future (by continuously monitoring the network). These patterns, previously ignored by any Bitcoin transactions analysis work, can be used to complete the user network $\mathcal{U}$ and the transaction network $\mathcal{T}$ [16]. Our methodology can be applied to any clustering and de-anonymization technique to improve its effectiveness by leveraging a complete and reliable Bitcoin transactions database. Moreover, our novel solution can be extended immediately to any other system where transaction patterns can be modeled using T-DAGs.

2. Related Work

To the best of our knowledge, very few works have investigated non-standard transactions in the Bitcoin ledger. In [3], the authors analyzed 1,887,708 non-standard transactions containing the `OP_RETURN` instruction. They found that 15% of them are empty transactions, generated by different activities on the Bitcoin network, such as stress tests or DoS attacks. The remaining transactions are not empty but, similarly to the previous ones, they are not used for transferring funds. In fact, they have a different, specific goal: to store data in the Bitcoin ledger. The `OP_RETURN` transactions do not have a valid recipient, since they are not used to transfer funds. Therefore, they cannot be redeemed. For this reason, these transactions are not of particular interest for clustering and de-anonymizing techniques of Bitcoin users, i.e., they are present neither in $\mathcal{T}$ nor in $\mathcal{U}$. A more in-depth analysis of non-standard transactions in the Bitcoin network has been proposed in [4]. The authors explored the ledger collecting and classifying both standard and non-standard transactions to understand why users sometimes do not adhere to the protocol. To achieve this goal, they mainly focus on analyzing non-standard transactions, classifying them into nine different typologies.
Although these studies analyzed some non-standard transactions, their purpose is only to analyze the semantic, considering every transaction as a stand-alone object. Consequently, such transactions are still ignored in the construction of both the user network $\mathcal{U}$ and the transaction network $\mathcal{T}$, leading to incomplete and possibly unreliable data structures. To solve this problem, we first collected all the unknown transactions in the Bitcoin ledger, regardless of their semantic. We then focused on those that have a valid locking script as they have an impact on the de-anonymization and clustering techniques, neglected by all previous works in the field. By using the proposed methodology, our framework is able to correctly parse unknown transactions, identify their patterns, and complete the user network $\mathcal{U}$ and the transaction network $\mathcal{T}$ with additional data never considered before.

3. Unknown Transactions Recognition

In this section, we introduce the general concept of *unknown* transactions, their classification, and the theoretical model to identify patterns generated by *unknown* transactions.

3.1. Unknown Transactions and Working Framework

The Bitcoin protocol provides its community with standard templates that must be used to create the locking and unlocking scripts that make up a transaction. The use of such templates is then enforced by miners using two functions, isStandardTx() and isStandard(), which check the compliance of each transaction's inputs and outputs, respectively. In fact, a transaction is considered standard, and therefore accepted

by the network, only if both functions return TRUE. If even one of them returns FALSE, the transaction is considered non-standard and discarded. This mechanism should prevent any use of Bitcoin transactions other than the ones conceived by the protocol, to avoid the spread of malicious transactions. However, even non-standard transactions can be included in the blockchain, thanks to miners who relax these controls [4]. Similar to the concept of non-standard transaction, we define *unknown* transactions as follow:

Definition 1 (Unknown Transaction)
We call a transaction (TX) unknown *if it contains an input or an output with a* `Null` *value address, i.e. not correctly identified by the Bitcoin Core client.*

This definition embraces a set of Bitcoin transactions, of which non-standards are currently a subset, regardless of what the protocol considers standard or non-standard. The concept of unknown transactions allows us to protect our framework from future variations of the Bitcoin protocol and guarantees compatibility with other blockchain-based systems. For elaborated blockchain analysis, it is a good idea to initially parse all the data from a running miner and, once the data is organized in a more accessible manner, apply further and more complex post-processing. However, we have discovered several issues in existing blockchain's parsing process:

(i) **Excess of abstraction** Blockchain parsers such as BlockSCI [11] introduce a completely new level of abstraction over the one already specified in the reference implementation [5]. Defining new wrappers, lots of different classes, and incomplete references can make a parser difficult to use and debug.;

(ii) **Excess of identifiers** Bitcoin's blockchain is an environment based on uniqueness. Every item must be uniquely identified and hashing algorithms already provide a way to do so. However, some parsers [11] insist on giving an alternative enumeration for transactions and addresses. This makes databases harder to navigate and makes it non-intuitive to mimic the client's behavior or debug the processed data;

(iii) **Using Public Keys as Identifiers** As introduced in Section 1, to the best of our knowledge, existing works [1], [13], clustering Bitcoin Addresses to find real end-users, assume that each transaction output must have an address assigned to it. This is false. In fact, up to block with height 481823, Bitcoin's blockchain contains 3255688 unknown transactions.

Our proposed framework aims to provide a reference for storing Bitcoin's data in a database; minimizing the amount of abstraction involved, reusing whenever possible the identifiers provided by the reference implementation, and keeping the structure simple and clear.

To be consistent with both (i) and (ii), we only introduce the critical functionalities not covered by the Bitcoin core [1]. Further layers of abstraction depending on the application should be detached from the parsing phase to avoid situations where complex post-processing is discredited by incorrect data parsing. This means that all non-relevant information for blockchain navigation is not included in the framework.

3.2. The Framework

Our framework uses only minimal abstraction and provides a robust, reliable, and fast way to navigate through the Bitcoin's blockchain. It is also easily portable: all applications that query or do some sort of post-processing with Bitcoin's data can use it. To fulfill these conditions and to preserve minimality, only the necessary attributes are included. All other features included in the reference implementation, that provide key information about each transaction, but do not improve the exploration of the blockchain, are discarded. In fact, they can be easily obtained by using the identifiers provided by our framework, together with any Bitcoin client.

We present a database layout that only contains two types of entities: `block` and `tx`.

