Blockchain Privacy Notions Using the Transaction Graph Model

Master Thesis

François-Xavier Wicht

Faculty of Science at the
University of Fribourg

February 2023

Prof. Dr. Christian Cachin
Dr. Duc V. Le

Cryptology and Data Security Research Group
Institute of Computer Science
University of Bern, Switzerland

Abstract

Considerable work explores blockchain privacy notions. Yet, it usually employs entirely different models and notations, complicating potential comparisons. In this work, we use the Transaction Directed Acyclic Graph (TDAG) and extend it to capture blockchain privacy notions (PDAG). We give consistent definitions for untraceability, unlinkability, and confidentiality. Moreover, we specify conditions on a blockchain system to achieve each aforementioned privacy notion. Thus, we can compare the two most prominent privacy-preserving blockchains – Monero and Zcash, in terms of privacy guarantees. Finally, we unify linking heuristics from the literature with our graph notation and review a good portion of research on blockchain privacy.

Contents

1 Introduction **3**

2 Related Work **5**

3 Transaction Graph **7**
3.1 Background . . . 7
3.1.1 Extension for masking . . . 8
3.1.2 Extension for commitments . . . 8
3.2 Privacy-preserving TDAG . . . 8

4 Privacy Notions **12**
4.1 Overview . . . 12
4.2 Adversarial Model . . . 13
4.3 Untraceability . . . 13
4.4 Unlinkability . . . 17
4.5 Confidentiality . . . 19
4.6 Discussion: Relation Between Notions . . . 20
4.6.1 Untraceability and Unlinkability . . . 20
4.6.2 Unlinkability and Confidentiality . . . 21
4.6.3 Untraceability and Confidentiality . . . 22

5 Applications **23**
5.1 Bitcoin . . . 23
5.1.1 Background . . . 23
5.2 Monero . . . 24
5.2.1 Background . . . 24
5.2.2 Transaction graph . . . 25
5.2.3 Model analysis and execution example . . . 27
5.3 Zcash . . . 29
5.3.1 Background . . . 30
5.3.2 Transaction Graph . . . 31
5.3.3 Model analysis and execution sample . . . 33
5.4 Tornado Cash . . . 36
5.4.1 Background . . . 36
5.4.2 Transaction graph . . . 37
5.4.3 Sample transactions . . . 38
5.5 Using PDAG to Unify Deanonymisation Heuristics . . . 40
5.6 Comparisons . . . 44

6 Conclusion **46**

Chapter 1

Introduction

It is widely accepted now that cryptocurrencies bring a true revolution to the financial landscape. Complete decentralization and total transparency redefine trust altogether. As a result, individuals may readily conduct cross-border transactions without fear of censorship or reliance on one (or several) centralized authorities.

Yet, while blockchain technologies promise to democratize the current financial ecosystem, most systems fall short of even the most fundamental privacy standards [17]. If we consider the two most prominent blockchain implementations, Bitcoin and Ethereum, any conducted transaction fully discloses senders, recipients, and the exchanged amount. Thus, the ledger gives each user a transparent and immutable transaction history. Furthermore, identities are quite thinly protected behind pseudonymous addresses. They can easily be compromised through many mechanisms, and once they are, this privacy is difficult and costly to recover [10]. When the public address of one individual is known, one can read the public ledger to discover the precise financial situation of said user. So if this user relies on cryptocurrency as its sole financial platform, she would probably disclose her salary, bank account balance, or any transaction conducted at any given time. Such weak anonymity and privacy guarantees are not desirable for most people. Thus, one of the last remaining obstacles to the widespread adoption of cryptocurrencies is certainly privacy.

Of course, techniques, add-on solutions, or recommended behaviors exist to increase the base level of privacy. One such behavior is first for users to own multiple addresses and second to change them frequently. Yet, such a technique implies that users' wealth is scattered over several addresses. At some point, if a user wants to operate multiple of her addresses to conduct a single transaction, such a pattern is noticeable. Indeed, the original author of Bitcoin's white paper, Nakamoto, warns that linking addresses from multi-inputs transactions is still inevitable since they necessarily reveal that their inputs were owned by the same entity [40]. Several studies inspect the veracity of this claim [36, 37, 2, 48, 46, 38]. They find that many users who perform address changes have their addresses linked when merging their coins.

This linking is based on the assumption that multiple coins spent together have the same ownership. In Bitcoin, that assumption is not true by design. Various parties can combine their coins to send them to whomever they wish. That is the base principle of a Bitcoin add-on privacy solution, *Coinjoin* [34]. A more sophisticated mechanism exists in smart-contract-enabled blockchains such as Ethereum with Tornado Cash. It allows users to mix their cryptocurrency to obscure the origin of the funds. In that scenario, individuals deposit their coins and receive a note allowing them to withdraw their funds later. Thanks to zero-knowledge proof, the link between depositors and withdrawers is not revealed while ensuring full user custody of the funds.

Unfortunately, the opt-in nature of these tools does not increase the default privacy of the underlying chains. Also, the guarantees they provide are not absolute. There exist multiple ways to decrease the unlinkability properties of add-on privacy tools as shown by several studies [22, 25, 56]. This is also possible because, in such blockchains, no inherent mechanisms exist to hide the spending of coins. Thus, chain analysis can easily be conducted to trace coins back to their origin.

Some blockchain implementations overcome this shortcoming by concealing the author of transactions. For example, Zcash, a fork of Bitcoin's codebase, builds a so-called shielded pool where transactions are anonymized via zero-knowledge proofs [5]. In contrast, Monero expands on the CryptoNote protocol to hide the source of transactions with ring signatures [53]. Zcash and Monero are thus privacy-preserving cryptocurrencies, and they have attracted a lot of attention from the public and from research. However, despite extensive studies, no streamlined definition of blockchain privacy exists. Also, no unified framework is widely used to study and compare privacy guarantees.

This is the purpose of this work, *defining blockchain privacy notions*. For that, we extend a pre-existing model, the "Transaction Directed Acyclic Graph" (TDAG) defined by Cachin et al. [15], to capture additional properties such as the hiding of the transaction's source. Our first research question is a direct consequence of this desire. We first ask

"Is it possible to extend the TDAG to privacy-preserving blockchains?".

We introduce the original model and present our extensions in chapter 3. We also give a quick example of a Monero execution in the same chapter using our extended model, the PDAG. Chapter 5 goes into more detail and tackles formal executions of Monero, Zcash, and Tornado Cash. We continue and consider defining privacy notions using the PDAG. That introduces the second question as follows.

"Can we define privacy notions using the transaction graph model?"

That question occupies the whole chapter 4 in which we give formal definitions to *untraceability*, *unlinkability*, and *confidentiality* using PDAG's components. Untraceability is the property that, for a transaction, any sender is equiprobable. We define unlinkability as the inability to link addresses to the same entity. As for confidentiality, it is the property that the transacted data remains hidden. We conclude this very same chapter with an informal discussion about the relation between the three notions. We argue that confidentiality solidifies both untraceability and unlinkability and that untraceability allows for better unlinkability. Once defined, we wish to compare these notions for each system. We address this with our last research question.

"Does our model allow the comparison of the different blockchain systems in terms of privacy guarantees?"

We answer this question partially while defining the different notions in chapter 4 but effectively compare the different systems at the end of chapter 5. Consequently, we argue that, *according to our definitions*, Bitcoin itself, as well as Zcash transparent pool, does not provide any privacy notion. With Coinjoin and address change, Bitcoin may achieve unlinkability but only partially. The same goes for Tornado Cash which still suffers from specific linkability analysis [56]. Now Zcash's shielded pool provides pretty strong privacy properties, full untraceability, unlinkability, and confidentiality. Yet, some attacks on Zcash [45, 26, 7] prove that unlinkability is only achieved partially when the transparent pool interacts with the shielded one. Finally, Monero provides strong unlinkability guarantees and confidentiality as well. However, when it comes to untraceability, it lies behind Zcash's because of its limited traceability set.

Answering the three aforementioned questions enables us to provide the following contributions to the field. (1) We first successfully extend the TDAG to capture executions of privacy-preserving blockchains. (2) Then, we provide a unified definition for the different blockchain privacy notions. (3) Also, based on the literature, we give conditions for achieving each notion. (4) This allows us to compare the different blockchain systems in terms of privacy guarantees. (5) Furthermore, thanks to our notation, we can unify analysis patterns found in the literature – so-called linking heuristics – that are usually used to link addresses to the same entity. (6) Finally, we review and summarise a good part of the literature on the topic. The next chapter briefly introduces related works; otherwise, the thesis's structure is as given above.

Chapter 2

Related Work

This chapter presents existing works that we consider related to this thesis. For that, we focus on two different axes. Firstly, we introduce works that tackle one or multiple privacy notions in blockchains and, secondly, research that analyses the blockchain semantics using a graph-based model.

First, it is essential to mention that privacy notions in the existing literature do not follow any standards. Androulaki et al. [2], for instance, study Bitcoin privacy based on two different aspects, namely *activity unlinkability*, and *profile indistinguishably*. The first refers to the inability to link multiple addresses or transactions to one chosen user. The latter instead addresses the inability of the adversary to cluster addresses of Bitcoin users or – to use the words of the authors – reconstruct user profiles. In their work, Ober et al. [42] use the same approach to study anonymity using a transaction graph model. In contrast, other works define *unlinkability* as the anonymity of the receiver as opposed to *untraceability*, the anonymity of sender [57, 53]. Also, there seems to be some confusion between both unlinkability and untraceability, as the terms are sometimes used interchangeably or one to mean the other. Making the distinction might be challenging, considering the similarities between the two words in everyday speech. Some authors have even updated earlier works[1] to cope with agreed-upon definitions [44]. However, we observe that the literature slowly reaches a consensus regarding the definitions of both unlinkability and untraceability. We attribute this convergence to the emergence of privacy-preserving cryptocurrencies (e.g., Monero and Zcash) and intensive research on the privacy protection they aim to provide. In chapter 4, we give consistent definitions for both untraceability and unlinkability. We now describe related works individually for the different notions.

Regarding untraceability, Möser et al. [39] conduct an empirical analysis in the Monero blockchain. The authors identify two phenomena that jeopardize the untraceability guarantees of Monero. This blockchain allows users to obscure the transaction graph by including decoy inputs called "mixins" as part of the ring signature necessary to spend a coin. However, some users do not use any mixins in their transactions. Möser et al. [39] notice that these so-called 0-mixin transactions not only imply full traceability of such users, but also pose a privacy risk for other users that include poorly mixed outputs as mixins in subsequent transactions. Fortunately, since October 2018, the mixins input number is set globally to eleven [50], consequently fixing this issue. Yet, the authors [39] make a second observation regarding the sampling distribution of mixins inputs. The distribution of the sampling algorithm to choose the different mixins does not match the distribution of the coins spent in the transaction. The sampling distribution is uniform over the set of available transaction outputs, irrespective of the age of coins, whereas new coins are more likely to be spent directly. Old mixin coins can thus be considered decoys, reducing the true spender's anonymity set. Following this study, the Monero community also addressed this issue with a better sampling algorithm [20]. In a closely related work, Vijayakumaran [55] outlines the Dulmage-Mendelsohn decomposition as a mean to deanonymize users as part of multiple ring signatures. Egger et al. [19] use that decomposition to address untraceability and define theoretical foundations to this notion [19]. The authors describe an adversary that analyses the transaction graph to acknowledge the actual initiator of transactions. A bipartite graph describes which user is part of which ring, and its Dulmage-Mendelsohn decomposition enables an adversary to compromise untraceability

[1] We here refer to "An Empirical Analysis of Traceability in the Monero Blockchain", previously named "An Empirical Analysis of Linkability in the Monero Blockchain".

in some cases. In section 4.3, we generalize this concept to input to a transaction rather than a member of a ring signature.

As for unlinkability, various works devise empirical behavioral patterns in transactions – called heuristics – to link addresses together. Nakamoto [40] describes the most widely studied heuristic in the original Bitcoin white paper. This heuristic allows linking to the same entity, multiple addresses used as input to the same transaction [40]. Studies find it to be relatively accurate [2] and able to reveal hidden clusters of addresses [36]. The Coinjoin mechanism mitigates the threat posed by this heuristic by having different users join their coins into a single transaction to blur the links between several of their addresses. Yet, Goldfeder et al. [22] devise a counter-heuristic to detect such a technique by considering multi-input-multi-output transactions as part of a Coinjoin mixing. Victor [54] devises heuristics for specific patterns in the Ethereum and Wang et al. [56] for cases where users utilize mixers to break links between their transactions. Finally, Kappos et al. [26] and Quesnelle [45] expose heuristics specifically for Zcash. We present these heuristics in-depth and unify them with our model in section 5.5.

Another closely related notion is fungibility, the characteristic of every currency unit being identical. This notion is, in a sense, the property that each coin has the exact same identity (thus no identity) [6]. Although not exactly identical to untraceability or unlinkability, fungibility is the inability to connect cryptocurrency units with previous exchanges and is thus tied to those two notions. Knowing the true source of each transaction would allow tracing any coin back and jeopardize fungibility, but linking addresses together would have a similar effect. It is now widely accepted that Bitcoin is not fungible [49, 16], but other cryptocurrencies such as Monero and Zcash claim to provide such a feature. Yet, several studies show that Zcash as a whole, do not offer perfect fungibility due to the transparent pool and the possible inference with the shielded pool [26, 45, 7, 8]. Furthermore, although Monero fixed the above-mentioned issues, poorly mixed coins remain in the blockchain history, compromising total fungibility. Kelen and Seres [28] propose a graph-based framework to measure the fungibility and anonymity of cryptocurrencies [28]. They can quantitatively argue on the mixing characteristics of several cryptocurrencies. However, their framework does not allow them to analyze systems where untraceability is enforced, such as Monero and Zcash's shielded pool.

Concerning graph-based analysis of privacy, Ober et al. [42] conduct an empirical study of crucial properties of the Bitcoin transaction graph, such as unlinkability and coin dormancy. They use the multi-input heuristic mentioned above to merge addresses that belong to the same entity. As a result, they discover an impressive amount of public keys that belong together while only considering a small subset of the full blockchain history. Androulaki et al. [2] conduct a similar study through a simulator that mimics the user of Bitcoin within a university. In this setting, the authors [2] are in possession of the ground truth and are thus able to verify the results deviated from said heuristic. They find that almost 40% of user profiles can be recovered even though recommended privacy measures are applied.

On a more theoretical note, Cachin et al. [15] devise a transaction graph model to study blockchain semantics [15]. We use this model as a baseline for this work and thus describe it at length in chapter 3. Also, Atzei et al. [4] develop a model to formally prove the fulfillment of some properties in the Bitcoin blockchain [4]. Yet, their model only applies to Bitcoin and would probably induce major modifications for extending it to other blockchains. Conversely, Amarasinghe et al. [1] employ a common, universal framework to characterize the various aspects of anonymity provided by different blockchain implementations [1]. Their model is based on the adversary's knowledge and capacity, and they can compare several privacy-preserving blockchains using this framework. Finally, our work is also related to any research on privacy notions in anonymous communication networks. Kuhn et al. [30] provide an in-depth analysis on the matter [30]. They study privacy notions in anonymous communication and their relations and devise a formal hierarchy to classify each of these properties. Moreover, Henry and Goldberg [24] aim at formalizing anonymous blacklisting systems. They list the privacy and security properties that such systems should provide. In addition, they unify a set of definitions from the literature and propose a streamlined set of trust assumptions.

This is one of the purposes of this thesis, in which we aim to unify properties and definitions from the literature with a formal model. In the next chapter, we introduce such a model, the transaction directed acyclic graph.

Chapter 3

Transaction Graph

This chapter aims at introducing the *Transaction Directed Acyclic Graph* (TDAG) as proposed by Cachin et al. [15], with some extensions to capture privacy-preserving cryptocurrency transactions. We start by introducing the original definition and give motivation as to why an extension is needed.

3.1 Background

A transaction graph or TDAG is a directed acyclic graph $G = (\mathcal{V}, \mathcal{E})$ where the vertices are composed of states $\mathcal{S}$ and witnesses $\mathcal{W}$ such that $\mathcal{V} = \mathcal{S} \cup \mathcal{W}$. The edges, in turn, represent transitions between states and witnesses. In other words, an edge $e \in \mathcal{E}$ represents the transactional relationship between a state and a witness. Edges are partitioned into three different types, namely consuming, observing, and producing edges denoted respectively by, $\mathcal{E}_C$, $\mathcal{E}_O$, and $\mathcal{E}_P$. As in the original paper [15], we give informal explanations for those elements in the following. We subsequently motivate the need for an extension.

- **States**. A state $s \in \mathcal{S}$ is the first type of vertex denoting the output state of a transaction and is depicted by a circle $\bigcirc$. It refers to an individual asset on the blockchain, a digital coin, a coin controlled by a particular address, the state of a smart contract, etc. A state is the result of a transaction and can transition to other ones from a subsequent transaction. A unique state, the genesis state $s_g \in \mathcal{S}$, represents the initial state of the blockchain. There is only one genesis state since the blockchain system can only be bootstrapped once.
- **Witnesses**. A witness $w \in \mathcal{W}$ is the second kind of vertex, representing any data in a transaction for it to be valid according to the blockchain rules. It is depicted as a rectangle $\square$. Every transaction on the blockchain contains exactly one witness.
- **Consuming edges**. A consuming edge $e \in \mathcal{E}_C$ connects a state to a witness (i.e., depicted by $\bigcirc \longrightarrow \square$) and expresses that a state s is consumed by the transaction that involves a witness w.
- **Producing edges**. A producing edge $e \in \mathcal{E}_P$ connects a witness to a state (i.e., depicted by $\square \longrightarrow \bigcirc$) and expresses that the state is created by the transaction corresponding to the witness.
- **Observing edges**. An observing edge $e \in \mathcal{E}_O$ connects a state to a witness (i.e., depicted by $\bigcirc \dashrightarrow \square$) and express that the state enters into a transaction represented by the witness but that it stays available to another transaction for consumption.