(i) **`block`:** represents a block in the blockchain. It is uniquely identified by two parameters: `hash` and `height`. Both the parameters can be used to retrieve a `block` element from the database without ambiguity. Each `block` element has an additional parameter, `tx`. `tx` is an array of hashes, each one referencing a transaction included in the block (see next item for a description on the `tx` entity). Each element in the array can be uniquely identified, and accessed, by the index of their position within the array. This way, the m-th transaction of the n-th block can be identified without uncertainty;

(ii) **`tx`:** represents a confirmed transaction (TX) in the blockchain. Since the Bitcoin's ledger contains different cases of transactions with the same hash[2], this attribute cannot be used as a unique identifier. We realized that the Bitcoin Core client still uses the hash attribute to uniquely identify a transaction, causing a loss of information: searching for a particular transaction hash, the Core client returns only the last occurrence of that hash in the ledger. As a result, any transaction stored in the blockchain with a hash equal to a more recent transaction will never be returned by the client. For this reason, we uniquely identify a transaction using its attribute pair `<blockhash, hash>` which represents the hash of the block they belong to and its hash, respectively. The `vin` attribute is an array of pairs `<txID, txID[vout]>`. It represents the set of inputs contained in the transaction. Each input can be uniquely identified by their index within the transaction input array. The `txID` attribute from the pair is the `hash` attribute of the TX that contains the output that the

1. The core client cannot find which transaction input is spending a given unspent output.

2. Blocks 91812 and 91842 contain a transaction with hash: "d5d27987d2a3dfc724e359870c6644b40e497bdc0589a033220fe15429d88599".

input is spending and `txID[vout]` is the index of the spent output within the TX that contains it. Symmetrically, the `vout` attribute is an array of pairs `<txID, txID[vin]>` where, if the output is spent by some input in the future, the TX hash, and the index within the transaction where the output is spent, are included. Each output can be uniquely identified through the hash of the transaction they are contained in and their index in the output array (`tx.vout`).

The above introduced structure leads to Definition 2, which we will use often in the rest of the paper.

Definition 2 (TIO)
*A **Unique Transaction Input-Output** (TIO) is an identifier that can uniquely identify all the inputs and outputs contained in confirmed transactions within the blockchain. We denote the set of all inputs and outputs as $< TIO >$.*

A first contribution of our framework is the possibility to *travel to the future* in the blockchain. This allows us to easily identify the paths followed by bitcoins through the blockchain history. Definition 3 formalizes some new terminology related to our traveling mechanism.

Definition 3 (Traveling the Blockchain)
Given a TIO, we define the current terms:

(i) *If the TIO corresponds to an **Input**:*
*(a) The **spending output** of TIO refers to the output that this input is using;*
*(b) The **funded outputs** of TIO refers to the outputs that this input is providing bitcoins to. By Bitcoin design, we assume that the funded outputs for an input are all the outputs contained in the same transaction than the input.*

(ii) *If the TIO corresponds to an **Output**:*
*(a) The **spending inputs** of TIO are all the inputs that funded this output. By Bitcoin design, we assume that all the spending inputs for an output are all the inputs contained in the same transaction than the output.*
*(b) If the output is spent, the **funded input** is the input that is spending the output. Note that, this input will appear in a more recent transaction than the one containing the output.*

A brief summary of the data structure used in our framework is provided in Table 1.

Locking Script. In addition to its TIO, we are also interested in the locking script for an output. By *locking script*, we refer to the script that has to be redeemed in order to spend the output. In the Bitcoin Core reference implementation, it is referred as `scriptPubKey`.
In the later stages of our methodology, we will use TIOs to build the Unknown TX T-DAGs. Instead, the locking scripts will be used to filter our results by removing T-DAGs generated by transactions with purposes other than the transfer of crypto coins.

Table 1: Summary table of our data structure.

block
* hash
* height
+ tx ≔ [+ hash]$_{<n>(*)}$

tx
* hash
+ blockhash
+ vin ≔ [+ txID, + txID[vout]]$_{<n>(*)}$
+ vout ≔ [+ txID, + txID[vin] (#)]$_{<n>(*)}$

Legend:
$*$ ≔ Attribute is a unique identifier for the entity.
$+$ ≔ Attribute of a given entity.
$\#$ ≔ New attributes that do not appear in the reference implementation.
$[\cdots]_{<n>(*)}$ ≔ Array of elements with the attributes specified between brackets. These elements can be uniquely identified within their container by their position in the array (indexed by an integer n).

With both the TIO and the locking script for each output with a `Null` address, we move to the recognition phase.

3.3. Unknown TX T-DAG Construction

In this section, we lay the blockchain data in a graph using our framework, we introduce the concept of **Unknown TX graphs**, and we study the derived **T-DAGs**. The study of these directed graphs will enable us to describe, tailor and identify unknown transaction patterns on the Blockchain.

Definition 4 (TIO graph)
Let $< TX >$ be the set of confirmed transactions in the blockchain. Let $G_{TIO} = (V, E)$ be a directed unweighted graph such that:

(i) $V = < TIO >$

(ii) $$E = \left\{ \bigcup_{t \in <TX>} \left\{ E\left(K_{|I_t|,|O_t|}\right) \bigcup_{\substack{o \in O_t \\ o \notin UTXO}} (o, gFI(o)) \right\} \right\}$$

where gFI returns the funded input of a given output, given a transaction t, I_t and O_t denote t's set of inputs and outputs respectively, and UTXO is the unspent transaction output store.

Lemma 1. *The TIO graph, G_{TIO}, is a directed acyclic graph (DAG).*

Proof. Nodes in the graph represent validated inputs or outputs in the blockchain. This means that, when they were broadcast to the network, each miner checked them. For an input or an output to be validated, they must always point to an event that happened in the past. Each edge then goes from an event that happened further in the past to a more recent one. This timestamp characteristic is sufficient to ensure that there are no cycles. □

Definition 5 (α-nodes)
An α-node is a set of vertices S from G_{TIO} such that, exists a transaction T such that its set of inputs $I_T = S$ and

(i) *S is a `coinbase`[3] transaction, or*

3. A coinbase transaction is a special transaction in the Bitcoin protocol creating new coins as mining rewards [2].

(ii) $\exists\, s \in S$ *such that* s *is spending an output with a* `BTC Address`.

Remark 1. *For each transaction* T*, its set of inputs* I_T *fulfills:*

(i) I_T *is an* α*-node, or*
(ii) $\forall s \in I_T$*,* s *is spending an output with a* `None` *address.*

The introduction of α-nodes and the previous remark identifies a natural contracted graph of G_{TIO}.

Definition 6 (Contracted TIO graph)
The ***Contracted TIO graph****,* G^*_{TIO}*, is the graph resulting of applying the following two transformations to* G_{TIO}*:*

(i) *Identify (contract) all vertices [15] in an* α*-node. Repeat for each different* α*-node contained in* G_{TIO}*.*
(ii) *For each transaction* ***T*** *fulfilling the second condition in Remark 1,*
(a) for each spending output ***o*** *of each input in* I_T*, add an edge from* ***o*** *to each output in* ***T****.*
(b) remove every vertex in I_T*, as well as its inbound and outbound edges.*

Remark 2. *The transformations applied to* G_{TIO} *do not introduce cycles and, as a consequence,* G^*_{TIO} *is also a DAG.*

To define the subgraphs in the TIO graph relevant for our research, we still have to introduce some more concepts.

Definition 7 (Termination application)
Let f *be a function,* $f : < TIO > \longrightarrow \{0, 1\}$ *defined as follows:*

$$f(x) = \begin{cases} 0 & \text{if } x\text{'s address is } \texttt{None} \\ 1 & \text{otherwise} \end{cases}$$

Given a weakly connected single-source DAG, we call all nodes that are not the source nor sinks *internal nodes*.

Definition 8 (Unknown TX graph)
An ***Unknown TX graph*** *is a single-source, weakly connected, maximal induced subgraph of* G^*_{TIO} *such that:*

(i) *The source* s *is an* α*-node.*
(ii) *Each sink* t *fulfills* $f(t) = 1$*.*
(iii) *Each internal node* v *fulfills* $f(v) = 0$*.*

Unknown TX graphs will be our object of study for the rest of the paper. From their construction, we observe the following points.

Definition 9 (T-DAG)
A ***T-DAG*** *is a single-source directed unlabeled acyclic weakly connected graph.*

Remark 3. *Lemma 1 and Remark 2 prove that an Unknown TX graph is a T-DAG.*

From now on, we will refer to Unknown TX graphs as **Unknown TX T-DAGs**[4].

4. Unlike trees, a vertex in a T-DAG may have more than one parent.

Remark 4. *If we fix a source* s*, then there exists only one Unknown TX T-DAG with* s *as its root. We can then denote as* $G(s)$ *the Unknown TX DAG generated by a given root* s*.*

Definition 10 (Set of Unknown TX T-DAGs)
We define the set of Unknown TX T-DAGs, $\mathcal{D}$*, as follows:*

$$\mathcal{D} = \{G(s) : s \text{ is an } \alpha\text{-node and } G(s) \text{ has at least two vertices}\}$$

Algorithm 1 presents a procedure to generate the Unknown TX T-DAG given an α-node s.

Algorithm 1 Unknown TX T-DAG Generation from its root.

```
1:  procedure T-DAG GENERATION(s)
2:      G ← Graph()
3:      S ← Stack()
4:      G.addNode(s)
5:      for all tx_out in getFundOutput(s) do
6:          G.addNode(tx_out)
7:          G.addEdge(s, tx_out)
8:          S.push(tx_out)
9:      while ! S.isEmpty() do
10:         tx_out ← S.pop()
11:         if ! f(tx_out) then
12:             tx_in ← getFundInput(tx_out)
13:             for all new_out in getFundOutput(tx_in)
    do
14:                 G.addNode(new_out)
15:                 G.addEdge(tx_out, new_out)
16:                 S.push(new_out)
```

An example of an Unknown TX DAG is presented in Figure 1. Note that, we attach the associated address for each node.

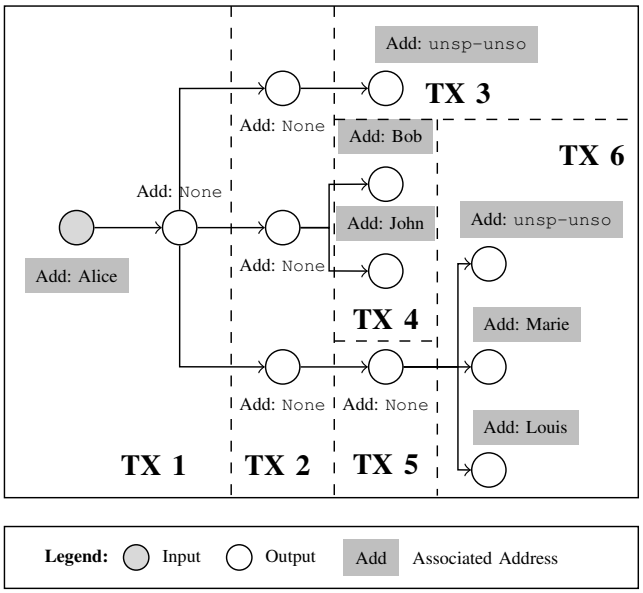

Figure 1: Representation of an Unknown TX T-DAG.

A total ordering for T-DAG isomorphism classes. Let us first introduce the notation that we will use in this section. Let T be a T-DAG (see Definition 9) with its root denoted by t. Given a T-DAG $T = (V, E)$, we say that $|T| \coloneqq |V|$. Given a vertex $v \in V$, $\Gamma^+(v)$ is its outdegree. Lastly, given a T-DAG T the children of the root t are $t_1, \dots, t_{\Gamma^+(t)}$. The set $(T_1, \dots, T_{\Gamma^+(t)})$ denotes the maximal collection of sub DAGs induced on T having $t_1, \dots, t_{\Gamma^+(t)}$ as roots.

Definition 11 ($\prec$ - relation)
Given two T-DAGs, S rooted at s and T rooted at t, we say $S \prec T$ if

(i) $|S| < |T|$, *or*
(ii) $|S| = |T| \wedge \Gamma^+(s) < \Gamma^+(t)$, *or*
(iii) $|S| = |T| \wedge \Gamma^+(s) = \Gamma^+(t) = k \wedge$ *for the first index $i \leq k$ for the ordered sets $(S_1, \dots, S_k)$ and $(T_1, \dots, T_k)$ (where $S_1 \preceq \cdots \preceq S_k$ and $T_1 \preceq \cdots \preceq T_k$) where S_i differs from T_i, it holds $S_i \prec T_i$.*

Definition 12 ($\equiv$ - equality)
Given two T-DAGs T and S, we say $T \equiv S$ if neither $T \prec S$ nor $T \succ S$ hold.