Cachin et al. also give conflict-freedom conditions to a TDAG. Conflicts in a blockchain underlying a cryptocurrency occur in case of a double-spending tentative. Such a situation occurs if the same unique asset is spent more than once. In standard UTXO-based blockchains (e.g., Bitcoin), the system tracks unspent transaction outputs (UTXO) to determine which transactions may be spent. On privacy-preserving chains, the UTXO set is built differently so as not to disclose any information to its users. For example, in Monero, each transaction generates a unique key image, and key images used more than once are rejected by the blockchain [29]. A transaction is, therefore, valid if a key image does not conflict with an existing one. Zcash's shielded equivalent of a UTXO is a "commitment" [11]. A node can, in turn, spend a commitment by revealing a "nullifier".

3.1.1 Extension for masking

Regarding the TDAG, conflict-freedom is expressed by having every state produced and consumed at most once, i.e., coins cannot be spent twice, nor can they be duplicated. Now, the same rule applies to privacy-preserving blockchains. Yet, state consumption (coin spending) is not directly visible to the network outside the sender and receiver. Say, for example, that Alice uses ring signatures with ten other keys for her to remain unlinkable. The edge from these ten keys is an observing edge since we do not alter their states but only read them. Yet, Alice's spent coins are effectively consumed but appear only as being observed for non-involved parties. That introduces our first extension, a masking edge $\mathcal{E}_M$.

- **Masking edges**. A masking edge $e \in \mathcal{E}_M$ connects a state to a witness (i.e., depicted by $\bigcirc\!\rightsquigarrow\!\square$) and is a consuming edge that hides behind cryptographic mechanisms. More precisely, it appears as an observing edge to nodes aside from the author but is a consuming edge. As its name suggests, it mask its actual behavior. It otherwise follows the same rules as producing edge.

3.1.2 Extension for commitments

Furthermore, as mentioned above, privacy-preserving blockchains output additional components at each transaction. These components allow to construct a cryptographic mapping with unspent transactions. This allows for preventing double-spending while hiding the source of a transaction.

To be more general, any hidden behavior in the blockchain requires users to output a specific value as a *commitment*. In addition, the exact same behavior produces the same value. Therefore, this redundancy is visible if users try to benefit from the same asset or service multiple times. The blockchain validation rules will thus forbid one such transaction since two values collide. This implies that each transaction must verify that the new values do not conflict with pre-existing ones. In turn the total set of generated values cannot be discarded at any point since it must be read for each transaction to prevent conflicts.

We, therefore, introduce this additional component as a state that can only be observed or produced. Depending on the implementation, this component may have different names, e.g., key images for Monero, nullifiers for Zcash, or withdrawal commitments for Tornado Cash. Without loss of generality, we adopt the name of *nullifiers* for this component.

- **Nullifiers**. A nullifier $n \in \mathcal{N}$ is a state that encapsulates cryptographic commitments which prevent users from benefiting from the same asset or service multiple times. We depict it as a diamond $\Diamond$. It may only be produced or observed.

Those two additions lead us to call this extension the "nullifier-based" model, as opposed to the "UTXO-based" (e.g., Bitcoin) or "account-based" model (e.g., Ethereum). Yet, those extensions do not impact pre-existing models, and the definitions and executions given in the original paper remain valid. We formalize the components in the next section and show that it does not invalidate UTXO and account-based models.

3.2 Privacy-preserving TDAG

This section defines our extended model, called the privacy-preserving TDAG (PDAG). We mark the extensions to the original definitions in $\boxed{\text{frame}}$. We begin by specifying a single transaction that consists of one witness and multiple input and output states. The witness reads the full set of nullifiers in the blockchain execution and outputs a single one.

Definition 3.1 (Transaction)**.** A weakly connected DAG $T = (\mathcal{V}, \mathcal{E})$ with a set of input states $\mathcal{S}_I$, a set of output states $\mathcal{S}_O$, $\boxed{\text{the full set of nullifiers } \mathcal{N}}$, and a witness w is called a *transaction* whenever

- every input state in $\mathcal{S}_I$ is a source (has indegree zero);
- every output state in $\mathcal{S}_O$ is a sink (has outdegree zero);

- $\mathcal{V} = \mathcal{S}_I \cup \mathcal{S}_O \boxed{\cup \mathcal{N}} \cup \{w\}$;
- Every edge e in $\mathcal{E}$ is either a consuming edge e_C, $\boxed{\text{a masking edge } e_H}$, or an observing edge e_O and links some input state $s_i \in \mathcal{S}_I$ to w, or it is a producing edge and links w to some output state $s_o \in \mathcal{S}_O$.

As depicted in figure 3.1, we can see a transaction taking place, where the author of the transactions uses a ring signature with her private key s_5 and ten decoy keys s_0 to s_{10}. All the keys appear to be observed to others, as s_5 is consumed. Thus, a masking edge is used. Conversely, the other states are decoy keys linked to w with an observing edge. The full set of nullifiers n_0 to n_k are observed to check for collision with the generated ones. Two states get produced, the transferred coins s_{11} and the new nullifier n_{k+1}.

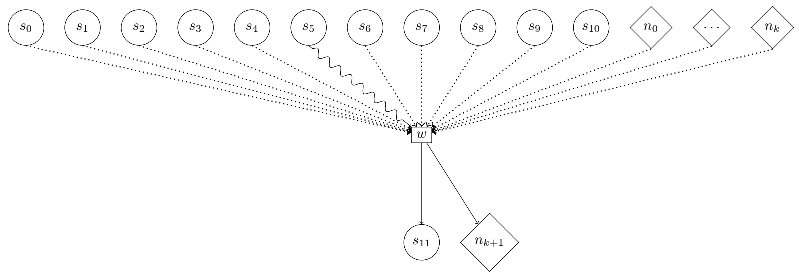


Figure 3.1: Illustrative example of a Monero transaction. Eleven keys are used for the ring signature to provide untraceability for the author of the transaction s_5, which has thus a masking edge, whereas other states have an observing edge. The nullifier state n_{k+1} contains the key images produced previously and taken as input $\{n_0, \ldots, n_k\}$ and the one produced with this transaction.

As the name suggests, a transaction graph contains several transactions. We thus formally define the PDAG below. We use the notation $\mathcal{E} \subseteq X \times Y$ to denote a relation $\mathcal{E}$ between X and Y. Also, the predicate $(x, y) \in \mathcal{E}$ can be written as $x\mathcal{E}y$ and we define the set $\{y : x\mathcal{E}y\}$ by $x\mathcal{E}\star$ and its cardinality by $|x\mathcal{E}\star|$.

Definition 3.2 (PDAG)**.** A *privacy-preserving transaction graph (PDAG)* is a directed unweighted graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{S} \cup \mathcal{N} \cup \mathcal{W}$ are the vertices and $\mathcal{E} = \mathcal{E}_C \cup \mathcal{E}_M \cup \mathcal{E}_O \cup \mathcal{E}_P$ are the edges. The set $\mathcal{S}$ denotes the states and contains a special state s_g called genesis. $\boxed{\text{The nullifiers } \mathcal{N} \text{ are states that encapsulate cryptographic commitments.}}$ The set $\mathcal{W}$ denotes the witnesses. Edges are partitioned into four subsets where $\mathcal{E}_C \subseteq \mathcal{S} \times \mathcal{W}$, $\mathcal{E}_M \subseteq \mathcal{S} \times \mathcal{W}$, $\mathcal{E}_O \subseteq \mathcal{S} \times \mathcal{W}$, and $\mathcal{E}_P \subseteq \mathcal{W} \times \mathcal{S}$. It satisfies the following conditions:

1. s_g does not have any producing or observing edges, and it has a single consuming edge, i.e., $|\star\mathcal{E}_P s_g| = 0 \wedge |s_g\mathcal{E}_P\star| = 0 \wedge \exists! w \in \mathcal{W} : s_g\mathcal{E}_C w$.
2. Every state, except for the genesis state, has exactly one producing edge, i.e., $\forall s \in \mathcal{S} \setminus \{s_g\}\ \exists! w \in \mathcal{W} : w\mathcal{E}_P s$.
3. Every state, except for the genesis state, may have multiple successors, but at most one among them is connected with a consuming $\boxed{\text{or masking}}$ edge, i.e., $\forall s \in \mathcal{S} : \left| s\mathcal{E}_C \star \boxed{\cup s\mathcal{E}_M\star} \right| \leq 1$.
4. $\boxed{\text{Nullifiers } n \in \mathcal{N} \text{ have no consuming edges and no masking edges, i.e., } \forall n \in \mathcal{N}\ |n\mathcal{E}_C\star| = 0 \wedge |n\mathcal{E}_M\star| = 0.}$
5. G is weakly connected.
6. G has no cycles.

It stems from this that for each witness w, there is a corresponding transaction t. Therefore, the following definition is equivalent to 3.1, again derived from the original model and extended to our needs.

Definition 3.3 (Transaction in a PDAG)**.** Given a TDAG $G = (\mathcal{S} \cup \mathcal{N} \cup \mathcal{W}, \mathcal{E})$ and a witness $w \in \mathcal{W}$, the *transaction* with witness w is the unique subgraph $t = \left(\mathcal{S}' \boxed{\cup \mathcal{N}} \cup \{w\}, \mathcal{E}'\right) \subset G$, where

- $\mathcal{S}'$ is the set of states connected to w, i.e., $\mathcal{S} = \left\{ s \in \mathcal{S} : s\mathcal{E}_C w \boxed{\vee s\mathcal{E}_M w} \vee s\mathcal{E}_O w \vee w\mathcal{E}_P s \right\}$;
- $\boxed{\mathcal{N} \text{ must be the full set of nullifiers, i.e., } \mathcal{N} = \{\forall w' \in \mathcal{W}, n \in \mathcal{N} : w'\mathcal{E}_P n \vee w\mathcal{E}_P n\};}$
- $w \in \mathcal{W}$ is the witness of the transactions; and

- $\mathcal{E}'$ are the edges with both endpoints in $\mathcal{S}' \cup \{w\}$.

The extensions do not conflict with the initial model since they are optional. Masking edges are inexistent in blockchains where the state consumption is visible, and nullifiers are not produced in such cases either. We here postulate that an original TDAG is also a PDAG.

Lemma 3.1. If $G_{\text{TDAG}} = (\mathcal{S}_{\text{TDAG}} \cup \mathcal{W}_{\text{TDAG}}, \mathcal{E}_{\text{TDAG}})$ is an original TDAG per [15], then G is also a PDAG $G_{\text{PDAG}} = (\mathcal{S}_{\text{PDAG}} \cup \mathcal{N}_{\text{PDAG}} \cup \mathcal{W}_{\text{PDAG}}, \mathcal{E}_{\text{PDAG}})$.

Proof. We need to prove that

1. a PDAG is a superset of an original TDAG and
2. that the original TDAG follows the definition of a PDAG.

We address those two points below.

1. By construction, we have that $\mathcal{V}_{\text{TDAG}} \subseteq \mathcal{V}_{\text{PDAG}}$ and $\mathcal{E}_{\text{TDAG}} \subseteq \mathcal{E}_{\text{PDAG}}$, so G_{PDAG} is a superset of G_{TDAG}.
2. Now, we need to consider the two additional points in the extended definition.
   (a) The statement that each state is consumed at most once, either by a consuming or by a masking edge, i.e., $\forall s \in \mathcal{S} : |s\mathcal{E}_C \star \cup s\mathcal{E}_M \star| \leq 1$, is still valid since an original TDAG does not have a masking edge and thus each state is consumed at most once by one consuming edge.
   (b) As for the condition that nullifiers have no consuming or masking edges, i.e., $\forall n \in \mathcal{N}\, |n\mathcal{E}_C \star| = 0 \wedge |n\mathcal{E}_M \star| = 0$, it is also valid, because an original TDAG has no nullifiers and thus this condition is de facto fulfilled by the trivial case.

   An original TDAG, therefore, follows the extended definition.

An original TDAG is thus also a PDAG. □

It also results from this lemma that if the graph G modeling an execution $\mathcal{L}$ is an original TDAG, then it is also a PDAG. This is particularly useful as it allows us to keep the original executions defined by Cachin et al. and even use it as part of other blockchain systems in chapter 5. We here recall that, in a TDAG, transactions t may be of multiple types, depending on the number of inputs ad outputs.

- INIT: An initialization transaction that represents the creation of the blockchain with the genesis block. It consists of a consuming edge that links the genesis state to a witness and a set of producing edges that link the witness to a set of states.
- SISO: A single-input, single-output transaction with one consuming edge that links an input state to a witness and one producing edge that links the witness to an output state.
- SIMO: A single-input, multi-output transaction with one consuming edge that links an input state to a witness and a set of producing edges that link the witness to a set of output states.
- MISO: A multi-input, single-output transaction with multiple consuming and observing edges that link distinct input states to a witness and one producing edge that links the witness to an output state.
- MIMO: A multi-input, multi-output transaction with multiple consuming and observing edges that link distinct input states to a witness and a set of producing edges that link the witness to a set of output states.

The initialization transaction represents the creation of the blockchain with the genesis block s_g. The other types may occur multiple times over the execution of the blockchain system. An interesting observation is that SISO and SIMO transactions de facto consume a single input. For SIMO, that means that a masking edge has no impact on privacy; we still represent it as a masking edge to outline the use of cryptographic mechanisms. MISO transactions may only occur in plain since otherwise, a nullifier would also be produced. MIMO is the most common type of transaction on privacy-preserving blockchains. The multiple inputs hide the true source of the transaction, and the various outputs are justified by the new states and the nullifiers.

One simple execution of Monero is represented in figure 3.2 inspired by the one of Bitcoin described in the original paper. As opposed to Bitcoin, however, there is no pre-mine; the total coin supply is not capped but increases indefinitely. We call such a paradigm inflationary.

The execution begins with the first transaction, in which the witness w_1 consumes the genesis state s_g and gives the first mining reward ($\sim$ 17.6) to one account, Alice[1], represented by state s_A. Alice then disposes of her coins and sends 2 XMR to Bob with a fee of 0.1. She uses a ring signature, thus the masking edge. Alice receives the remaining (15.6) and Bob 2.0 minus the fee, i.e., 1.9. The states s'_A, s_B, and s_1 correspond respectively to Alice, Bob, and the fee. The third transaction with witness w_3, occurs with Alice sending coins to Carol. She uses once again ring signature, but this time with Bob's stealth address. Therefore, s'_A has a masking edge (real author) as s_B has only an observing key (decoy key). The fee s_2, Alice's and Carol's new coin balance, respectively s''_A and s_C, get produced as part of this transaction. Finally, the fees s_1 and s_2 get consumed by the mining process w_4, and the miner state s_M gets produced with the fees 0.3 and the mining reward $\sim$ 17.6 i.e., 17.9. During the second and third transactions, nullifiers get produced as a consequence of the ring signatures. They contain the key images that prevent double-spending. Nullifier n_1 gets an observing edge to w_3 to compare the key image of w_3 to the one produced previously with w_2. The transaction cannot go through if they match because of a double-spending conflict.

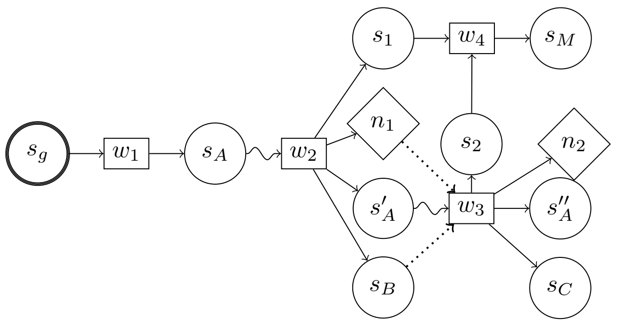


Figure 3.2: Illustrative example of a TDAG, where four transactions w_i, $i \in \{1, 2, 3, 4\}$ take place.

We formalize here the conflict-freedom definition.

Definition 3.4 (Conflict-freedom)**.** Consider a DAG $\mathcal{T} = (\mathcal{S}_T \cup \mathcal{W}_T, \mathcal{E}_T)$ with states $\mathcal{S}_T$, witnesses $\mathcal{W}_T$, producing edges $\mathcal{E}_P \subseteq \mathcal{E}_T$ and consuming edges $\mathcal{E}_C \subseteq \mathcal{E}_T$ that contains a transaction for every witness $w \in \mathcal{W}_T$. We say that $\mathcal{T}$ is *conflict-free* if every state has at most one producing or masking edge and one consuming edge, i.e., $\forall s \in \mathcal{S}_T : |\star\mathcal{E}_P s| \leq 1 \wedge |s\mathcal{E}_C\star| + |s\mathcal{E}_M\star| \leq 1$.

Conflict freedom is an essential property of the PDAG as it demonstrates that an asset has not been spent or paid out more than once. However, blockchains also possess a set of rules that must be met for each transaction. If a transaction is malformed, other nodes will reject it. Thus, we introduce the notion of validity from the original paper [15], which models that only valid transactions are executed.

Definition 3.5 (Validity [15])**.** Let t be a transaction in a PDAG G. Then t is *valid* whenever $\mathbb{P}(t) = \texttt{true}$. Furthermore, G is a valid transaction graph if all transactions in G are valid. [15]

At this point, the properties given by the definitions above do not provide much information on the privacy guarantees offered by the underlying blockchain. This is mainly the object of the next chapter, where we formalize the different privacy notions at hand. For that purpose, we also define a concept of size for the witnesses. The size or length of a witness is determined by the number of states it is linked to. The below definition states that formally.