Let $\cong$ be the isomorphism operator. We derive the following lemma:

Lemma 2. *Given two T-DAGs T and S with $|T| = |S| = n$, then $T \equiv S \Leftrightarrow T \cong S$.*

Proof. We prove each implication separately,

[$\Rightarrow$] Let S_n be the symmetric group acting on the vertices of T, $V(T)$. Given $\sigma \in S_n$, we denote the action of σ on $v \in V(T)$ by $\sigma(v)$. We can naturally extend the definition to sets of vertices, $S \subseteq V, \sigma(S) = \{\sigma(v) : v \in S\}$, and to the T-DAG itself $\sigma(T) \coloneqq (\sigma(V), E')$, where $E' = \{(\sigma(u), \sigma(v)) : (u, v) \in E(T)\}$.

Remark 5. *Let S and T be two T-DAGs, $|T| = |S| = n$. Then,*

$$S \cong T \Leftrightarrow \exists \sigma \in S_n \text{ such that } \sigma(S) = T$$

We now prove this direction of the lemma by induction on the size of the T-DAG, $|T|$.

If $|T| = 1$: then both S and T are T-DAGs formed by a single vertex, hence they are the same graph and therefore isomorphic taking the identity permutation.

If $|T| = n$: to prove the inductive step we assume that for any pair of T-DAGs with size $< n$, then $T \equiv S \Rightarrow T \cong S$. If now $|T| = n$, $T \equiv S \Rightarrow |T| = |S| = n$, $\Gamma^+(t) = \Gamma^+(s) = k$, and $(T_1, \dots, T_k) \equiv (S_1, \dots, S_k)$ pairwise, where $S_1 \preceq \cdots \preceq S_k$ and $T_1 \preceq \cdots \preceq T_k$. That is, $\forall i \in 1, \dots k \begin{cases} T_i \equiv S_i \\ |T_i| = |S_i| < n \end{cases} \overset{Ind.H}{\Longrightarrow} T_i \cong S_i \overset{R.5}{\Longrightarrow} \exists \sigma_i \in S_{|T_i|}$ such that $\sigma_i(T_i) = S_i$. We now consider the following permutation: $\sigma = \sigma_1 \circ \cdots \circ \sigma_k \circ (t \to s)$, the composition of all the permutations that match each subtree and the map from a root to the other. σ fulfils that $\sigma(T) = S \overset{R.5}{\Longrightarrow} T \cong S$

[$\Leftarrow$] We argue again by induction on the size of the T-DAG. The base case is the same as before so we do not repeat it. For the induction step, we have:

If $|T| = n$: $T \cong S \Rightarrow |T| = |S| = n \wedge \Gamma^+(t) = \Gamma^+(s) = k$. Additionally, ordering the subtrees $(T_1, \dots, T_k)$, $(S_1, \dots, S_k)$ such that $T_1 \preceq \cdots \preceq T_k$ and $S_1 \preceq \cdots \preceq S_k$ necessarily $T_i \cong S_i \forall i \in \{1, \dots, n\}$ with $|T_i| = |S_i| < n \overset{Ind.H}{\Longrightarrow} T_i \equiv S_i \Rightarrow (T_1, \dots, T_k) \equiv (S_1, \dots, S_k) \Rightarrow T \equiv S$. □

Remark 6. *The operators $(\prec, \succ, \equiv)$ induce a total ordering on T-DAGs isomorphism classes.*

Proof. Given T and S two T-DAGs,

(i) **Antisymmetry:** $S \preceq T \wedge S \succeq T \Leftrightarrow \neg(S \succ T) \wedge \neg(S \prec T) \Leftrightarrow S \equiv T \Leftrightarrow S \cong T$
(ii) **Transitivity:** clearly holds by definition.
(iii) **Connex Property:** $S \preceq T \vee S \succeq T \Leftrightarrow \neg(A \succ B) \vee \neg(A \prec B) \Leftrightarrow \neg(A \succ B \wedge A \prec B) \Leftrightarrow \neg 0 = 1$

□

Canonical labeling for T-DAGs. We now introduce a canonical labeling for T-DAGs. We provide unique representatives for T-DAG isomorphism classes and their string representation.

Definition 13 (Δ-operator)
*The **indegree operator** (Δ) is a total ordering on equivalence classes of the $\equiv$-relation. Let T be a T-DAG such that $T_1 \preceq \cdots \preceq T_{\Gamma^+(t)}$. That is, it takes a set of representatives of the $\equiv$-relation and orders it. Formally,*

$$\Delta : \{T_1, \dots, T_{\Gamma^+(t)}\}/\equiv \longrightarrow \{T_1, \dots, T_{\Gamma^+(t)}\}/\equiv$$
$$\{\overline{T_i}, \dots, \overline{T_{i+k}}\} \longmapsto \Delta(\{\bar{T}_i, \dots, \overline{T_{i+k}}\}) \coloneqq (\{\overline{T_i}, \dots, \overline{T_{i+k}}\}, \leq_*)$$

where $k \in \mathbb{N}$ and $(\{\overline{T_i}, \dots, \overline{T_{i+k}}\}, \leq_*)$ is the totally ordered set according to the following relation:

$$\overline{T_i} \leq_* \overline{T_j} \Leftrightarrow \left(\{|\Gamma^-(t_{i_1})|, \dots, |\Gamma^-(t_{i_{\Gamma^+(t_i)}})|\}, \leq\right) \leq \left(\{|\Gamma^-(t_{j_1})|, \dots, |\Gamma^-(t_{j_{\Gamma^+(t_j)}})|\}, \leq\right)$$

That is, $\overline{T_i} \leq_* \overline{T_j}$ iff the non-decreasing indegree sequence of t_i's children is pairwise smaller than that of t_j.