Definition 3.6 (Size of a witness)**.** The *size of a witness* w denoted by $|w|$ is defined as the number of states it is linked to, i.e., $|w| = |\{s \in \mathcal{S} \mid \exists e$ such as $e = (w, s)\}|$.

This definition follows the specifications of $\min$ and $\max$ functions of $\mathcal{W}$, i.e., the smallest and largest w in $\mathcal{W}$ according to size.

The witness's ordering property is beneficial when an adversary tries to guess which edge is a masking one. She has more chance to guess right, taking the minimum element of $\mathcal{W}$. This strategy is further explained in the next chapter under the untraceability notion.

[1] In reality, the account `thankful_for_today` received the first reward.

Chapter 4

Privacy Notions

This chapter formalizes the different privacy notions in blockchain systems with respect to PDAG outlined in chapter 3. We first give an informal overview of the properties and the adversary at hand. We secondly, define the adversary's capacities and the game that is used thereafter to formalize the different privacy notions.

4.1 Overview

We start here by giving an example scenario that implicitly showcases the different privacy notions.

Example. A repressive regime has full control over the regular banking system. It uses this control – among other means – to establish a mass surveillance scheme and if necessary financial coercion. It may even forcefully prohibit cash payments. Any resistance group is therefore very quickly deprived of financial means when the state deems it so. One of the few ways for a resistance group to thrive is to use a decentralised cryptocurrency. Yet, this repressive state scans and analyses the blockchain history in order to identify

(1) its main creditors and debtors,

(2) addresses of the resistance group,

(3) the funds at its disposal.

Alice regularly gives financial support to this resistance group. If she is suspected, she risks being prosecuted or even worse.

This example implicitly demonstrates three privacy notions. The first one is *untraceability* where the true author of one transaction is not trivially visible to an external observer. Second, *unlinkability* is achieved when multiple addresses cannot be tied together to the same entity. Finally, *confidentiality* represents the property that the transacted amount remains hidden from non-involved parties. The goal of the following sections is to formally define those three properties.

The game that we describe here consists in the adversary constructing the transaction graph from reading the blockchain system. The challenger, in turn, provides a puzzle to the adversary. To do that, she may apply any polynomial-time computationally bounded algorithm to output a guess as to jeopardize one property. For each of untraceability and unlinkability, we define a separate game in which the adversary tries to compromise the notion at hand by extracting knowledge from the transaction graph. As for confidentiality, we give a more standard cryptographic definition. This entire work furthermore assumes all the other layers of the blockchain system to be secure. The object of the next section formalizes the adversary capacities on the transactional level.

4.2 Adversarial Model

In this section, we introduce the adversary capability and knowledge with respect to the PDAG defined in chapter 3. The words “states”, “witnesses” and “edges” are to be understood accordingly.

We use Π to denote the analyzed blockchain system, Ch for the challenger and $\mathcal{A}$ for the adversary. $\mathcal{A}$ is a probabilistic polynomial time (PPT) algorithm that has full access to blockchain history. We focus the power of the adversary on the transaction layer and assume the adversary cannot temper with the consensus mechanism [43]. As for the game, it consists in the adversary making a guess according to a property X as in modern cryptography. This property may either be one of untraceability or unlinkability. For confidentiality, we give a more generic definition. Furthermore, we assume that the adversary is able to construct the transaction graph G from the blockchain history in polynomial time with a security parameter of $1^{|G|}$. The game has the following broad structure.

1. The experiment begins when challenger Ch is given a PDAG G.
2. Ch may query an oracle to get a construct Ω that contains solutions for guesses of the adversary.
3. The challenger sends back Ω.
4. $\mathcal{A}$ runs with G as input and outputs a guess.
5. The adversary wins the game if the guess is in Ω.

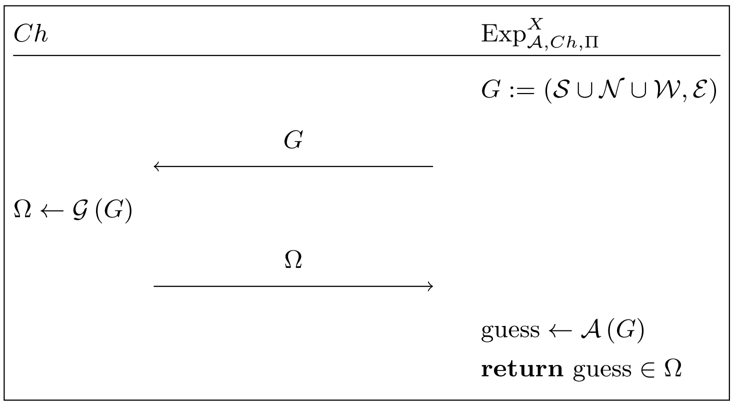

Ch
$\mathrm{Exp}^{X}_{\mathcal{A},Ch,\Pi}$
$G := (\mathcal{S} \cup \mathcal{N} \cup \mathcal{W}, \mathcal{E})$
G
$\Omega \leftarrow \mathcal{G}(G)$
Ω
guess $\leftarrow \mathcal{A}(G)$
return guess $\in \Omega$

Figure 4.1: A broad structure of the game for notion X.

This structure, illustrated in figure 4.1, is the general frame to define the notions of untraceability and unlinkability. Intuitively, Π achieves notion X if any possible adversary has a bounded chance of winning the game. The concepts X are further introduced in the following sections, and we give each of these a condition for their fulfillment. For each of untraceability and unlinkability, we adapt the game in specifying the content of Ω and the guess of $\mathcal{A}$. For confidentiality, we settle on a generic cryptographic definition. Finally, we try to define or at least narrow down the probability of $\mathcal{A}$ to compromise each notion to compare the different blockchain systems in chapter 5. The following section tackles the first privacy notion, namely untraceability.

4.3 Untraceability

Informally, untraceability means that for each incoming transaction, all possible senders are equiprobable. This property may stem from different mechanisms depending on the blockchain implementation: ring signatures or one-out-n proof. For now, we only describe ring signature as part as the reduction that we consider in this section. We describe other mechanisms in chapter 5. About a PDAG, however, the principle remains the same for whichever protocols. Multiple input states are linked to the witness by an observing edge and one with a masking edge. From the adversary’s view, the masking edge appears as an observing one; thus, any state is equiprobable to be consumed or observed. This, therefore, obfuscates the true origin of the transaction. This problem is called a *traceability* problem from the adversary’s view. The purpose of this game is to guess which of the observing edges is, in fact, a masking one and hence which of the input states is consumed.

In other words, for an adversary $\mathcal{A}$ and a transaction graph $T = (\mathcal{S} \cup \mathcal{W}, E_O \cup E_M \cup \mathcal{E}_C)$, the goal of $\mathcal{A}$ is to output a guess as to which edge e is in $\mathcal{E}_M$ and thus which state $s \in \star\mathcal{E}w$ is consumed. Throughout multiple transactions, $\mathcal{A}$ has to map the input states to possible transaction sources, resulting in a bipartite graph matching as illustrated in figure 4.2.

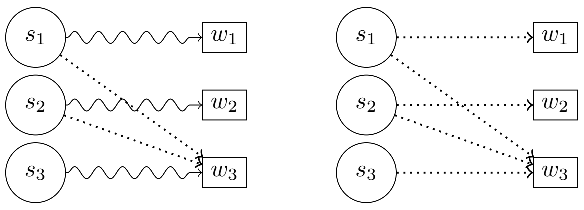


Figure 4.2: Three transactions w_1, w_2 and w_3 have a total of three input states s_1, s_2 and s_3. On the *left* is the graph from the view of the oracle, and on the *right* from the adversary. The masking edges highlight the true matching that $\mathcal{A}$ must guess.

We argue and prove that the traceability problem concerning the PDAG reduces to the graph-based deanonymisation which Egger et al. [19] describe in their work. This attack takes place in Monero, in which ring signatures promote untraceability. A ring signature is a cryptographic primitive which consists of a ring and a signature. The first is a set of public keys, one of which belongs to the signer. The latter is computed with the ring and the signer's private key. A valid signature guarantees that it comes from one of the ring members, but any verifier cannot tell which one of them is the signer. In Monero, when spending a coin, users sign the transaction with a ring signature of size eleven. This means that for an adversary, the ring members are visible but are equiprobable to be the actual initiator of the transaction. In their work, Egger et al. [19] describes an adversary that uses graph-based deanonymisation to discover the true signer of the ring. The authors model this problem as a bipartite graph with a set of users U and a set of rings R. For instance, in figure 4.3, each user is a member of multiple rings, and the adversary must discover which user is the signer of each ring. Consequently, the adversary must output one edge for each of the rings $r_i \in R$, a so-called maximum matching.

$$\begin{tikzcd}
U & R \\
u_1 \arrow[r] \arrow[rdd] & r_1 \\
u_2 \arrow[r] \arrow[rd] & r_2 \\
u_3 \arrow[r] & r_3
\end{tikzcd}$$

Figure 4.3: An illustration of the graph-based deanonymisation problem as described by Egger et al. [19].

This graph purposely resembles a lot the one shown in the preamble to give the intuition of the reduction. Therefore, there exists such a reduction from the traceability problem to the graph-based deanonymisation described by Egger et al. [19].

For instance, consider a PDAG $G = (\mathcal{S} \cup \mathcal{N} \cup \mathcal{W}, \mathcal{E})$. For all witnesses that possess an incoming edge $w \in \mathcal{W}$, let us take all input states $s \in \star\mathcal{E}w = \mathcal{S}_I$. Those input states S_I and witnesses W form disjoint and independent sets. That is, by the construction of the PDAG, every edge connects a vertex in $\mathcal{S}_I$ to one in $\mathcal{W}$. This method allows the obtention of a bipartite graph. However, the maximum matching does not necessarily represent the full state consumption. If a witness consumes more than one state, one of the two consuming or masking edges of that witness is not included in the maximum matching. One construction that alleviates this issue is to consider consuming and masking edges directly. We construct a set of such edges and consider any adjacent witness. If a witness appears twice, it gets duplicated to preserve the property of the maximum matching with regard to state consumption. We postulate in the following that the traceability problem thus reduces to the graph-based deanonymisation problem. The proof follows the above intuition.

Lemma 4.1 (Traceability reduction). The traceability problem reduces to the graph-based deanonymisation problem.

Proof. Let $\mathbf{A}$ be the traceability problem and $\mathbf{B}$ the graph-based deanonymisation problem. Then, we aim to transform the instance of problem $\mathbf{A}$ into an example of problem $\mathbf{B}$. For that, we construct an algorithm $\mathbf{flatten}$ that takes a PDAG G as input and outputs a bipartite graph G^*. Before that, we recall essential properties of the PDAG that makes $\mathbf{flatten}$ a valid reduction.

(1) In a PDAG $G = (\mathcal{S} \cup \mathcal{N} \cup \mathcal{W}, \mathcal{E})$, states and witnesses are disjoint sets, i.e., $\mathcal{S} \cap \mathcal{W} = \emptyset$.

(2) The states linked with a consuming or a masking edge represent the true signers.

(3) A state can only be consumed once.

The algorithm $\mathbf{flatten}$ constructs the bipartite graph G^* by first building the set of consuming and masking edges. Second, it adds to a *list* any witness adjacent to such edge, potentially duplicating witnesses linked with multiple consuming or masking edges. Finally, the set of states is any input states connected to a witness in the list, and we add these connecting edges to the initial set of edges. We now expose the logic behind the algorithm $\mathbf{flatten}$.

```
flatten ($G$ : PDAG)
1 :  $G = (\mathcal{S} \cup \mathcal{N} \cup \mathcal{W}, \mathcal{E}_C \cup \mathcal{E}_M \cup \mathcal{E}_P \cup \mathcal{E}_O)$
2 :  $\mathcal{E}^* := \mathcal{E}_C \cup \mathcal{E}_M$
3 :  $W := [\perp]^{|\mathcal{E}^*|}$
4 :  $\mathcal{S}^* := \emptyset;\ \mathcal{E}^* := \emptyset$
5 :  foreach $(s^*, w^*) \in \mathcal{E}^*$ such that $w^*$ is not a mining transaction do
6 :      $W.\mathrm{add}(w^*)$
7 :      $\mathcal{S}^* = \mathcal{S}^* \cup \{s^*\}$
8 :  endforeach
9 :  $\mathcal{W}^* := W.\mathrm{toSet}()$    // converts list to set while renaming redundant items
10 : $\mathcal{E}^* = \mathcal{E}^* \cup \{(s^*, w^*) \mid (s^*, w^*) \in \mathcal{E}_O,\ s^* \in \mathcal{S}^*, w^* \in \mathcal{W}^*\}$
11 : return $G^* := (\mathcal{S}^*, \mathcal{W}^*, \mathcal{E}^*)$
```

It appears from the properties above and the construction of G^* that

- G^* is a bipartite graph since both the witnesses and states are disjoint and independent,
- the masking and consuming edges form a maximum matching since each witness has exactly one such edge and
- the maximum matching outlines the true signers.

Consequently, $\mathbf{flatten}$ reduces $\mathbf{A}$ to $\mathbf{B}$.

□

As we have proven a reduction from the traceability problem to the graph-based deanonymisation one, we speak about the generated bipartite graph as an inferred graph G^*. This implies that graph-based deanonymisation can be applied not only to ring protocols but to any protocol. That introduces the following corollary in which we postulate that the graph-based deanonymisation applies to any valid PDAG – and thus, regardless of the untraceability technique.

Corollary 4.1. If G is a valid PDAG that models a blockchain execution $\mathcal{L}$, the graph-based deanonymisation [19] applies to G.

Proof. By reduction 4.1, we transform G into a bipartite graph G^*. We apply the graph-based deanonymisation on this bipartite graph G^*. Therefore, this technique applies to G. □

With this reduction, we may furthermore derive some underlying properties. As in the work of Egger et al. [19], we observe in figure 4.2 that the maximum matching highlighted by the masking edges is unique. This (extreme) case leads to full traceability since we may devise the unique set of tuples $\mathcal{M} = \{(\textit{consumed state}, \textit{witness})\}$, which in this case is $\{(s_1, w_1), (s_2, w_2), (s_3, w_3)\}$. However, in most cases, the inferred bipartite graph has multiple maximum matchings, meaning traceability is not trivial. For that purpose, we define the core of G^* – in the sense of the Dulmage-Mendelsohn (DM) decomposition [18] – which corresponds to the union of all its possible maximum matchings.

Definition 4.1 (Core [19])**.** The *core* of a bipartite graph $G^* = (A, B, \mathcal{E})$, denoted by $\mathbf{Core}(G^*) = (A, B, \mathcal{E}')$, is a subgraph of G^* where $\mathcal{E}' \subseteq \mathcal{E}$ is the union of all maximum matchings in G^*.

From what we have seen above and from the original paper [19], we conclude that if $G \neq \mathbf{Core}(G^*)$, then some information can be deduced by $\mathcal{A}$ and untraceability is thus compromised. According to that condition, we define a *safety* property for the PDAG.

Definition 4.2 (PDAG's safety)**.** A PDAG G is considered *safe* if it holds for its inferred bipartite graph G^* that $G^* = \mathbf{Core}(G^*)$. When a PDAG G is safe, we define a predicate $\textbf{safe}(G)$.

Faced with a safe PDAG, the adversary cannot deduce further information. Her best strategy is to select the witness with the fewest states and output a random guess as to which state is consumed. This strategy thus bounds the adversary traceability capacities in the traceability game. For example, given a safe PDAG and the smallest witness having $k+1$ states linked to it, the traceability game is upper bounded by $\frac{1}{k+1}$.

The game that we define here, therefore, consists of an adversary $\mathcal{A}$ getting a transaction graph G and trying to identify any edge of the maximum matching M. This maximum matching corresponds to the true state consumption. Since this matching is part of the ground truth of the blockchain, we define a graph oracle that returns M accordingly.

Definition 4.3 (Tracing graph oracle)**.** A *tracing graph oracle* $\mathcal{G}_{\Pi}^{\text{Trace}}$ takes a transaction graph (PDAG) $G = (\mathcal{S} \cup \mathcal{W}, E_C \cup E_M \cup \mathcal{E}_O)$ of a blockchain system Π as input and outputs the maximum matching $M := \mathcal{E}_M \cup \mathcal{E}_C$ that corresponds to the true state consumption.

Being subject to the attack of the adversary, it is thus $Ch_{\mathcal{G},\Pi}$ under oracle access $\mathcal{G}(\cdot)$ which indeed defines the conditions of untraceability. We establish the game $\text{Exp}^{\text{Trace}}_{\mathcal{A},Ch_{\mathcal{G},\Pi},\Pi}$, outlined in figure 4.4, for adversary $\mathcal{A}$ using graph analysis algorithm under blockchain system Π. In the light of definition 4.2 and the explanation above, we already understand the upper bound of this game, given a safe PDAG. The probability for $\mathcal{A}$ to win the game given a safe graph is upper bound by the size of the smallest witness. We thus have for $k+1 = \min(\mathcal{W})$

$$\Pr\left[\text{Exp}^{\text{Trace}}_{\mathcal{A},Ch_{\mathcal{G},\Pi},\Pi}(G) \mid \textbf{safe}(G)\right] \leq \frac{1}{k+1}.$$

```
$Ch_{\mathcal{G},\Pi}(G)$
1 :  $M \leftarrow \mathcal{G}_{\Pi}^{\text{Trace}}(G)$
2 :  $M := \mathcal{E}_M \cup \mathcal{E}_C \subset \mathcal{E}$
3 :  $\textbf{return } M$
```

```
$\text{Exp}^{\text{Trace}}_{\mathcal{A},Ch,\Pi}(G)$
1 :  $G := (\mathcal{S} \cup \mathcal{N} \cup \mathcal{W}, \mathcal{E})$
2 :  $M \leftarrow Ch_{\mathcal{G},\Pi}$
3 :  $G^* = (\mathcal{S}^*, \mathcal{W}^*, \mathcal{E}^*) \leftarrow \textbf{flatten}(G)$
4 :  $(s^*, w^*) \leftarrow \mathcal{A}(G^*)$
5 :  $\textbf{return } (s^*, w^*) \in M$
```

Figure 4.4: Challenger and traceability game for a transaction graph G

More generally and irrespective of PDAG's safety, we define a specific bound $\epsilon \geq 0$ that defines the untraceability guarantees of Π.