Naturally, applying the operator to the whole set (i.e. $\Delta(T)$) means applying it element-wise in the quotient set, reordering only elements that were considered $\equiv$-equal.

Definition 14 (T-DAG isomorphism classes representative)
Given a T-DAG, T, we reorder it so that $T_1 \preceq \cdots \preceq T_{\Gamma^+(t)}$. We denote the reordered T-DAG with T^*. The representative of T's isomorphism class $\bar{T}$ is defined as $\bar{T} \coloneqq \Delta(T^*)$.

Lemma 3. *Given T and S T-DAGs, $T \cong S \Rightarrow \bar{T} = \bar{S}$. Thus $\bar{T}$ is well defined.*

Proof. Given a vertex $v \in V(T)$, the **height** of v is the number of edges of the longest path between v and one of its leafs. Let $V(T)_h \subset V(T)$, be the set of vertices with height equal to h. We prove that, given an h, the set of maximal induced T-DAGs rooted at $V(T)_h$ and $V(S)_h$, reordered with $\preceq$ and then with Δ, are the same. In particular, when h equals the height of the T-DAG (i.e. the height of its root),

this yields $\bar{T} = \bar{S}$. We proceed by induction on the height h.

If the height is 0: $T \cong S \Rightarrow |\{v \in V(T) : \Gamma^+(v) = 0\}| = |\{v \in V(S) : \Gamma^+(v) = 0\}| = k$. In fact both T-DAGs have the same number of leafs and therefore $\{t_1, \ldots, t_k\} = \{s_1, \ldots, s_k\}$.

If the height is h: we assume that, for heights $\leq h$, the set of T-DAGs reordered with $\preceq$ and then with the Δ operator are the same. $T \cong S \Rightarrow |V(T)_{h+1}| = |\{v \in V(T) : \text{height}(v) = h+1\}| = |\{v \in V(S) : \text{height}(v) = h+1\}| = |V(S)_{h+1}| = k$. We now order $V(T)_{h+1}$ and $V(S)_{h+1}$ in non-decreasing outdegree order and we apply the Δ operator to $\{T_1, \ldots, T_k\}$ and $\{S_1, \ldots, S_k\}$, the T-DAGs with roots in $V(T)_{h+1}$ and $V(S)_{h+1}$. We will refer to the before-mentioned roots as $\{t_1, \ldots, t_k\}$ and $\{s_1, \ldots, s_k\}$, and to the $j-$th sibling of the $i-$th root as t_i^j, where $j \in \{1, \ldots, \Gamma^+(t_i)\}$ (with T_i^j being the T-DAGs rooted at this nodes). $\{T_1, \ldots, T_k\}$ and $\{S_1, \ldots, S_k\}$ ordered in this manner satisfy $T_1 \preceq \cdots \preceq T_k$ and $S_1 \preceq \cdots \preceq S_k$. Furthermore, for each T-DAG T_i, $i \in \{1, \ldots, k\}$, with root t_i at layer with height h, we have that

$$\left.\begin{array}{l} \Gamma^+(t_i) = \Gamma^+(s_i) \\ \forall j \in \{1, \ldots, \Gamma^+(t_i)\} \left\{ \begin{array}{c} \Gamma^-(t_i^j) = \Gamma^-(s_i^j) \\ \text{Ind. H} \Rightarrow T_i^j = S_i^j \end{array} \right. \end{array}\right\} \Rightarrow T_i = S_i$$

All vertices with height $h+1$ have as children the roots of T-DAGs with heights $\leq h$.

If now we make $h+1$ equal T and S's height, we have $V(T)_{h+1} = t$ and $V(S)_{h+1} = s$. Therefore, the set of maximal induced T-DAGS are $\{T\}$ and $\{S\}$ respectively. We have proven that, reordering with $\preceq$ and Δ, both sets are equal. Hence, $\bar{T} = \bar{S}$. □

Remark 7. *It holds $T \cong \bar{T}$.*

Proof. From Definition 14 we observe that, in order to obtain $\bar{T}$ from T, we reorder the subtrees non-decreasingly and we apply the Δ operator. Let then σ be a permutation such that $\sigma(T)$ generates $T_1 \preceq \cdots \preceq T_{\Gamma^+(t)}$. We then consider μ as the permutation resulting of doing σ and Δ one after the other in this order. From Remark 5 it follows that $\mu(T) = \bar{T} \overset{R.\ 5}{\Longrightarrow} T \cong \bar{T}$. □

Definition 15 (T-DAG labeling)
*Given a T-DAG T we identify its vertices traversing T breadth-first with a FIFO queue and, starting from the root, for each new vertex (not identified that we dequeue) we assign it the current vertex count value, increment the count by one and queue its set of children. The **labeling** $lbl(T)$, associated to T, is the string result from traversing the identified T breadth-first with a FIFO queue and, starting from the root, for each vertex (not processed that we dequeue) we append each of its children identifier to the labeling and queue each of its children. We separate children of the same parent with a comma ',', sets of siblings with a colon ':', and we denote the end of the label with a semi-colon ';'* [5]. *We will refer to the identifier (the label) of a vertex v obtained through this procedure as $id(v)$.*

5. We are aware that these separators depend heavily on the labeling implementation.

Definition 16 (T-DAG canonical labeling)
*Given a T-DAG T, the **canonical labeling** of T, $c(T)$ is the **labeling** of its isomorphism class representative, $\bar{T}$. That is, $c(T) \coloneqq lbl(\bar{T})$.*

In a nutshell, the canonical labeling is obtained with a total ordering for T-DAGs, an indegree-based operator and an additional labeling

Before proving the sufficiency of the three operations, we introduce additional concepts regarding labelings.

Definition 17 (Labeling clause)
*Given a labeling $lbl(T)$ of a T-DAG T, a **clause** in the labeling is a set of identifiers contained within either two colons, a colon and a semi-colon, or the beginning of the labeling and a colon.*

Note that, given a labeling $\text{lbl}(T)$, we can index the clauses by order of appearance starting from 0. Thus, we can think of $\text{lbl}(T)$ as an ordered array of clauses: $\text{lbl}(T) = [c_0, \ldots, c_n]$, where $n = |V(T)|$. Thus, an upper-bound to get the length of each clause, by preprocessing the label, is $\mathcal{O}(m)$, where $m = |E(T)|$.

Given a T-DAG labeling $\text{lbl}(T)$, we can obtain an array `out_deg` that in the i-th position contains the outdegree of $v \in V(T)$ such that $\text{id}(v) = i$. Algorithm 2 presents a pseudo-code to do so. The proofs of correctness (Lemma 4) and complexity (Lemma 5) are omitted for space reasons.

Algorithm 2 Outdegree sequence from a label.

```
1:  procedure OUTDEGREE PARSING(lbl(T))
2:      m ← max(lbl(T))
3:      out_deg ← zeros-array(m + 1)
4:      processed ← zeros-array(m + 1)
5:      Q ← FIFO_Queue()
6:      n_id ← 0
7:      for i : 1 to # clauses in lbl(T) do
8:          while ! Q.empty() do
9:              n_id ← Q.dequeue()
10:             if ! processed[n_id] then
11:                 break
12:         if ! processed[n_id] then
13:             processed[n_id] ← True
14:             out_deg[n_id] ← len(c_i)
15:             for all id in c_i do
16:                 Q.queue(id)
17:     return out_deg
```

Lemma 4. *Algorithm 2 is correct.*

Lemma 5. *The Algorithm described in Algorithm 2 is linear in the number of edges of the labeled T-DAG T. Thus, if $m = |E(T)|$, the complexity is $\mathcal{O}(m)$.*

Lemma 6. *Let S and T be T-DAGs with $|S| = |T|$ and $\Gamma^+(s) = \Gamma^+(t) = k$. Let $\bar{S}$ and $\bar{T}$ be their isomorphism*

class representatives.

$$\bar{S} \ncong \bar{T} \Rightarrow \exists\, u \in V(\bar{S}), v \in V(\bar{T}) \textit{ s.t.}$$

$$id(u) = id(v) \textit{ but } \Gamma^{+}(u) \neq \Gamma^{+}(v)$$

where id is the identifier assigned to each vertex when labeled.

Proof. W.l.o.g. we assume $\bar{S} \prec \bar{T}$. This implies that either (i), (ii), or (iii) from Definition 11 must hold. Let us define the following set of indices, $D \coloneqq \{i \in \{1, \ldots, k\} : \bar{S}_i \ncong \bar{T}_i\}$. From the hypothesis we can ensure that $D \neq \emptyset$. For each index $i \in D$, we take u_i the first vertex traversing $\bar{S}_i$ breadth-first with a FIFO queue such that condition (ii) fails. Note that, from Definition 12 and Lemma 2, $\bar{S}_i \ncong \bar{T}_i$ implies that at some point of the recursion (i) or (ii) will fail. And if (i) fails, necessarily does (ii). Let U be, $U \coloneqq \{u_i : i \in D\}$, and u^*,

$$u^* \coloneqq \underset{u \in U}{\operatorname{argmin}}\{id(u)\}$$

Let v^* be the corresponding vertex in $V(\bar{T})$ that made condition (ii) fail for u^*. For all vertices with a smaller id, the outdegree is the same. Additionally, since $\bar{S}$ and $\bar{T}$ are both ordered non-decreasingly, when traversed breadth-first with a FIFO queue, we can affirm that u^* and v^* will be given the same identifier but they have different outdegrees. □

Lemma 7. *Given two T-DAGs T and S, it holds $T \cong S \Leftrightarrow c(T) = c(S)$. Thus, the canonical labeling presented in Definition 16 is well defined.*

Using the constructions and the efficient labeling derived in this section, we can state our main theoretical result.

Theorem 1
The set of unknown inputs/outputs induces a family of subgraphs (patterns), within the Bitcoin User Network, which can be efficiently labeled and tested for isomorphism.

To the best of our knowledge, this is the first result that systematically addresses unknown transactions, which are often neglected in the current technical literature. In the next section, we will use our isomorphism algorithm to cluster TX T-DAGs that interact with the same patterns. Note that, while presented only for unknown transactions, our approach can be immediately extended to any transaction system where patterns can be modeled by T-DAGs.

4. Experimental Results

To validate our methodology, we first parsed the Bitcoin blockchain. Blocks and transactions information were retrieved by querying directly a Bitcoin full-node, importing data into a MySQL database. We then retrieved all the unknown transactions with a valid locking script from our database, and we applied our methodology to build TX T-DAGs and study their patterns. Finally, we clustered all the TX T-DAGs found in the previous step according to their isomorphism classes to group all similar patters that could have been created by the same entity. The above steps are described in more details in the following.

Number of isomorphic TX T-DAG	Height	Cardinality	Number of edges	Number of roots
29218	2	3	2	1
607	2	11	14	2
51	2	6	7	1
36	2	6	5	1
32	2	5	4	1
25	2	7	6	1
20	2	7	6	1
14	2	6	6	1
12	2	4	3	1
9	2	6	5	1
9	2	40002	40000	20000
...	...	...	...	...
1	2260	6878	7176	10
1	514	2073	4144	7
1	383	1568	3096	13
1	381	1059	1058	5
1	2	5	4	1

Table 2: 10 most common isomorphism classes and some other more complex patterns.

Database: Complete and reliable Bitcoin blockchain data are essential to correctly build the TX T-DAGs. The official software release used by the Bitcoin protocol, i.e. Bitcoin-core, is not suitable for this purpose as it is not designed for the analysis of the data contained in the blockchain. To the best of our knowledge, all freely available blockchain explorer tools suffer from the problems described in Section 3. That is, they do not properly handle custom transactions. For this reason, we have designed a MySQL database to store transaction data retrieved by parsing the Bitcoin ledger using our model. We imported all the Bitcoin blockchain data starting from the genesis block 0 (mined on 2009-01-03), up to block 591,872 (mined on 2019-08-26). Our database consists of more than 448 million Bitcoin transactions, over 1.1 billion transaction inputs, over 1.2 billion transaction outputs and around 550 million Bitcoin addresses. We hosted our MySQL database on a Dell Poweredge R740 Server, equipped with 2 CPU Intel® Xeon® Gold 6144 3.5G, RAM 512GB, running Ubuntu Server 18.04.4 LTS. To parse the Bitcoin ledger, we used Bitcoin Core Daemon v.0.18.0.0, running on a Dell XPS laptop equipped with an Intel® Xeon® CPU E3-1505M 2.80GHz, RAM 32GB - OS: Ubuntu 18.04.4 LTS.