Definition 4.4 (ϵ-untraceability)**.** The graph G modelling an execution of a blockchain system Π is *ϵ-untraceable* for some $\epsilon \geq 0$ if

$$\Pr\left[\text{Exp}^{\text{Trace}}_{\mathcal{A},\mathcal{G},\Pi}(G)\right] \leq \epsilon.$$

Trivially $\epsilon = \frac{1}{k+1}$, whenever G is safe. However, the transaction graph G is unsafe in many cases, as shown by Vijayakumaran [55]. This means the adversary is much more potent in tracing the different state consumptions. In turn, this depends on the probability of G not being safe, i.e., $\Pr\left[\overline{\mathbf{safe}}\left(G\right)\right]$. We take, once again, the results of Egger et al. [19] and adapt their lemma in the following way.

Lemma 4.2. Let $G = (\mathcal{S} \cup \mathcal{W}, \mathcal{E})$ be a transaction graph, $k \in \mathbb{N}$ for which $k + 1 = \min(\mathcal{W})$ holds, and $\mathcal{G}_{\Pi}^{\text{Trace}}(G)$ be the graph oracle for blockchain system Π. For an adversary $\mathcal{A}$,

$$\Pr\left[\text{Exp}_{\mathcal{A},Ch,\Pi}^{\text{Trace}}\right] \leq \Pr\left[\overline{\mathbf{safe}}\left(G\right)\right] + \frac{1}{k+1}$$

where the probabilities are taken over the randomness of $\mathcal{A}$ and $M \leftarrow Ch_{\mathcal{G},\Pi}(G)$.

Proof. The proof is derived from the lemma 7.1 of the work by Egger et al. [19] and the reduction per lemma 4.1. □

Although it is relatively straightforward to estimate k, the probability of G not being safe is unclear. Intuitively, this probability is higher if the bipartite graph is strongly connected, but this remains challenging to prove. To get an upper bound of $\Pr[\mathbf{safe}(G)]$, Egger et al. [19] make several assumptions on the distribution of graphs. They approximate the distribution of k-in-degree regular graphs with the one over random graphs with n vertices where each possible edge appears with probability p. In turn, the parameter p allows them to simulate the strong connectivity of regular graphs. They reach the target in-degree of k with a probability of $p = \frac{k}{n-1}$. Assuming the approximation of regular graphs by random ones hold, they give a closed-form upper bound to the safety of the graph. At this point, the authors cannot prove the upper bound but provide empirical evidence that seems to hold for parameters of interest [19].

They find that the challenger is ϵ-untraceable for $\epsilon = \frac{2}{k+1}$ and with $k \geq \ln(2|\mathcal{S}|) + \sqrt{2\ln(2|\mathcal{S}|)}$. We base ourselves on such results to define the achievement of untraceability.

Definition 4.5 (Achieving untraceability)**.** A blockchain system Π

- ● *achieves untraceability*, if said system is ϵ-secure against any graph-based deanonymisation adversary $\mathcal{A}$ for $0 \leq \epsilon \leq \frac{2}{k+1} = \mathcal{O}\left(\frac{1}{k}\right)$ with $k \geq \ln(2|\mathcal{S}|) + \sqrt{2\ln(2|\mathcal{S}|)}$ [19],
- ◑ *partially achieves untraceability*, if there exists an adversary $\mathcal{A}$ for which $\mathcal{O}\left(\frac{1}{k}\right) < \epsilon < 1$ for any value of k or
- ○ *does not achieve untraceability*, if there exists an adversary $\mathcal{A}$ for which $\epsilon = 1$.

With the above definitions, we can give a proper bound to the untraceability property of a blockchain system Π and thus compare them accordingly. The following section tackles unlinkability and provides analog comparison means.

4.4 Unlinkability

Whereas untraceability addresses the relation between input states with witnesses, unlinkability is a notion that focuses on the complete set of states in the graph. Indeed, unlinkability refers to the inability of the adversary to link two different addresses or transactions together [2]. Now, if the adversary can link two addresses, she can also link transactions involving those addresses. We consequently focus on a more concise definition of unlinkability: for any two addresses, it is impossible to prove that they belong to the same entity. For a full transaction graph, that translates to an adversary unable to identify the states that belong to the entity. The notion of unlinkability is also related to untraceability since identifying a transaction's true source could most likely increase the chance of linking states together.

Although blockchain systems function differently, all the states have an underlying address by which they are identified. Therefore, the main idea behind unlinkability is shared among the various systems. It consists of the adversary trying to compromise unlinkability by grouping the different states that belong to the same entity. The literature [35] agrees on the term cluster to address these groups of states. Also, to our knowledge, there is no systematic way to form these clusters. Research on this subject thus uses heuristics to associate multiple different addresses together. Those heuristics are mostly patterns in transactions, based on some evidence of shared ownership, that lead to believe that some states may refer to the same entity [26]. The term entity is intentionally imprecise here as it allows us to remain flexible in the linking we try to capture in this section. For example, an entity can refer to a physical person, a DeFi service, but also multiple people that buy goods together (e.g., a married couple, a group of friends, or an enterprise). Also, the literature on the topic does not agree on one definition or does not address the terminology, certainly because the term is evident based on the context. We thus define a **link** $(\cdot, \cdot)$ predicate to abstract notions of *linking* and *entity*.

Definition 4.6 (Link predicate)**.** The *link predicate* **link** (s_i, s_j) takes as input two states $s_i, s_j \in \mathcal{S}$ and outputs `true` if the two states refer to the same entity and `false` otherwise.

Yet, some states, although not equal, trivially use the same underlying address. They, therefore, return true on the link predicate evaluation, but we still need to differentiate both cases. We thus define a complementary predicate.

Definition 4.7 (Address predicate)**.** The *address predicate* **addr** (s_i, s_j) takes as input two states $s_i, s_j \in \mathcal{S}$ and outputs `true` if both states have the same underlying address and false otherwise. Moreover, we have that

1. if two states are equivalent, they both hold the same address, i.e., $\forall s_i, s_j : s_i = s_j \implies \mathbf{addr}(s_i, s_j)$;
2. if two states hold the same address, they are linked, i.e., $\forall s_i, s_j : \mathbf{addr}(s_i, s_j) \implies \mathbf{link}(s_i, s_j)$.

Finally, the property $\forall s_i, s_j : s_i = s_j \implies \mathbf{link}(s_i, s_j)$ follows by transitivity.

Per the above definition, linkability is trivial in blockchains where users do not perform address changes on different transactions. That gives us our first heuristic: if two states have the same address, they are linked. We also define it formally with our PDAG notation as

$$\forall s_i, s_j \in \mathcal{S} : \mathbf{addr}(s_i, s_j) \implies \mathbf{link}(s_i, s_j). \tag{4.1}$$

We denote the trivial heuristic in equation 4.1 as l and use it further to denote systems that do not provide unlinkability guarantees. When addresses are unique to each state, an adversary must rely on non-trivial heuristics. They can be seen as a condition on a subgraph $G' \subseteq G$ where the probability of linking a pair of states (s_i, s_j), i.e., $\Pr[\mathbf{link}(s_i, s_j)]$ is supposedly higher. We model here an adversary that relies on heuristics to cluster the different states of the graph; we define the adversarial heuristics accordingly.

Definition 4.8 (Adversarial heuristics)**.** The *adversarial heuristics* $\mathcal{H}_\Pi$ is a set that contains all the patterns and information regarding Π that help an algorithm $\mathcal{A}$ to progress. We denote by $\mathcal{A}(\mathcal{H}_\Pi)$ the use of $\mathcal{H}_\Pi$ by $\mathcal{A}$.

In contrast, the challenger may access a graph oracle to get the accurate linking tuples, the so-called ground truth.

Definition 4.9 (Linking graph oracle)**.** A *linking graph oracle* $\mathcal{G}_\Pi^{\text{Link}}$ takes a transaction graph (PDAG) G of a blockchain system Π as input and outputs a set L of tuple (state, state) such that both states are linked and distinct, i.e., $L = \{(s_i, s_j) \mid \mathbf{link}(s_i, s_j) \wedge s_i \neq s_j,\ \ s_i, s_j \in S\} \subseteq \mathcal{S} \times \mathcal{S}$.

We describe here a game $\text{Exp}_{\mathcal{A},Ch,\mathcal{H},\Pi}^{\text{Link}}$ – illustrated on figure 4.5 – that features, $\mathcal{A}(\mathcal{H}_\Pi, G)$ which takes a graph G as input and returns a pair of states (s_i, s_j). The adversary wins if both states are linked.

In the game, the adversary builds the graph G from the blockchain history and sends it to the challenger. In turn, Ch gets a set of tuples L from the linking graph oracle. The different tuples in L are pairs of states (s_i, s_j) that are linked, i.e., **link** (s_i, s_j). Finally, the challenger sends L to the adversary, who outputs a pair of states (s_i, s_j) as a guess. If that guess is in L, the adversary wins the game.

```
$Ch_{\mathcal{G},\Pi}(G)$
1 : $L \leftarrow \mathcal{G}_\Pi^{\text{Link}}(G)$
2 : $L := \{(s_i, s_j) \mid \mathbf{link}(s_i, s_j) \wedge s_i \neq s_j,\ \ s_i, s_j \in \mathcal{S}\}$
3 : return $L$
```

```
$\text{Exp}_{\mathcal{A},Ch,\Pi}^{\text{Link}}(\mathcal{H}_\Pi, G)$
1 : $G := (\mathcal{S} \cup \mathcal{N} \cup \mathcal{W}, \mathcal{E})$
2 : $L \leftarrow Ch_{\mathcal{G},\Pi}(G)$
3 : $(s_i, s_j) \leftarrow \mathcal{A}(\mathcal{H}_\Pi, G)$
4 : return $(s_i, s_j) \in L$
```

Figure 4.5: Challenger and linkability game for a transaction graph G

The adversary capacities, therefore, bound the graph's unlinkability. Similarly to the previous definition of untraceability, we define here ϵ-unlinkability concerning the experience outlined above.

Definition 4.10 (ϵ-unlinkability)**.** The graph G modelling an execution of blockchain system Π is ϵ-*unlinkable* for some $\epsilon \geq 0$ if

$$\Pr\left[\mathrm{Exp}^{\mathrm{Link}}_{\mathcal{A},\mathcal{G},\Pi}(G)\right] \leq \epsilon.$$

We assume that ϵ may be given for each blockchain using their best heuristics. In section 5.5, we show through some examples that the PDAG can be used to formalize different heuristics and thus compare unlinkability guarantees for other systems. These heuristics allow the adversary to identify the various states that are linked. We, therefore, define the fulfillment of unlinkability depending on the set of heuristics and its content.

Definition 4.11 (Achieving unlinkability)**.** A blockchain system Π

- ● *achieves unlinkability* if the adversary takes as input an empty set of heuristics and thus must rely on random guessing over the whole set of states, i.e., $\mathcal{H}_\Pi = \emptyset$ and $0 \leq \epsilon \leq \frac{1}{\mathcal{O}(|\mathcal{S}|^2)}$,
- ◑ *partially achieves unlinkability*, if the adversary takes as input a non-empty set of heuristics without the trivial heuristic, i.e., $l \notin \mathcal{H}_\Pi \neq \emptyset$ and $\frac{1}{\mathcal{O}(|\mathcal{S}|^2)} < \epsilon < 1$ or
- ○ *does not achieve unlinkability*, if the adversary takes as input a non-empty set of heuristics containing the trivial heuristic, i.e., $l \in \mathcal{H}_\Pi$ and $\epsilon = 1$.

This definition allows the comparison of the different blockchain systems presented in chapter 5. The following section also gives conditions for the fulfillment of confidentiality.

4.5 Confidentiality

In this section, we investigate confidentiality. Similarly to the previous ones, we present this notion and then provide a formal definition.

Informally, confidentiality represents the property that the transacted amount is not disclosed to non-involved parties. The two most prominent blockchain implementations – Bitcoin and Ethereum – do not have any mechanism to provide confidentiality. By reading the blockchain history, anybody can see every transaction amount. In contrast, some blockchains (e.g., Monero and shielded Zcash) obfuscate that information behind cryptographic mechanisms. Privacy-preserving blockchains thus typically contain two components: commitments and encrypted output amounts. The first guarantees that the amount respects specific properties, for instance, that inputs equal outputs or that the amount falls within a particular range. Meanwhile, the latter allows the recipients to know how many coins are in each output they own.

We here generalize the notion of *amount* to *payload*, i.e., any encrypted data secretly transmitted from sender to recipient during a transaction. The adversary thus tries to discover information on the *payload* γ, using the encrypted *payload* $\mathcal{E}_\gamma$ and additional information $h(\gamma)$ about the *payload* (e.g., the requirement of the commitment). We adapt the concepts of semantic security from modern cryptography that describes an eavesdropping adversary. According to this definition, a *payload* remains confidential if no PPT adversary can learn any polynomial-time computable functions f of the *payload* γ, regardless of the distribution $\mathcal{D}$ of γ. Confidentiality for a single state is, in turn, defined as the probability that an adversary $\mathcal{A}$ can output some polynomial-time computable function $f(\gamma)$ on the hidden *payload* γ.

Definition 4.12 (Single state ϵ-confidentiality)**.** For a transaction graph $G = (\mathcal{S} \cup \mathcal{N} \cup \mathcal{W}, \mathcal{E})$ and a single state $s \in \mathcal{S}$ with hidden *payload* γ and additional information $h(\gamma)$, *single state confidentiality* is defined as the probability that adversary $\mathcal{A}(s, h(\gamma))$ outputs some polynomial-time computable function $f(\gamma)$ on the hidden *payload* γ and a single state is ϵ-confidential for some $\epsilon \geq 0$ if

$$\Pr\left[\mathcal{A}(s, h(\gamma)) = f(\gamma)\right] \leq \epsilon.$$

However, if we consider a regular transaction with a witness w, a set of input states S_I and output states S_O, defining ϵ-confidentiality for every state $s \in S_I \cup S_O$ seems too strong since the fee of such transaction is usually in plain. There exists, therefore, a state where the adversary's probability of returning a function of the *payload* is always 1, i.e., $\exists s \in w\mathcal{E}_{C^\star} : \Pr\left[\mathcal{A}(s, h(\gamma)) = f(\gamma)\right] = 1$. Hence, we say such a state is 1-confidential.

Yet, a transaction might be confidential even though the fee is not. In turn, we consider only a subset of states $\mathcal{S}^\epsilon$ that exclude the fee. We thus define the non-trivial subset of states $\mathcal{S}^\epsilon \subset \mathcal{S}$ as a set of states that do not contain 1-confidential states. In contrast, the trivial set of states $\mathcal{S}^t$ contains every such states, i.e., $\mathcal{S}^t = \mathcal{S} \setminus \mathcal{S}^\epsilon$

This gives us the following definition of ϵ-confidentiality for a transaction where we only consider the confidentiality of the non-trivial states but with the knowledge of the trivial states.

Definition 4.13 (Transaction ϵ-confidentiality)**.** A transaction $t = (\mathcal{S}^{\epsilon\prime} \cup \mathcal{S}^{t\prime} \cup \mathcal{N} \cup \{w\}, \mathcal{E}')$ is *ϵ-confidential* for $\epsilon \geq 0$, if

$$\forall s \in \mathcal{S}^\epsilon : \Pr\left[\mathcal{A}\left(s, h\left(\gamma\right), \mathcal{S}^{t\prime}\right)\right] = f\left(\gamma\right)] \leq \epsilon,$$

for an adversary $\mathcal{A}$ with external knowledge $h\left(\gamma\right)$ about the hidden *payload* γ.

That definition is, therefore, weaker than having every transaction state being ϵ-confidential but does not allow for partial confidentiality either. For instance, in Zcash, a transaction might be partially shielded and thus have a subset of states with an encrypted *payload* and others with a plain one. In such a case, an adversary can devise a function on the encrypted *payload*s with the knowledge of the plain *payload*s, for instance, that encrypted amounts are upper bounded by the sum of the plain ones. Consequently, per definition 4.13, such a partially shielded transaction is not confidential.

Therefore, in this work, we consider either a fully confidential transaction or one which does not provide that property entirely. A system achieves confidentiality if, for all adversaries, there exists a negligible ϵ for which the adversary can devise a function on the hidden *payload*. We specify "negligible" as a function $\mathsf{negl}\left(n\right)$ according to the use in modern cryptography.

Definition 4.14 (Negligible [27])**.** A function f from the natural numbers to the non-negative real numbers is *negligible* if for every polynomial p there is an N such that for all $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.

Otherwise, when there exists an adversary that can output a function with probability $\epsilon \geq \frac{1}{p(n)}$, the system does not achieve confidentiality. The following definition sums up the previous statements.

Definition 4.15 (Achieving confidentiality)**.** A blockchain system Π

- ● *achieves confidentiality* if said system is ϵ-confidential with $\epsilon = \mathsf{negl}\left(n\right)$ for all PPT adversary $\mathcal{A}$ or
- ○ *does not achieve confidentiality* if there exists an adversary $\mathcal{A}$ for which $\epsilon \geq \frac{1}{p(n)}$.