Unknown TX T-DAG Construction: After the parsing phase, our database contains all the transactions included in the Bitcoin ledger, both unknown and standard. At this point, we started analyzing the unknown ones, identifying around 22 million α-nodes that can generate an Unknown TX T-DAG (DAGs where the inner nodes are only unknown transaction outputs). We then built a forest by iterating Algorithm 1 for each α-node. Each weakly connected component of this forest represents an Unknown TX T-DAG originating from one or more alpha nodes. We used the library networkx 2.3 (together with the Python 3.5 interpreter) to create and

manage the forest containing all the Unknown TX T-DAGs.
Pruning Phase: By construction, Unknown TX T-DAGs represent transaction patterns in the blockchain network generated by unknown transactions. The root denotes the set of (standard) inputs that generated the pattern. Each inner node represents an unknown transaction output, i.e. an output with a Null value, that has been spent. Finally, each leaf represents either a known transaction output (with a valid Bitcoin address attached to it) or an unspent transaction output (either with a valid or a null address). Therefore, an unknown TX T-DAG of height 1 is trivial: in fact, such graph represents a single transaction with an unknown output that has not been spent. An output of this type could be an invalid/unspendable output or simply a custom but valid output that has not been spent in the considered blockchain portion. In the first case, since its locking script is malformed, this output is impossible to redeem. Therefore, the corresponding Unknown TX T-DAG can never grow further. In the second case, however, if the unknown output is spent in the future, our methodology will capture this event during the update of our database, and the associated Unknown TX T-DAG structure will be updated and considered for further analysis. Following this observation, we pruned the forest by dropping the weakly connected components of height 1 as they do not represent a relevant pattern for either $\mathcal{T}$ or $\mathcal{U}$. Finally, we obtained a forest with 803,782 nodes and 797,432 edges, having 30,333 weakly connected components left, i.e., our Unknown TX T-DAGs. These T-DAGs can be easily integrated into the transaction network $\mathcal{T}$ and the user network $\mathcal{U}$. In this way, bot the blockchain analysis tools and the proposals in the literature which use these data structures will rely on complete information, never considered before.
Isomorphism Detection: In the last step of our methodology, we clustered the 30,333 Unknown TX T-DAGs, obtained in the previous phase, according to their isomorphism classes. For each T-DAG, we built its isomorphism class representative (definition 14) by using the total ordering introduced in Section 3.3. Before performing the clustering procedure, we augmented every TX T-DAGs with more than one root, such as the one depicted in Figure 2.2. In particular, for each graph of the cited type, we created a new root connected to each of the old ones. Consequently, each graph g with multiple roots is converted into a new graph g^* having a single root and the same nodes of g plus one (the new root). Once all the Unknown TX T-DAGs were standardized to have a single root, we clustered them according to their isomorphism class representative. We identified 273 different isomorphism classes. Figure 2 shows the 10 most common classes, while Table 2 reports the height, cardinality, number of edges, and number of roots for the same classes and some other ones characterized by a more complex pattern.
Discussion: Our approach is, to the best of our knowledge, the only one capable of identifying transaction inputs and outputs without relying on Bitcoin addresses, providing a framework to correctly handle unknown transactions. This feature allowed us to detect unknown transaction patterns not captured by state of the art blockchain analysis techniques. The 30,000+ Unknown TX T-DAGs discovered, can be easily integrated into the Bitcoin User Network, finally providing a complete transaction history, taking into account also unknown transactions. It is worth highlighting that even the simplest of the identified structures, such as the one in Figure 2.1, that appears more than 29,000 times in the Bitcoin ledger, is sufficient to make current parsers not to consider potentially relevant data. Indeed, although the transaction contains the root (the payer address) and the leaf (the recipient address), the User Network built without considering unknown transactions will not contain the middle node (unknown output), hence breaking the connection between the two users. Instead, integrating the User Network with our T-DAGs, automatically increases the accuracy of any Bitcoin clustering heuristic used so far, as well as any deanonymization technique, by simply considering never-used before data. In addition, our isomorphism classes could lead to new clustering techniques: non-trivial patterns, i.e. transaction schemes not originated from a normal user behavior, can be used to cluster services/entities that exhibit the same patterns. Other than the 10 most common isomorphism classes shown in Figure 2, we found several other patterns that deserve particular attention from a semantic point of view. As an example, we discovered an Unknown TX T-DAG of height 2260, having 6878 nodes, and 7176 edges. This complex pattern has 10 roots, i.e., is generated by 10 different standard transactions and therefore potentially by up to 10 different entities. Started on 2014-02-19, and concluded on 2014-12-08, such a pattern moved a total of 247.36 Bitcoins that, according to the historical Bitcoin prices, were worth about 92,500 US Dollars at the time of the last transaction (December 2014)—a complete analysis of the just introduced graph, and other interesting ones, will be provided in future work.

5. Conclusion and Future Work

We have shown that the current assumption that each transaction output has an address attached to it, is false. We have identified transactions that violate the cited assumption, labelling them *unknown* transactions. These unknown transactions imply, among other, that current clustering techniques are incomplete. Starting from the above observation, we proposed a theoretical model rooted on sound graph theory to detect, study, and classify patterns of unknown transactions. Exploring the Bitcoin network via our new tool we unveiled non-trivial classes of transaction patterns never considered before. We were able to identify over 30,000 unknown TX T-DAGs. Each of them represents a money flow that is invisible to standard parsing techniques. Some of these patterns show a high level of sophistication, with a complex topology potentially associated with automated payment services. Our novel approach to the Bitcoin graph opens up a brand new vein of research. For instance, the semantic associated to the discovered patterns is still to be explored. Furthermore, by extending the theoretical model to every transaction (using for example flow theory to remove

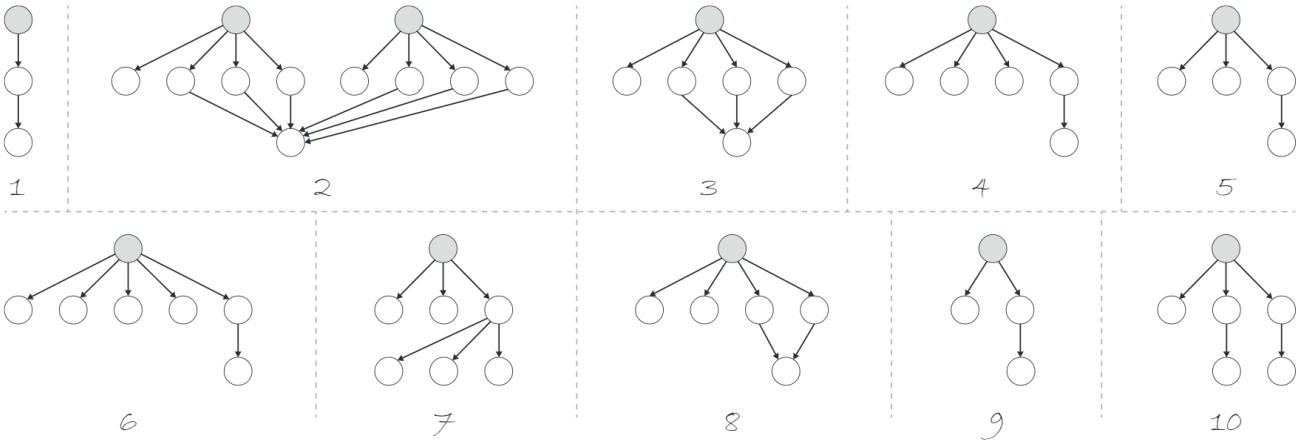

Figure 2: 10 most common isomorphism classes.

cycles in the user network or, in general, focusing on transaction patterns that are represented by T-DAGs), it could be possible to study and cluster other types of non-standard behaviors within the `Bitcoin` environment. Moreover, our solution could be adapted to detect anomalous behaviour in the flourishing field of blockchain-based applications. In conclusion, we believe that the contribution provided in this work, from both a theoretical and practical point of view, other than being interesting on their own—providing for the first time a complete view of the bitcoin blockchain—, also pave the way for further research and applications in the Bitcoin domain and its spin-off technologies.

References

[1] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun, "Evaluating user privacy in bitcoin," in *Financial Cryptography and Data Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 34–51.

[2] A. Antonopoulos, *Mastering Bitcoin*, 2nd ed. 5 St George's Yard Farnham, Surrey: O'Reily, 2017.

[3] M. Bartoletti and L. Pompianu, "An analysis of bitcoin op_return metadata," in *International Conference on Financial Cryptography and Data Security*, Springer. Cham: Springer International Publishing, 2017, pp. 218–230.

[4] S. Bistarelli, I. Mercanti, and F. Santini, "An analysis of non-standard transactions," *Frontiers in Blockchain*, vol. 2, p. 7, 2019. [Online]. Available: https://www.frontiersin.org/article/10.3389/fbloc.2019.00007

[5] Bitcoin Community, "Bitcoin core," https://bitcoincore.org/en/about/, accessed: June 2021.

[6] M. Conti, A. Gangwal, and S. Ruj, "On the economic significance of ransomware campaigns: A bitcoin transactions perspective," *Computers & Security*, vol. 79, pp. 162–189, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818304334

[7] M. Conti, E. S. Kumar, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3416–3452, 2018.

[8] J. Crawford and Y. Guan, "Knowing your bitcoin customer: money laundering in the bitcoin economy," in *2020 13th International Conference on Systematic Approaches to Digital Forensic Engineering (SADFE)*. IEEE, 2020, pp. 38–45.

[9] R. Di Pietro, S. Raponi, M. Caprolu, and S. Cresci, *New Dimensions of Information Warfare*. Springer International Publishing, 2021, vol. 84, part of the Advances in Information Security book series.

[10] H. A. Jawaheri, M. A. Sabah, Y. Boshmaf, and A. Erbad, "Deanonymizing tor hidden service users through bitcoin transactions analysis," *Computers & Security*, vol. 89, p. 101684, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818309908

[11] H. Kalodner, S. Goldfeder, A. Chator, M. Möser, and A. Narayanan, "BlockSci: Design and applications of a blockchain analysis platform," *ArXiv e-prints*, vol. abs/1709.02489, Sep. 2017.

[12] D. Kamenski, A. Shaghaghi, M. Warren, and S. S. Kanhere, "Attacking with bitcoin: Using bitcoin to build resilient botnet armies," in *13th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2020)*, Á. Herrero, C. Cambra, D. Urda, J. Sedano, H. Quintián, and E. Corchado, Eds. Cham: Springer International Publishing, 2021, pp. 3–12.

[13] D. Maesa, A. Marino, and L. Ricci, "Uncovering the bitcoin blockchain: An analysis of the full users graph," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. Alberta, Canada: IEEE, Oct 2016, pp. 537–546.

[14] J. D. Nick, "Data-Driven De-Anonymization in Bitcoin," Master's thesis, ETH Zurich, 2015.

[15] S. Pemmaraju and S. Skiena, *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Cambridge: Cambridge University Press, 2003.

[16] F. Reid and M. Harrigan, *An Analysis of Anonymity in the Bitcoin System*. New York, NY: Springer New York, 2013, pp. 197–223. [Online]. Available: https://doi.org/10.1007/978-1-4614-4139-7_10

[17] C. SecTech, "Notpetya attack," https://cyber-sectech.fandom.com/wiki/NotPetya_Attack, 2017, accessed: June 2021.

[18] D. A. Wijaya, J. K. Liu, R. Steinfeld, S.-F. Sun, and X. Huang, "Anonymizing bitcoin transaction," in *Information Security Practice and Experience*, F. Bao, L. Chen, R. H. Deng, and G. Wang, Eds. Cham: Springer International Publishing, 2016, pp. 271–283.

[19] Wikipedia, "Wannacry ransomware attack," https://en.wikipedia.org/wiki/WannaCry_ransomware_attack, 2019, accessed: June 2021.