In the last chapter 5, we will compare different blockchain systems and show which achieves confidentiality using the above definition. Before, we discuss, in the next section, how confidentiality may increase untraceability and unlinkability guarantees.

4.6 Discussion: Relation Between Notions

In this section, we emit intuitions about the relationship between the different privacy notions: untraceability, unlinkability, and confidentiality. We address relations between (1) untraceability and unlinkability, (2) unlinkability and confidentiality and (3) untraceability and confidentiality.

4.6.1 Untraceability and Unlinkability

As briefly addressed in section 4.4, knowing the true source of a transaction may increase the linkability capability of the adversary. We further show in section 5.5 that heuristics in the literature apply to systems where the source of the transaction is known. It appears easier to infer patterns and observe empirical relations between states if we know the actual state consumption. However, when the true sender is obfuscated, no consuming edges are visible from the adversary's point of view. Thus, we argue that compromising untraceability leads to a better chance of compromising unlinkability.

To the best of our knowledge, we have previously stated that no systematic way to break unlinkability exists. An adversary thus relies on heuristics to increase her chance of winning the linkability game. We consider here the heuristic that Nakamoto [40] describes in Bitcoin white paper. We call "input states grouping" a heuristic that assumes all input states to a witness belong to the same entity. This assumption looks plausible since, to spend a coin, one must know the private signing key belonging to the public key used as input. Multiple input states to the same witness are thus unlikely tied to different entities.

Moreover, many studies have shown that this heuristic is entirely accurate and difficult to evade for users [36, 46, 48, 2]. For example, if we consider the sample transaction in figure 4.6, our heuristic indicates that states from s_0 to s_4 all belong to the same entities, i.e.,

$$\forall s_i, s_j \in \star\mathcal{E}_C w : \mathbf{link}\left(s_i, s_j\right).$$

Now, we consider the case where the state consumption is not trivially visible, for instance, when decoy keys are used to mix inputs. In such a case, the input states grouping could not be used without a high rate of false positives since unrelated decoy keys would be linked to the accurate spender. It, therefore, becomes clear that if an adversary can compromise untraceability and identify the state consumption, she can use such linking heuristics and, thus, better jeopardize unlinkability.

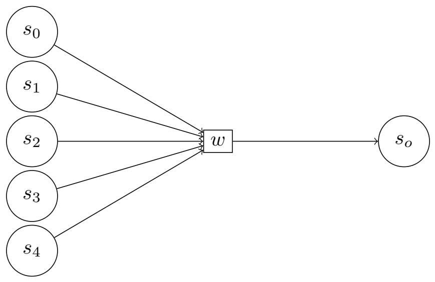


Figure 4.6: Under input states grouping heuristic, states s_0 to s_4 belong to the same entity.

In conclusion, untraceability facilitates unlinkability by rendering some linking heuristics obsolete or difficult to use. Confidentiality also has this property towards unlinkability, as explained in the next section.

4.6.2 Unlinkability and Confidentiality

The same principle applies when a system implements confidentiality. As a result, some heuristics are invalidated by obfuscating the transacted amount. We present a few simple heuristics that depend on the amount in section 5.5. Still, we describe in this section a far more pernicious linking attack that confidentiality could avoid happening. This attack [7] takes place in Zcash's blockchain that comprises a transparent pool (in which the amount is visible) and a shielded one that provides unlinkability and confidentiality. This attack could, however, be generalized to any setting in which confidentiality is only implemented at a subset of the transactions of the blockchain.

The principle is as follows. Biryukov and Feher [7] find that minimal coin values are scarce in the ledger. Yet, if they appear twice, an adversary can conclude that those two states or transactions are linked. Unique values in the ledger, therefore, act as fingerprints. This observation allows one active adversary to transfer a small but carefully chosen amount to a known public address of the victim. In turn, the victim transfers coins to the shielded pool to break the link between those subsequent transactions. At one point, the victim transfers some coins to a transparent address. The adversary monitors the activity out of the shielded pool, and as soon as the transaction takes place, she can link the two addresses due to the small amount resurfacing.

There exist other blockchain analyses based on the transacted amount. Mackenzie et al. [33] describe an attack against the original CryptoNote protocol where an adversary monitors individual outputs in the blockchain to link them together. For instance, if an adversary is aware of 0.9 coins being sent at a specific moment, they might be able to connect potential states in the blockchain by searching for transactions that contain 0.9 coins [41]. Now, the sender may use many change addresses in a transaction and obfuscate the transacted amount by dispersing it to multiple addresses. However, such a mechanism creates a lot of dust transactions in the ledger. If one merges the various states into one transaction, an adversary might be able to link back some keys belonging to the same entity.

This attack creates one of the main motivations for implementing Ring Confidential Transaction (Ring CT) in Monero. The following section also exposes additional motivations to Ring CT by presenting the link between untraceability and confidentiality.

4.6.3 Untraceability and Confidentiality

When the amount in each output is visible, untraceability is restricted to the set of outputs with the same amount. In Monero, before Ring CT, the members of the traceability set (the ring members) are required to be of the same amount. For less common transaction amounts, this would mean a smaller number of potential decoy outputs [33] and thus a smaller anonymity set.

Even though this problem arises with CryptoNote-based cryptocurrencies, we could see it being generalized to other blockchain systems that promote untraceability without implementing confidentiality. For example, suppose a technology does not reveal any information about which output is spent but lets the transacted amount be visible. In that case, an adversary could very well reduce the traceability set to states with the same amount by observing the inputs and outputs of a transaction. In the following, we speak about attacks on Monero prior to Ring CT but believe that it may apply to any blockchain implementation that promotes untraceability without hiding the amount.

Because before Ring CT, an adversary could quickly reduce the traceability of some transactions, Monero was vulnerable to various attacks. A Chain Reaction attack [39, 31], for instance, leverage the fact that some transactions with a smaller traceability set might reduce the traceability of subsequent transactions. Furthermore, Yu et al. [59] study an attack where an adversary can infer that some inputs are decoys by considering their historical usage [59]. For example, suppose two inputs, A and B, are the sole members of two identical rings. In that case, the adversary can deduce that further use of such inputs are decoys, thus reducing once more the traceability set. Similarly, Yu et al. [60] show an adversary that aims to exhaust all possible input choices by including them in multiple ring transactions. Hence, the attacker can reduce the ring size of the new transaction that spends the targeted output.

Consequently, we can assume that confidentiality increases the number of candidates to the traceability set and, for this reason, may enhance untraceability. In addition, confidentiality makes it hard to predict any future traceability set for subsequent transactions. We, therefore, think that confidentiality is a crucial property as it solidifies existing untraceability and unlinkability guarantees. This subsection concludes the chapter on the privacy notion. In the next chapter, we apply our model to existing privacy-preserving blockchains and compare them accordingly.

Chapter 5

Applications

In this chapter, we apply our model to two privacy-preserving blockchains – Monero and ZCash – and an add-on privacy solution for the Ethereum blockchain – Tornado Cash. We also briefly describe Bitcoin as background and reference to this chapter.

5.1 Bitcoin

This section aims at covering the Bitcoin blockchain. We are purposely brief in this section since the original TDAG paper [15] already describes a Bitcoin execution at length. We thus only give a general background of Bitcoin.

5.1.1 Background

Bitcoin (abbreviation: BTC) is a decentralized digital currency that relies on a blockchain of the same name. This system relies on Nakamoto consensus that utilizes Proof-of-Work (PoW) to decide on the ordering of transactions in the network. Transactions are recorded in blocks and can be of two types: coinbase or regular.

1. Coinbase transactions are created by miners as a reward for validating a block. They transfer yet unmined bitcoins to an address chosen by the miner. For such transactions to be valid, they must meet the following two conditions. First, transactions must transfer a specific number of bitcoins to an address based on the block height. Second, transactions must include a solution to the Proof-of-Work puzzle used to mine the block successfully.
2. Regular transactions involve transferring bitcoins from a set of input addresses to another set of output addresses. The fee for such transactions is calculated by subtracting the total amount of bitcoins in the input addresses from the total amount of bitcoins in the output addresses. This fee is paid to the miner who processes the transaction and includes it in a block. Regular transactions must have a confirmation for each input and output and must not create new bitcoins to be considered valid.

Bitcoins exist as unspent transaction outputs (UTXOs), which are assigned to addresses, i.e., public keys. The value of a UTXO is controlled by the holder of the corresponding private key and can be transferred to another address by signing a transaction with the private key. According to the original paper, we, therefore, model a state in Bitcoin as the following tuple

$$(addr, val, hash, height)\,,$$

where *addr* denotes the public key to which the output belongs to, *val* the number of coins held by said address, *hash* the cryptographic hash of previous states and *height* the block index at which the state is produced.

Bitcoin transactions are validated by following specific rules, such as ensuring that each input has a corresponding confirmation for the amount and that no new bitcoins are created.

Finally, Bitcoin does not provide inherent privacy features, but it is possible to use techniques like coin mixing and Coinjoin to increase unlinkability for transactions.

Coin mixing is a process by which a user's cryptocurrency is blended with other users' cryptocurrency to obscure the funds' origin and destination. This is typically done by sending the user's cryptocurrency to a mixing service, blending the funds with those of other users, and sending the mixed funds back to the user. The goal of coin mixing is to make it difficult for an outside observer to determine which inputs correspond to which outputs, thus increasing the unlinkability of the transaction.

Coinjoin [34] is also a technique that allows multiple users to combine their transactions into a single one. However, it does not involve a centralized mixing service, and the inputs and outputs are typically controlled by the participating users rather than a third party. Coinjoin can be implemented in various ways. Yet, the main idea is that multiple users agree to create a single transaction that includes inputs and outputs from all of their individual transactions. We do not model a Coinjoin execution since it only consists of a MIMO transaction. However, we speak further about this mechanism in section 5.5 and give intuition to its unlinkability guarantees.

Consequently, there exist mechanisms to promote unlinkability inside Bitcoin's blockchain. Yet, to the best of our knowledge and *according to our definitions*, there exists no mechanism to provide untraceability or confidentiality without requiring inherent changes in the bitcoin blockchain protocols, like – as we will see in section 5.3 – Zcash does.

We here do not provide a transaction graph nor an execution sample for Bitcoin as opposed to subsequent sections. Interested readers may find a similar graph and execution in the transparent pool of Zcash outlined in section 5.3, and a thorough description of such elements in the original paper [15]. The following section, however, tackles in depth the Monero blockchain.

5.2 Monero

This section explains the Monero blockchain and defines a formal transaction graph according to chapter 3.

5.2.1 Background

Monero[1] (Abbreviation: XMR) is based on the CryptoNote protocol, which by default provides untraceability with ring signatures and unlinkability with stealth addresses [53]. Since transaction outputs cannot be inspected to determine if they have been spent or not, CryptoNote and thus Monero, are in no way a UTXO-based model. We argue below that Monero is a nullifier-based model since additional components are produced at each transaction and must be verified for subsequent ones, bringing the need for nullifier states. We, therefore, give an overview of the operation in Monero and support our explanation with a technical guide by Koe et al. [29]. We thus omit the citation in this section. We begin by explaining the system of stealth addresses, then proceed to discuss ring signatures and finally ring confidential transactions.

Unlike in other conventional cryptocurrencies, Monero users have two sets of private/public keys, (k^v, K^v) and (k^s, K^s), called respectively the *view* and the *spend* key pairs. Users' public addresses are the pair of public keys (K^v, K^s), and their private keys are consequently the pair (k^v, k^s). To receive coins, users may therefore disclose their public address to others. Yet, the address itself is never used directly, but, instead, a one-time address K^o (also called stealth address) is created for each transaction output o. This address is then used in subsequent transactions to prove ownership of the output o by signing a message with the corresponding private key k^0. This first mechanism ensures that one-time addresses are used for each transaction, thus promoting unlinkability.

As for untraceability, Monero enforces that property by using a cryptographic primitive called ring signatures [47]. Ring signatures are composed of a ring and a signature. The former is a set of public keys, one of which belongs to the signer. The latter is computed with the ring and the signer's private key. A valid signature guarantees that it comes from one of the ring members, but any verifier cannot tell which one of them is the signer. Both of these elements are part of the input of a transaction. Yet, since the ring signature hides which output is spent, some mechanism must be implemented to prevent an output from being spent twice. For that purpose, a key image I is computed as part of the output of a transaction. It uses the related private key of the output k^0 to compute such an

[1]The native cryptocurrency is also named Monero, and Moneroj for plural.

image. Since that private key uniquely identifies the output, key images are distinct for two different outputs but identical for the same output. For each transaction, one needs to verify if the key image is already in the full set of spent key images; if not, the transaction has a valid input and can thus proceed. As we will see, this set of key images will form the nullifier states in the transaction graph of the next section, and that is why we refer to Monero as a nullifier-based model.

Since 2017, Monero has further strengthened its privacy properties by introducing *ring confidential transactions* (RingCTs) [41]. This scheme provides amount hiding compatible with ring signatures. It consists of several mechanisms to implement confidentiality while still maintaining some crucial properties. Firstly, there is an encryption scheme of the amount to provide confidentiality itself. The encrypted amount can be decrypted using the private view key of the user that owns the output. By proceeding this way for each output they own, users can reconstruct the total balance of their wallet. Furthermore, no information may be derived from this scheme when transaction amounts are encrypted. However, one property that must hold is that the sum of the inputs equals the one from the outputs. For that purpose, an extra scheme called Pedersen commitment is used in Monero. This mechanism hides the transaction amounts while being additively homomorphic, i.e., for two plain values a and b and a commitment $\mathcal{C}$, it holds that $\mathcal{C}(a+b) = \mathcal{C}(a) + \mathcal{C}(b)$. Therefore, one can ensure that input and output sums are equal without revealing further information about the amount. One problem with additive commitments is that it also holds if one value is negative, say $\mathcal{C}(6+5) = \mathcal{C}(21-10)$. In that case, the value -10 is illegal since values must be greater or equal to 0. In turn, another scheme, called Bulletproof [12], guarantees that the amount is within a specific range, between 0 and $2^{64}-1$.

A state, therefore, contains the encrypted amount, output commitment, and range proof. We detail this in the following subsection with a thorough execution of Monero.

5.2.2 Transaction graph

We revisit the structure of the original paper [15] to define an execution, a transaction graph, transaction types, and a transaction predicate in Monero. Also, any notation we introduce in this section, we summarise in table 5.1.

Much like Bitcoin, a Monero blockchain is composed of blocks, with a new block being added once every 2 minutes. A block is a container for transactions, which are created through the process of mining. The set of transactions included in the block is *regular transactions* and one *coinbase transaction*, which adds newly created coins to the network. The total output of a coinbase transaction is equal to fees from all the regular transactions included in the block and the mining reward. This reward is computed with $R = \frac{L-M}{10^{12}}$ with $L = 2^{64}-1$ being the fixed limit of money supply and M the current money supply in the blockchain. That means the reward is slowly declining until a point, after which all block rewards will be fixed. The first reward is given in the genesis block with $R_0 = \frac{L}{10^{12}} \approx 17.6$. There is consequently no initial pre-mine, as opposed to Bitcoin. The total coin supply is, therefore, not capped but increases indefinitely. We call such a paradigm inflationary. Regarding regular transactions, one is considered valid if

(1) the ring signatures are individually signed with a valid private key corresponding to one public key from the ring itself (also meaning the output belongs to the spender),

(2) the commitments are valid (no new coins are produced),

(3) the range proofs are valid (the amount is legal),

(4) the key images produced are not contained in the set of all previously produced images (no double-spending).

Finally, as stated above, a regular transaction has an associated fee relative to the transaction's size (0.002 XMR per kB). We proceed with the first definition regarding Monero's execution.

Definition 5.1 (Monero execution)**.** A *Monero execution* $\mathcal{L}_{\text{XMR}}$ is a set of blocks $\mathcal{B} := \{B_g, B_1, \ldots, B_n\}$, where B_g is the genesis block and contains a single initialization coinbase transaction CBTX with a mining proof MP, i.e., $B_g = (\text{MP}, \text{CBTX})$. Other blocks are tuples with a mining proof MP and a set of transactions with one coinbase transaction, CBTX, and n regular transactions, RTX, i.e., $B_i = (\text{MP}, \{\text{CBTX}, \text{RTX}_1, \ldots, \text{RTX}_n\})$. In addition, a coinbase transaction CBTX contains a Monero address ADDR to convey the block reward. A regular transaction RTX is a tuple $(\mathcal{I}_{\text{IN}}, \mathcal{R})$ where $\mathcal{I}_{\text{IN}}$ is a set of key images given as input and $\mathcal{R}$ is a set of rings. The set $\mathcal{R}$ contains rings $r_1, \ldots, r_k$ with $r_j := (\text{SIG}, \text{ADDR}_{\text{IN}}, I)$, where SIG is a valid signature from one of the members of the rings included in the set of ring addresses ADDR_{IN} and I is the key image produced by the spender.

Now, we specify a given Monero execution as a PDAG. A state in Monero represents a stealth address that holds an output or a transaction fee. A transaction fee does not belong to any address before being mined; we denote this by the placeholder value $\perp$. The genesis state is one such state, representing the first transaction fee of 0 that can be mined to get the initial reward of R_0. Each witness represents a valid mining proof or the four prerequisites listed above. Furthermore, we consider all possible types of edges. Consuming edges are mostly used for states that represent fees, observing ones for decoy keys and nullifiers taken as input, and masking edges to highlight the state that is truly consumed inside a ring signature. As for producing edges, they link a witness to newly created states and nullifiers.

Definition 5.2 (Transaction graph for Monero)**.** We model an execution of Monero system as a *transaction graph* $G_{XMR} := (\mathcal{S}_{XMR} \cup \mathcal{N}_{XMR} \cup \mathcal{W}_{XMR}, \mathcal{E}_{XMR})$ defined as follows:

- **State**. Each state $s \in \mathcal{S}_{XMR}$ is defined as $(addr, commit, bproof, \mathrm{Enc}_k(\gamma), hash, height)$ where *addr* is the recipient stealth address of the state, *commit* is the Pedersen commitment, *bproof* is the Bulletproof, and $\mathrm{Enc}_k(\gamma)$ is the encrypted amount γ, *hash* is the result of applying $\mathbf{H}$ to a set of vertices $S'_{\mathrm{XMR}} \cup \mathcal{N}_{XMR} \cup \{w\}$ with $S'_{\mathrm{XMR}} \subset S_{\mathrm{XMR}}$, and *height* denotes a block index. If γ is plain, the state still possesses valid commitment and Bulletproof. The genesis state s_g is defined as the fixed tuple $(\perp, \mathcal{C}(0), \mathcal{P}(0), 0, \mathbf{H}(\emptyset), 0)$.
- **Nullifier**. Each nullifier $n \in \mathcal{N}_{XMR}$ is a set of key images $\mathcal{I} = \{I_{w_t}, \ldots, I_{w_{t+i}}\}$ for transaction w_t to w_{t+i}.
- **Witness**. Each witness $w \in \mathcal{W}_{XMR}$ is defined by a tuple $(txtype, \mathcal{F})$, where *txtype* is the transaction type and determines the content of $\mathcal{F}$ according to definition 5.1.
- **Edge**. Each edge $e \in \mathcal{E}_{\mathrm{XMR}}$ may have any type, i.e., producing, masking, observing, or producing.

The transaction graph defined above determines the possible transactions in a Monero execution. The following definition maps transactions in a Monero execution to transaction types supported in a PDAG.

Definition 5.3 (Transaction types in Monero)**.** A coinbase transaction is modeled as either a SISO or MISO transaction, depending on the number of regular transaction fees taken as input. Furthermore, there is only one miner that takes the reward. A regular transaction is either a SIMO or MIMO. The multiple outputs are inevitable since ring signature has been mandatory since 2018 [50] and will thus output a nullifier. A single input is possible – even though now the ring signature must be of size 11 – because the first transaction is left with no choice but to be alone in the ring (only one output state exists). When the ring size of 11 can be respected, only MIMO transactions may occur for regular transactions.

Finally, we conclude our description of the Monero context with the corresponding transaction predicate $\mathbb{P}$. We define $\mathrm{VerifyRing}(r)$ to return `true` for a ring $r := (\mathrm{ADDR_{IN}}, \mathrm{SIG}, I)$, whenever SIG is a valid signature for the ring with the set of addresses $\mathrm{ADDR_{IN}}$ as input and `false` otherwise. Moreover, we define $\mathrm{Range}(\mathcal{P}(\cdot))$ to be a valid Bulletproof between range 0 and $2^{64} - 1$. Also, we define $\mathrm{VerifyCommitments}(S_1, S_2)$ that takes two sets of states $S_1, S_2 \subseteq S$ and verifies if the sum of the commitments included in S_1 is equal the sum of commitments in S_2. Lastly, we also define $\mathrm{VerifyWork}(\mathrm{MP})$ a predicate that takes a mining proof MP as input and returns `true` if it is valid and `false` otherwise.

Definition 5.4 (Transaction predicate in Monero)**.** Let $t := (\mathcal{S} \cup \mathcal{N} \cup \{w\}, \mathcal{E})$ be a transaction. Then, the *transaction predicate* $\mathbb{P}(t)$ returns `true` if the following conditions hold and `false` otherwise. We denote a transaction $w := (txtype, \mathcal{F})$ with $\mathcal{F}$ being $(\mathcal{I}_{\mathrm{IN}}, \mathcal{R})$ for a regular transaction and $(\mathrm{MP}, \mathrm{ADDR})$ for a coinbase one.

1. If t is a regular transaction, the witness holds valid ring signatures:

$$\forall r \in w.\mathcal{R},\ \mathrm{VerifyRing}(r).$$

2. If t is a regular transaction, the key images produced are not part of the input key images:

$$\forall I \in w.\mathcal{R},\ I \notin \mathcal{I}_{\mathrm{IN}}.$$

3. If t is a coinbase transaction, the witness contains a valid mining proof:

$$\forall w.\mathrm{MP} : \mathrm{VerifyWork}(\mathrm{MP}).$$

4. All input, and output states hold a valid Bulletproof:

$$\forall s_i, s_o \in s_i \mathcal{E}_I w \mathcal{E}_P s_o, \mathcal{E}_I = \mathcal{E}'_M \cup \mathcal{E}'_O : \mathrm{Range}\left(s_i.\mathcal{P}\right) \wedge \mathrm{Range}\left(s_o.\mathcal{P}\right).$$

5. The sum of the input commitments is equal to the sum of the output commitments:

$$\forall S_I, S_O : S_I \mathcal{E} w \mathcal{E}_P S_O \wedge \mathrm{VerifyCommitments}\left(S_I, S_O\right).$$

6. Each output state contains the evaluation of the hash function over consumed input states, nullifiers, and the witness:

$$\forall s \in w \mathcal{E}_P \star : s.\mathit{hash} = \mathbf{H}\left(\mathcal{N} \cup \star \mathcal{E}_C w \cup \{w\}\right).$$

Notation	Name
G_{XMR}	Monero TDAG
$\mathcal{V}_{\text{XMR}}$	Monero vertices
$\mathcal{E}_{\text{XMR}}$	Monero edges
$\mathcal{S}_{\text{XMR}}$	Monero states
$\mathcal{N}_{\text{XMR}}$	Monero nullifiers
$\mathcal{W}_{\text{XMR}}$	Monero witnesses
$(\mathit{addr}, \mathit{commit}, \mathit{bproof}, \mathrm{Enc}_k\left(\gamma\right), \mathit{hash}, \mathit{height})$	Monero state with stealth address, commitment, Bulletproof, encrypted amount, hash of previous vertices and block height
(k^v, K^v)	View key pairs
(k^s, K^s)	Spend key pairs
R	Block reward
L	Total money supply
$\mathcal{C}\left(\gamma\right)$	Commitment for value γ
$\mathcal{P}\left(\gamma\right)$	Bulletproof for value γ
$\mathcal{L}_{\text{XMR}}$	Monero execution
$\mathcal{B}$	A set of blocks
ADDR	A stealth address
B_i	A block i
B_g	Genesis block
CBTX	Coinbase transaction
RTX	Regular transaction
MP	Mining proof
I	A key image
$\mathcal{I}$	Set of key images
$\mathcal{R}$	A set of rings
r_j	A ring
SIG	A signature to a ring
$\mathcal{F}$	A set of confirmation
$\mathbf{H}\left(S'\right)$	Evaluation of hash function over the set S'
$\mathrm{VerifyRing}\left(\cdot\right)$	Boolean function for ring verification
$\mathrm{VerifyWork}\left(\cdot\right)$	Boolean function for mining proof verification
$\mathrm{VerifyCommitments}\left(\cdot, \cdot\right)$	Boolean function for commitments summation verification
$\mathrm{Range}\left(\cdot\right)$	Boolean function for Bulletproof range verification

Table 5.1: Notations used to describe a Monero execution.

5.2.3 Model analysis and execution example

We prove here that the Monero execution defined above is a valid PDAG and give an example execution.

Lemma 5.1. Assume $\mathbf{H}\left(\cdot\right)$ is a collision-resistant hash function [27] and assume that $\mathcal{L}_{\text{XMR}}$ is a legal Monero execution. Then, the graph G_{XMR} resulting from modelling $\mathcal{L}_{\text{XMR}}$ is a PDAG.

Proof. We show that $G_{\text{XMR}} = (\mathcal{S}_{\text{XMR}} \cup \mathcal{N}_{\text{XMR}} \cup \mathcal{W}_{\text{XMR}}, \mathcal{E}_{\text{XMR}})$ fulfils the conditions to be a PDAG according to definition 3.2.

1. The genesis state has only one consuming edge since it allows the first coinbase transaction to take place.
2. Every state, except the genesis state, is produced exactly once. Assume by contradiction that this is false and that a state $s \in S_{\text{XMR}}$ has two producing edges. That would imply that there exist two different sets V and V' such that $\mathbf{H}(V) = \mathbf{H}(V')$ or that two distinct witnesses w_1 and w_2 are valid with the same key image I as input. The first contradicts the assumption of collision resistance of $\mathbf{H}(\cdot)$. The latter contradicts the prerequisite of $\mathbb{P}$.
3. Every state except for the genesis state may have multiple successors, but at most, one among them is connected with a consuming edge. Every state is consumed exactly once in Monero for an execution to be legal. This condition therefore holds.
4. The nullifier states have no consuming or masking edges by construction.
5. The graph must be weakly connected because each transaction consumes a previously unconsumed state in the graph. Either a Monero coin remains unspent or, when spent, is consumed.
6. The graph has no cycles. If there were a cycle, that would mean that two distinct transactions would produce the same state. That would imply that two different transactions, t' and t', produce the same state. As seen before, this contradicts the fact that $\mathbf{H}(\cdot)$ is collision-resistant and that key images are unique.

□

Following this proof, we model a sample execution of Monero. We assume for simplicity that the transaction fee is 0.1 instead of one dependent on the transaction size. We focus on the illustrative example in figure 5.1. It shows a possible Monero execution $\mathcal{L}_{\text{XMR}} = \{B_g, B_1, B_2\}$ inspired by the original paper [15], where $B_g := (\emptyset, \{t_0\})$, $B_1 := (MP, \{t_1\})$ and $B_2 := (MP', \{t_2, t_3, t_4\})$. This sample execution is the same as given in figure 3.2, but we here give the multiple execution details since we have defined them earlier in this section. We furthermore provide a high-level illustration of the cryptocurrency flow in figure 5.2 in which Alice is depicted with variable shirt color along multiple transactions to showcase the stealth address mechanism inherent to Monero.

- $t_1 := (\{s_g, s_A, w_1\}, \{(sg, w_1), (w_1, s_A)\})$, where $(s_g, w) \in \mathcal{E}_C$ and $(w, s_A) \in \mathcal{E}_P$. This is the initialization transaction where
  - $s_g := (\perp, \mathcal{C}(0), \mathcal{P}(0), 0, \mathbf{H}(\emptyset), 0)$,
  - $w_0 :=$ (TINITX, MP) and
  - $s_A := (\text{ADDR}_{s_A}, \mathcal{C}(17.6), \mathcal{P}(17.6), 17.6, \mathbf{H}(\{s_g, w_1\}), 1)$
- $t_2 := (\{s_A, s_1, n_1, s'_A, s_B, w_2\}, \{(s_A, w_2), (w_2, s_1), (w_2, n_1), (w_2, s'_A), (w_2, s_B)\})$, where $(s_A, w_2) \in \mathcal{E}_M$ and $(w_2, s_1), (w_2, n_1), (w_2, s'_A), (w_2, s_B) \in \mathcal{E}_P$. This is a transaction where Alice uses a ring signature to send 2 coins to Bob with a fee of 0.1. We detail the different states below, with
  - $s_1 := (\perp, \mathcal{C}(0.1), \mathcal{P}(0.1), 0.1, \mathbf{H}(\{w_2\}), 1)$,
  - $s'_A := \left(\text{ADDR}_{s'_A}, \mathcal{C}(15.6), \mathcal{P}(15.6), \text{Enc}_{k_{s'_A}}(15.6), \mathbf{H}(\{w_2\}), 1\right)$
  - $s_B := (\text{ADDR}_{s_B}, \mathcal{C}(1.9), \mathcal{P}(1.9), 1.9, \mathbf{H}(\{w_2\}), 1)$,
  - $w_2 := (\text{RTX}, (\emptyset, \{(\{s_A\}, \text{SIG}_{s_A}, I_{s_A})\}))$,
  - $n_1 := \{I_{s_A}\}$.
- $t_3 := (\{n_1, s'_A, s_B, w_3, s_2, n_2, s''_A, s_C\}, \{(n_1, w_3), (s'_A, w_3), (s_B, w_3), (w_3, s_2), (w_3, n_2), (w_3, s''_A), (w_3, s_C)\})$, where $(s'_A, w_3) \in \mathcal{E}_M$, $(n_1, w_3), (s_B, w_3) \in \mathcal{E}_O$ and $(w_3, s_2), (w_3, n_2), (w_3, s''_A), (w_3, s_C) \in \mathcal{E}_P$. This is a transaction where Alice uses a ring signature to send 5 coins to Carol with a fee of 0.1. We again detail the different states below, with
  - $s_2 := (\perp, \mathcal{C}(0.1), \mathcal{P}(0.1), 0.1, \mathbf{H}(\{n_1, w_3\}), 1)$,
  - $s''_A := \left(\text{ADDR}_{s''_A}, \mathcal{C}(10.6), \mathcal{P}(10.6), \text{Enc}_{k_{s''_A}}(10.6), \mathbf{H}(\{n_1, w_3\}), 1\right)$

  - $s_C := \left(\text{ADDR}_{s_C}, \mathcal{C}(4.9), \mathcal{P}(4.9), \text{Enc}_{k_{s_C}}(4.9), \mathbf{H}(\{n_1, w_3\}), 1\right)$,
  - $w_2 := \left(\text{RTX}, (\{I_{s_A}\}, \{(\{s'_A, s_B\}, \text{SIG}_{s'_A}, I_{s'_A})\})\right)$ and
  - $n_2 := \{I_{s_A}, I_{s'_A}\}$.

- $t_4 := (\{s_1, s_2, w_4, s_M\}, \{(s_1, w_4), (s_2, w_4), (w_4, s_M)\})$ where $(s_1, w_4), (s_2, w_4) \in \mathcal{E}_C$ and $(w_4, s_M) \in \mathcal{E}_P$. This transaction is the mining process from user Maurice, which gets both fees of s_1 and s_2 and the mining reward of roughly 17.6. We detail the state of Maurice below as well as the witness of the coinbase transaction.
  - $s_M := (\text{ADDR}_{s_M}, \mathcal{C}(17.8), \mathcal{P}(17.8), 17.8, \mathbf{H}(\{s_1, s_2, w_4\}), 2)$
  - $w_4 := (\text{CBTX}, \text{MP}')$

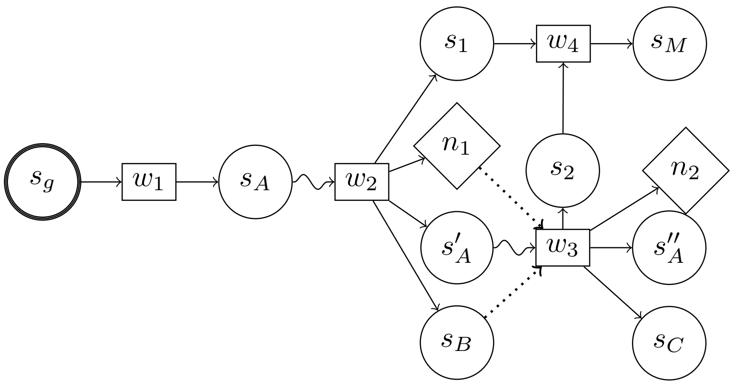


Figure 5.1: A sample execution of Monero's blockchain

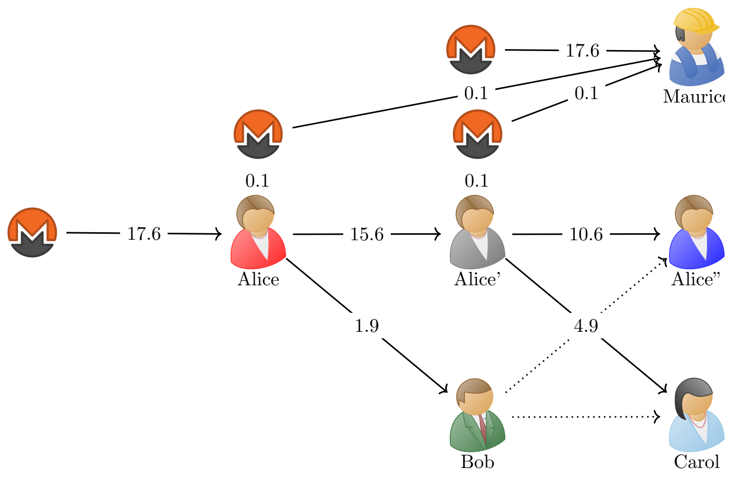


Figure 5.2: High-level cryptocurrency flow.

This section shows that the PDAG can adequately capture Monero's execution. We demonstrate in the next section that it can also represent one of Zcash.

5.3 Zcash

This section follows the same structure as the previous one to describe Zcash, beginning with a general background on its activity, proceeding to specify the transaction graph, and finally describing a model analysis as well as an execution example. As a reference, we use the Zcash protocol specification by Bow et al. [11] and allow ourselves to omit this reference in the remainder of this section.

5.3.1 Background

Zcash (Abbreviation: ZEC) is a fork of the Bitcoin codebase, but with the added feature of providing enhanced privacy for its users. To address this, Zcash implements a so-called shielded pool, which is a blockchain subset where advanced cryptographic mechanisms are used to provide untraceability, unlinkability, and confidentiality. Outside the shielded pool, the blockchain is referred to as the transparent pool and its organization is very much similar to the one of Bitcoin. Since there are two parts to the blockchain, namely transparent (t) and shielded (z), Zcash supports four different types of transactions: t-to-t, z-to-z, t-to-z and z-to-t. We here briefly describe their main features and go further into detail in the rest of this subsection. We thus momentarily use words such as *obfuscate* or *hide* to abstract away the technical details.

(1) Like previously said, a t-to-t transaction (or public transaction) functions like a standard Bitcoin transaction, with the sender, receiver, and transaction values completely visible.

(2) A z-to-z transaction (or private transaction) appears on the blockchain but, the addresses and the transacted amount are all obscured. Only the fee paid for the transaction appears in plain. This type of transaction uses zero-knowledge proofs in order to hide the source of a transaction, and shielded one-time addresses to blur the links between several private transactions. We explain those two mechanisms further down this subsection.

(3) A t-to-z transaction (or shielding transaction) obscures the recipient of the transaction but not the sender or the amount.

(4) In contrast, a z-to-t transaction (or deshielding transaction) hides the source but reveals the destination, including the received amount.

In Zcash, any transaction interacting with the shielded pool (i.e., transactions (2) to (4) above) does so by utilizing a vJoinSplit, a cryptographic construction which serves to designate the origin and destination of the coins being transferred. The incoming part of a vJoinSplit includes a list of input t-addresses, two nullifiers, and a zero-knowledge proof. If the list of t-addresses is empty, the transaction is converting shielded coins to transparent ones (type (4)) or transferring coins inside the shielded pool (type (2)).

As for the outgoing part, the vJoinSplit includes a list of output t-addresses, two shielded outputs and an encrypted memo field. Now, if the t-address list is empty, the transaction is either converting transparent coins to shielded coins (type (3)) or transferring coins privately (type (2)). Each shielded output contains an unknown quantity of ZEC and a hidden double-spending nullifier. A shielded output may also be a decoy output, containing zero ZEC, to conceal the absence of a shielded output. As for the encrypted memo, it can be used to privately send additional information to the recipients.

Two elements stand out in a vJoinSplit transaction, nullifiers – also called double-spending tokens – and the zero-knowledge proof. The first is uniquely computed for each shielded output and therefore ensures that each output is only spent once. For decoy outputs, a dummy nullifier is computed. As for the latter, it is a novel form of zero-knowledge cryptography, a so-called "Zero-Knowledge Succinct Non-Interactive Argument of Knowledge" (zk-SNARK). It allows for the spender to prove multiple properties without leaking any information on them. First, she proves that the nullifiers belong to some shielded output. Secondly, she proves that the sum of the inputs is equal to the sum of the outputs. To be more precise, the sum of the input t-addresses and the values represented by the nullifiers is equal to the sum of the output t-addresses, the values of the generated shielded outputs, and the miner's fee. The proof is also succinct, meaning it can be verified in a matter of a few milliseconds. Moreover, the proof does not require multiple rounds of message exchanges but possesses a "non-interactive" construction that only requires a single message from prover to verifier.

The fundamental principle outlined above is that, when an output is consumed, the spender needs only demonstrate that *some* commitment associated with the note has been disclosed, without revealing which one in particular. This means that it is not possible to link a specific note to the transaction in which it was created. From the perspective of an adversary, the potential set of previous notes that could be the input for a given transaction — the traceability set — includes all previous notes that the adversary does not control or know to have been spent. This contrasts with other approaches in private payment systems, such as Monero, which rely on mixing a limited number of outputs and therefore have smaller traceability sets.

Regarding unlinkability, users have a spending key that they use to generate shielded payment addresses, which they can publicly disclose to receive payments. A single shielded payment may be used to receive multiple different payments from several users, and the link between those payments will not be revealed. However, if two parties collude, they will trivially be able to figure out that their payments are destined for the same user. To prevent that, a user may generate different shielded addresses for each payer. Also, to each shielded payment address, there is an incoming viewing key that allows recipients to recognize which output they own. Furthermore, there is an authorizing key that allows one to actually spend a note but, also, to uniquely compute a corresponding nullifier when spending a note.

It should now appear clearly that the transparent pool of Zcash obeys the UTXO-based model, but that the shielded blockchain is a nullifier-based one. The transparent pool is indeed a fork of Bitcoin and therefore follows the logic of unspent transaction outputs. The shielded pool possesses additional components to prevent double-spending – nullifiers – and thus belongs to a nullifier-based model. This property of Zcash makes it a *hybrid* model. In the next subsection, we further describe Zcash with a transaction graph.

5.3.2 Transaction Graph

Like in the previous section, we here define an execution, a transaction graph, transaction types, and a transaction predicate in Zcash and summarise our notations in table 5.2.

Since Zcash is a Bitcoin code fork, they share many core features. Zcash relies however on a slightly different Proof-of-Work that uses Equihash instead, a memory-oriented and instantly verifiable PoW algorithm [9]. Zcash has therefore a faster block mining rate of roughly 75 seconds (against 600 for Bitcoin) but also a lower ZEC creation per block to mirror Bitcoin's monetary supply. Unsurprisingly, transparent Zcash supports two types of transactions – coinbase and regular, and has a pre-mine of $21 \cdot 10^6$ ZEC. A coinbase transaction transfers unmined ZEC to a set of transparent Zcash addresses as a reward for creating the block. A regular transparent transaction (t-to-t) transfers ZEC from a set of transparent addresses to another one. Such a transaction is valid if

1. it includes a confirmation for each input address and
2. does not create new coins.

In contrast, a transaction that interacts with the shielded pool is valid if the zero-knowledge proof is valid, meaning

1. the sender proves knowledge of the private spending keys of the input notes (sender has spending authority),
2. nullifiers are computed correctly (they belong to some shielded notes),
3. the generated nullifiers do not collide with pre-existing nullifiers (no double-spending) and
4. the sum of the inputs is equal to the sum of outputs (no new coins are produced).

Such a transaction can be a shielding, private or deshielding transaction. We here simplify the terminology and refer to such transactions as shielded transactions to define a Zcash execution.

Definition 5.5 (Zcash execution)**.** A *Zcash execution* $\mathcal{L}_{\text{ZEC}}$ is a set of blocks $\mathcal{B} := \{B_g, B_1, \ldots, B_n\}$, where B_g is the genesis block and contains a single initialization coinbase transaction CBTX with a mining proof MP, i.e., $B_g = (\text{MP}, \text{CBTX})$. Each other block $B_i := (\text{MP}, \{\text{CBTX}, \text{RTX}_1, \ldots, \text{RTX}_n\})$ is a tuple composed of a proof of successful mining MP, and a set of transactions containing a coinbase transaction CBTX and n regular transactions RTX_i. A CBTX contains a Zcash address, either transparent TADDR or shielded ZADDR. A RTX is either transparent (TRTX) or shielded (ZRTX). A TRTX is a tuple $(\text{ADDR}_{\text{IN}}, \mathcal{F}, \text{ADDR}_{\text{OUT}})$, where ADDR_{IN} and ADDR_{OUT} are two sets of transparent Zcash addresses and $\mathcal{F}$ is a set of confirmation. A ZRTX is a tuple $(\text{ADDR}_{\text{IN}}, \mathcal{Z}, \text{ADDR}_{\text{OUT}})$, where ADDR_{IN} and ADDR_{OUT} are two sets of Zcash addresses (with at least one of which being shielded) and $\mathcal{Z}$ is a zero-knowledge proof.

Unsurprisingly, our PDAG model is very close to the one of Bitcoin in the original paper introducing TDAGs [15]. A state represents a Zcash address with a certain amount of coins. We also associate unmined ZEC and fees to states even though they are not tied to any address yet. The genesis state holds the 21M ZEC ever existing in the system. Moreover, each witness represents either proof of successful mining for a block or the zero-knowledge proof required in a regular transaction. Finally, we consider all possible types of edges over the whole blockchain, but observing and masking edges are only used inside the shielded pool and consuming edges are only in the transparent pool. Consuming edges link unconsumed addresses for unmined ZEC and transaction fees to the mining proof or an input address to the zero-knowledge proof. Producing edges link a mining proof to the Zcash addresses getting the reward or confirmations to the corresponding output addresses receiving the transferred coins. Observing edges links nullifiers to a witness to access the nullifier set of the blockchain or any state that is not consumed in a shielded transaction. Masking edges link yet unconsumed states to a witness in a shielded transaction.

Definition 5.6 (Transaction graph for Zcash)**.** We model an execution of Zcash system as a *transaction graph* $G_{\text{ZEC}} := (\mathcal{S}_{\text{ZEC}} \cup \mathcal{N}_{\text{ZEC}} \cup \mathcal{W}_{\text{ZEC}}, \mathcal{E}_{\text{ZEC}})$ defined as follows:

- **State**. Each state $s \in \mathcal{S}_{\text{ZEC}}$ is defined as tuple $(\mathit{addr}, \mathit{val}, \mathit{hash}, \mathit{height})$, where *addr* is a Zcash address, *val* is a value held by said address (which can be encrypted), hash denotes the result of applying $\mathbf{H}(\cdot)$ to a set of vertices $\mathcal{S}_{\text{ZEC}}' \cup \mathcal{N}_{\text{ZEC}} \cup \{w\}$ with $\mathcal{S}_{\text{ZEC}}' \subset \mathcal{S}_{\text{ZEC}}$, and *height* denotes a block index. The genesis state s_g is defined as the fixed tuple $(\text{ADDR}_0, 21M, \mathbf{H}(\emptyset), 0)$.
- **Nullifier**. Each nullifier $n \in \mathcal{N}_{\text{ZEC}}$ is a set of uniquely computed values for each spent note, i.e., $\mathrm{P} = \{\rho_{w_t}, \ldots, \rho_{w_t+i}\}$ for transaction w_t to w_{t+i}.
- **Witness**. Each witness $w \in \mathcal{W}_{\text{ZEC}}$ is defined by a tuple $(\mathit{txtype}, \mathcal{R})$, where *txtype* denotes the type of transaction and determines the content of $\mathcal{R}$.
- **Edge**. Each edge $e \in \mathcal{E}_{\text{ZEC}}$ is defined as being consuming, masking, observing or producing.

The PDAG is composed of several transaction types, depending on whether it is a coinbase or a regular transaction, but also depending on the interaction with the shielded pool. The next definition maps transactions in a Zcash execution to transaction types supported in a PDAG.

Definition 5.7 (Transaction types)**.** A coinbase transaction is modeled as a MISO transaction. A regular t-to-t transaction is modeled as a SISO, SIMO, MISO or MIMO transaction depending on $|\text{ADDR}_{\text{IN}}|$ and $|\text{ADDR}_{\text{OUT}}|$. In contrast, t-to-z transactions are modeled either as SIMO or MIMO due to nullifiers output. As for z-to-z transactions, they consist of multiple inputs because of the zk-SNARK and multiple outputs due again to nullifiers. Such transactions may thus only be of MIMO type. Regarding z-to-t transactions, they constitute of multiple input due to the reading of nullifiers and thus are modeled as MISO or MIMO. Finally, we define the initialization transaction included in the genesis block as an INIT transaction of the form $t := (\{s_g, s, w\}, \{(s_g, w), (w, s)\})$, where $(s_g, w) \in \mathcal{E}_C, (w, s) \in \mathcal{E}_P$ and $s := (\text{ADDR}_m, 21M, \mathbf{H}(\{s_g, w\}), 0)$, where ADDR_m denotes a Zcash address that contains unmined ZEC.

We conclude our description of the Zcash context with the transaction predicate $\mathbb{P}$. For that purpose, we define a function $\text{VerifyContract}(\text{TADDR}, \text{CF})$ that takes a Zcash address TADDR as input and a confirmation CF and returns `true` if CF encodes a valid confirmation to spend the Zcash held at TADDR and otherwise returns `false`. Additionally, we define $\text{VerifyProof}(\mathcal{Z})$ that takes a zero-knowledge as input and returns true if (1) the sender proves knowledge of the private spending keys of the input notes, (2) the nullifiers are computed correctly, (3) the generated nullifiers do not collide with pre-existing nullifiers and (4) the sum of the inputs is equal to the sum of outputs. It returns `false` if any of these four conditions is not fulfilled. We also repeat the function $\text{VerifyWork}(\text{MP})$ defined in the previous section that returns `true` if MP is a valid Proof-of-Work.

Definition 5.8 (Transaction predicate in Zcash)**.** Consider a transaction $t := (\mathcal{S} \cup \mathcal{N} \cup \{w\}, \mathcal{E})$. Then, the *transaction predicate* $\mathbb{P}(t)$ returns `true` if the following conditions hold and `false` otherwise.

1. If t is a regular transparent transaction ($w.\mathit{txtype} = \text{TRTX}$), the witness holds a valid confirmation for each input state i.e.,
$$\forall s \in \star\mathcal{E}w, \exists\, \text{CF} \in w.\mathcal{F} : \text{VerifyContract}(s.\mathit{addr}, \text{CF}).$$
2. If t is a regular shielded transaction ($w.\mathit{txtype} = \text{ZRTX}$), the witness holds a valid zero-knowledge proof, i.e.,
$$\text{VerifyProof}(w.\mathcal{Z}).$$
3. If t is a coinbase transaction ($w.\mathit{txtype} = \text{CBTX}$), the witness contains a valid mining proof, i.e.,
$$w.\mathcal{F} := \{\text{MP}\} \wedge \text{VerifyWork}(\text{MP}).$$
4. Each output state in the transparent pool represents a positive number of ZEC, i.e., $\forall s \in w\mathcal{E}\star : s.\mathit{val} > 0$.

5. The sum of ZEC held at the input states in the transparent pool must be equal to the sum of ZEC held at the output states, i.e.,
$$\sum_{s \in \star\mathcal{E}w} s.val = \sum_{s' \in w\mathcal{E}\star} s'.val.$$

6. Each output state contains the evaluation of the hash function over consumed input states, input nullifiers and the witness, i.e.,
$$\forall s \in w\mathcal{E}\star : s.hash = \mathbf{H}\left(\star\mathcal{E}_C w \cup \mathcal{N}\mathcal{E}_O w \cup \{w\}\right).$$

Notation	Name
G_{ZEC}	Zcash TDAG
$\mathcal{V}_{\text{ZEC}}$	Zcash vertices
$\mathcal{E}_{\text{ZEC}}$	Zcash edges
$\mathcal{S}_{\text{ZEC}}$	Zcash states
$\mathcal{N}_{\text{ZEC}}$	Zcash nullifiers
$\mathcal{W}_{\text{ZEC}}$	Zcash witnesses
$(addr, val, hash, height)$	Zcash state with address, value, hash of previous vertices and block height
$\mathcal{L}_{\text{ZEC}}$	Zcash execution
$\mathcal{B}$	A set of blocks
ADDR	An address (either transparent or shielded)
TADDR	A transparent address
ZADDR	A shielded address
B_i	A block i
B_g	Genesis block
CBTX	Coinbase transaction
RTX	Regular transaction
TRTX	Regular transparent transaction
TRTX	Regular shielded transaction
MP	Mining proof
$\mathcal{Z}$	Zero-knowledge proof
ρ	Uniquely computed value for each output
$\mathcal{F}$	A set of confirmation
$\mathbf{H}(S')$	Evaluation of hash function over the set S'
$\text{VerifyWork}(\cdot)$	Boolean function for mining proof verification
$\text{VerifyProof}(\cdot)$	Boolean function for zero-knowledge proof verification

Table 5.2: Notations used to describe a Zcash execution.

5.3.3 Model analysis and execution sample

Now that we have defined a proper Zcash execution as well as a corresponding PDAG, we proceed further to prove that the above-specified model for Zcash respects the graph semantics imposed in chapter 3.

Lemma 5.2. Assume $\mathbf{H}(\cdot)$ is a collision-resistant hash function [27] and assume that $\mathcal{L}_{\text{ZEC}}$ is a legal Zcash execution. Then, the graph G_{ZEC} resulting from modeling $\mathcal{L}_{\text{ZEC}}$ is a PDAG.

Proof. We show that $G_{\text{ZEC}} = (\mathcal{S}_{\text{ZEC}} \cup \mathcal{N}_{\text{ZEC}} \cup \mathcal{W}_{\text{ZEC}}, \mathcal{E}_{\text{ZEC}})$ fulfils the conditions to be a PDAG according to definition 3.2.

1. The genesis state has only one consuming edge and no producing edges. It is the case due to the original design of the INIT transaction.

2. Every state, except for the genesis state, is produced exactly once. Assume, by contradiction, that this is not true and that a state $s \in \mathcal{S}_{\text{ZEC}}$ has two producing edges. That would imply that there exists two different sets $\mathcal{V}$ and $\mathcal{V}'$ such that $\mathbf{H}(\mathcal{V}) = \mathbf{H}(\mathcal{V}')$ or that two distinct witnesses w_1 and w_2 have produced overlapping

nullifiers. The first contradicts the assumption of collision resistance of $\mathbf{H}(\cdot)$. The latter contradicts the prerequisite of $\mathbb{P}$ with an invalid zero-knowledge proof.

3. The nullifier states have no consuming or masking edges by construction.
4. The graph must be weakly connected because transactions consumed a previously unconsumed state in the graph. Either a ZEC coin remains unspent or, when spent, is consumed. This is guaranteed by only accepting unique nullifiers.
5. The graph has no cycles. If there was a cycle, that would mean that two distinct transactions would produce the same state. That would imply that there exist two different transactions t' and t' that produce the same state. As seen before, this contradicts the fact that $\mathbf{H}(\cdot)$ is collision-resistant and that nullifiers are unique.

□

Following the above proof, we specify an example of a Zcash graph G_{ZEC}. To ease the illustration, we depict G_{ZEC} in three separated figures 5.3, 5.4 and 5.5. For the same purpose, we omit the producing edges in both figures 5.4 and 5.5, as well as state s_5 in figure 5.4. Additionally, we provide a high-level cryptocurrency flow in figure 5.6. In all the figures, the shielded pool is represented as a light grey area. The Zcash execution $\mathcal{L}_{\text{ZEC}} = \{B_g, B_1, B2\}$ is similar to the sample Bitcoin execution example of the original paper [15] with $B_g := (\emptyset, \{t_0\})$, $B_1 :=$ (MP, $\{t_1\}$) and $B_1 :=$ (MP$'$, $\{t_2, t_3, t_4\}$). We here detail the description of $G_{\text{ZEC}} := (\mathcal{S}_{\text{ZEC}} \cup \mathcal{N}_{\text{ZEC}} \mathcal{W}_{\text{ZEC}}, \mathcal{E}_{\text{ZEC}})$.

- The following three transactions are depicted in figure 5.3. The informal interpretation that we give here, is Alice first gets a reward of 50 ZEC for mining the first block. Second, she transfers 2 ZEC to Bob's shielded address and also provides a shielded address for her unspent amount.
  - $t_0 := (\{s_g, s_0, w_0\}, \{(s_g, w_0), (w_0, s_0)\})$, where $(s_g, w_0) \in \mathcal{E}_C$ and $(w_0, s_0) \in \mathcal{E}_P$. This $\mathcal{E}_{\text{ZEC}}$ transaction initialises the ledger with the genesis state $s_g := (\text{TADDR}_0, 21M, \mathbf{H}(\emptyset), 0)$, $w_0 :=$ (TINITX, $\emptyset$) and $s_0 := (\perp, 21M, \mathbf{H}(\{s_g, w_0\}), 0)$.
  - $t_1 := (\{s_0, w_1, s_1, s_2\}, \{(s_0, w_1), (w_1, s_1), (w_1, s_2)\})$, where $(s_0, w_1) \in \mathcal{E}_C$ and $\{(w_1, s_1), (w_1, s_2)\} \subseteq \mathcal{E}_P$. A SIMO transaction issues 50 ZEC to Alice as a reward for mining the block. That means, $w_1 :=$ (CBTX, MP) $s_1 := (\text{TADDR}_{\text{Alice}}, 50, \mathbf{H}(\{s_0, w_1\}), 1)$ and $s_2 := (\perp, 21M - 50, \mathbf{H}(\{s_0, w_1\}), 1)$, where $\text{TADDR}_{\text{Alice}}$ denotes a Zcash address owned by Alice. The notation remains the same for each subsequent transaction.
  - $t_2 := (\{s_1, w_2, s_3, s_4, s_5\}, \{(s_1, w_2), (w_2, s_3), (w_2, s_4), (w_2, s_5)\})$, where $(s_1, w_2) \in \mathcal{E}_C$ and $\{(s_1, w_2), (w_2, s_3), (w_2, s_4), (w_2, s_5)\} \subseteq \mathcal{E}_P$. This represents a t-to-z SIMO transaction, where Alice sends Bob 2 ZEC and her unspent amount to shielded addresses. The fee is here of 0.1 for simplicity. We consequently have $s_3 := (\text{ZADDR}_{\text{Alice}}, \text{Enc}_{k_{\text{Alice}}}(48), \mathbf{H}(\{s_1, w_2\}), 1)$, $s_4 := (\text{ZADDR}_{\text{Bob}}, \text{Enc}_{k_{\text{Bob}}}(1.9), \mathbf{H}(\{s_1, w_2\}), 1)$ and $s_5 := (\perp, 0.1, \mathbf{H}(\{s_1, w_2\}), 1)$. Since it is a t-to-z transaction, the spender is visible and the state consumption is as well, thus a consuming edge is used. A zero-knowledge proof is however used as part of the vJoinSplit to ensure that some properties are fulfilled, such as the balance of the inputs and outputs. We therefore have that $w_2 :=$ (ZRTX, $\mathcal{Z}$).

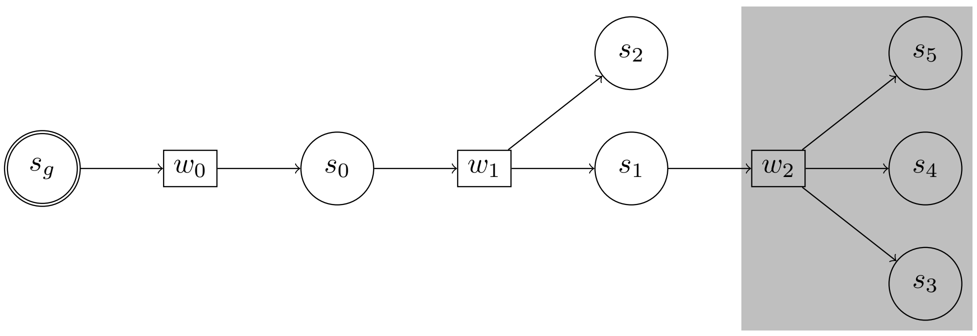


Figure 5.3: Zcash sample execution (1)

- We pursue our description inside the shielded pool, with Bob sending all his coins to Carol. Again for simplicity, a fee of 0.1 ZEC is collected. Here, the state consumption is obfuscated, and a nullifier is computed as a double-spending token. The last part of the execution comprises a deshielding transaction. Carol sends 1 ZEC to Dylan's transparent address while providing a shielded address for the rest amount. A fee of 0.1 ZEC is taken out of the transaction. Moreover, since a shielded output is spent, a nullifier is computed to make sure that nullifiers do not conflict with the existing ones – taken as input to the transaction.

- $t_3 := (\{s_4, s_3, w_3, n_0, s_7, s_6\}, \{(s_3, w_3), (s_4, w_3), (w_3, s_6), (w_3, s_7), (w_3, n_0)\})$ where $(s_3, w_3) \in \mathcal{E}_M$, $(s_4, w_3) \in \mathcal{E}_O$ and $\{(w_3, s_6), (w_3, s_7), (w_3, n_0)\} \subseteq \mathcal{E}_P$. This transaction is MIMO and private (z-to-z). The transacted amount is encrypted except for the fee. We have $s_6 := (\text{ZADDR}_{\text{Carol}}, \text{Enc}_{k_{\text{Carol}}}(1.8), \mathbf{H}(\{w_3\}), 1)$, $s_7 := (\perp, 0.1, \mathbf{H}(\{w_3\}), 1)$ and $n_0 := \{\rho_3\}$ with ρ_3 being the nullifier belonging to note s_3. We furthermore have the witness being $w_3 :=$ (ZRTX, $\mathcal{Z}'$)
- $t_4 := (\{s_3, s_4, n_0, s_6, w_4, s_8, s_9, s_{10}, n_1\}, \{(s_3, , w_4), (s_4, w_4), (s_6, w_4), (n_0, w_4), (w_4, s_8), (w_4, s_9), (w_4, s_{10}, w_4, n_1))\}$ with $\{(s_3), (w_4), (s_4), (w_4), (n_0), (w_4)\} \subseteq \mathcal{E}_O$, $s_6 \in \mathcal{E}_M$ and $\{(w_4), (s_8), (w_4), (s_9), (w_4), (n_1)\} \subseteq \mathcal{E}_P$. This represents a MIMO z-to-t transaction with four different states as input to the vJoinSplit. Note that s_7 is not part of it since there is no underlying address attributed to that particular state. We consequently have $s_8 := (\text{TADDR}_{\text{Dylan}}, 0.9, \mathbf{H}(\{n_0, w_4\}), 1)$, $s_9 := (\perp, 0.1, \mathbf{H}(\{n_0, w_4\}), 1)$, $s_{10} := (\text{TADDR}_{\text{Carol}}, \text{Enc}_{k_{\text{Carol}}}(0.8), \mathbf{H}(\{n_0, w_4\}), 1)$ and $n_1 := \{\rho_3, \rho_6\}$. We describe the witness with $w_4 :=$ (ZRTX), $(\mathcal{Z})''$

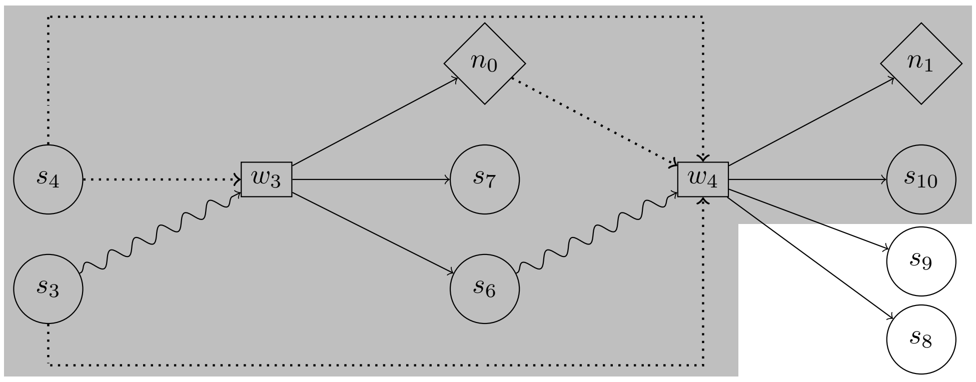


Figure 5.4: Zcash sample execution (2)

- Finally, the mining takes place with the fees and the ZEC supply as input to the coinbase transaction. The output is the fees and the mining reward attributed to the miner Maurice and the new ZEC supply.
  - $t_5 := (\{s_2, s_5, s_7, s_9, w_5, s_{11}, s_{12}\}, \{(s_2, w_5), (s_5, w_5), (s_7, w_5), (s_9, w_5), (w_5, s_{11}), (s_5, s_{12})\})$ where $\{(s_2, w_5), (s_5, w_5), (s_7, w_5), (s_9, w_5)\} \subseteq \mathcal{E}_C$ and $\{(w_5, s_{11}), (s_5, s_{12})\} \subseteq \mathcal{E}_P$. The states are $s_{11} := (\text{TADDR}_{\text{Maurice}}, 50.3, \mathbf{H}(\{s_2, s_5, s_7, s_9, w_5\}), 2)$ and $(\perp, 21M - 100, \mathbf{H}(\{s_2, s_5, s_7, s_9\}), 2)$ and the witness $w_5 :=$ (CBTX, MP$'$).

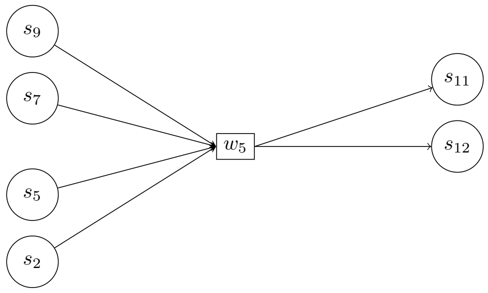


Figure 5.5: Zcash sample execution (3)

This example concludes the description of Zcash blockchain using the PDAG and it shows that the PDAG is effectively capable of representing such system. The next section takes care of describing an add-on privacy solution, namely Tornado Cash.

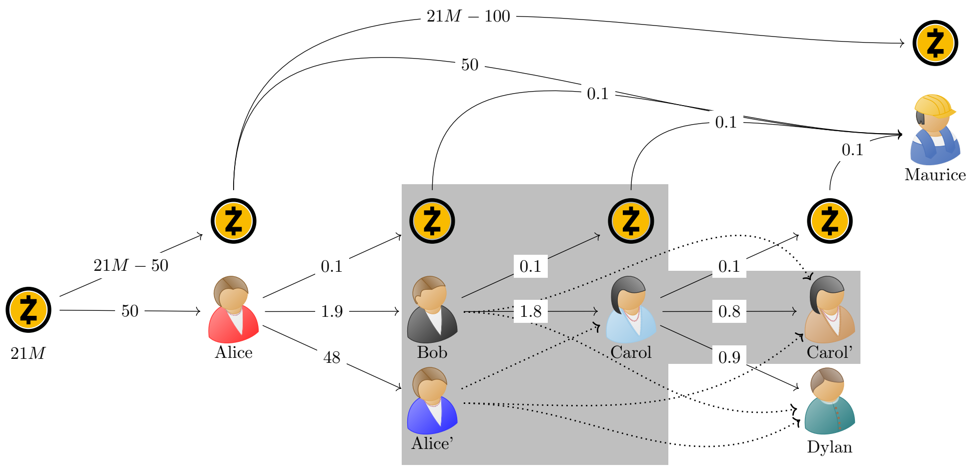


Figure 5.6: High-level cryptocurrency flow.

5.4 Tornado Cash

Finally, we take on the graph description of Tornado Cash, a cryptocurrency mixer that runs on Ethereum (and other compatible blockchains). We begin by describing the background of such technology, in order to specify the corresponding transaction graph. We finally proceed to provide examples of transactions using said mixer with our model. For that section, we base our explanation on multiple sources that we cite accordingly.

5.4.1 Background

As mentioned, Ethereum and other similar chains support Tornado Cash as a mixer. Thus, without loss of generality, we only provide a brief explanation of the Ethereum blockchain. Ethereum is one of the most prominent blockchains but has a different organization than its counterpart Bitcoin. Although both are distributed state machines, Ethereum tracks the state transitions of a general-purpose data store, whereas Bitcoin only tracks ownership of coins [3]. That flexibility allows Ethereum to support two types of accounts, externally owned accounts (EOA) and contract accounts (CA). The former effectively tracks the balance of the native cryptocurrency Ether (Abbreviation: ETH) held by each user. The latter, while still able to hold coins, store a piece of code, known as a smart contract, which executes a set of instructions when provided with a given input [14]. In addition, to invoke a smart contract, an external account needs to pay a fee (gas). Consequently, Ethereum supports more than coinbase and regular transactions. Transactions can indeed serve the purpose of transferring coins between two accounts or mining blocks, but the novelty here is that they also allow the creation of smart contracts or their invocation [14]. However, unlike Monero and Zcash, there exist no shielded transactions in the Ethereum blockchain.

Mixers partially overcome that limitation by allowing users to deposit funds and blend them with others, effectively breaking the link between several transactions. Tornado Cash is one of such mixer centered around pools. Pools are smart contracts that support two types of operations, namely *deposit* and *withdrawal*. A user can deposit tokens from one address and then, later on, withdraw those tokens to a separate address. Any link between the deposit and withdrawal is broken, even though those operations take place openly on Ethereum's transparent ledger. One key aspect here is that Tornado Cash pools are non-custodial by design, meaning users maintain ownership of their tokens throughout the entire process. Such guarantees are provided by zk-SNARKs also used as an obfuscation mechanism in the shielded pool of Zcash – as explained in section 5.3. When it comes to Tornado Cash, the prover is the user withdrawing tokens and the verifier is the pool contract. When clients deposit tokens into the pool, they generate a deposit note, and supply it in hashed form to the pool, along with the tokens. The pool records the different entries of users in a public list of hashed notes. At some point, users may decide to withdraw their funds. To that purpose, users provide a cryptographic proof certifying that the client issued a deposit without revealing precisely which one [32]. The user also supplies a unique commitment that the contract in turn records in a public list. This commitment ensures that the same tokens cannot be withdrawn again. During the process, the only information openly visible are therefore the addresses involved in those operations, but there is no data tying depositors to withdrawers any more. According to our definitions, this mechanism provides unlinkability since it breaks the link between multiple states. However, the state consumption and the transacted amount are openly visible and thus untraceability and confidentiality are two properties that Tornado Cash does not provide.

This mixer has been extremely popular over the years owing to its unlinkability guarantees. Multiple billions of dollars have gone through the mixer [56], allegedly including more than $455 million stolen by the Lazarus Group, a hacking organization supported by the Democratic People's Republic of Korea [51]. The U.S. Department of the Treasury, therefore, took measure by banning Tornado Cash on 8 August 2022, making it unlawful for American individuals, residents, and businesses to receive or transfer money using the service [52].

Even though Tornado Cash has been made illegal in one country, we still think that it is worth being studied. We thus go further in its analysis in the next two subsections by first providing a transaction graph and second by giving sample transactions involving the smart contract.

5.4.2 Transaction graph

As briefly mentioned in the previous subsection, Ethereum supports multiple types of transactions. Transfer transactions convey coins between (external or contract) accounts, and mining transactions have the purpose of creating new blocks. Moreover, transactions may be conducted to create a contract and other transactions allow to invoke one. Regardless of its type, a transaction is valid if

1. it contains a valid signature from the sender of the transaction and
2. all the externally owned accounts receive a non-negative amount of ETH.

A transaction is the building block of a PDAG, thus introducing our first definition that we extend from the original one [15]. The main differentiating factor here are the nullifiers that encapsulate the withdrawal commitments of Tornado Cash. We therefore specifically define this transaction graph for Ethereum *including Tornado Cash.*

Definition 5.9 (Transaction graph for Ethereum including Tornado Cash)**.** We model an execution of Ethereum system $\mathcal{L}_{\text{ETH}}$ including Tornado Cash as a *transaction graph* $G_{\text{ETH}} := (\mathcal{S}_{\text{ETH}} \cup \mathcal{N}_{\text{ETH}} \cup \mathcal{W}_{\text{ETH}}, \mathcal{E}_{\text{ETH}})$ defined as follows:

- **State**. Each state $s \in \mathcal{S}_{\text{ETH}}$ is represented by a tuple $(\mathit{sid}, \mathit{stype}, \mathit{sval})$ where
  - *sid* is a tuple $(\mathit{account}, \mathit{blockHeight})$ where *account* denotes an account in Ethereum and *blockHeight* denotes the position of the block in the Ethereum ledger that contains a transaction that has modified the *account* for the last time;
  - *stype* is the type of state, either being *externalAccount* or *contractAccount*;
  - *sval* represents the value associated to this state depending on the type. If the state is an external account, *sval* denotes the amount of ETH held at this state. Otherwise, if the state is a contract account, *sval* is defined as a tuple $(\mathit{bytecode}, \mathit{gasPrice}, \mathit{funds})$ with *bytecode* being the bytecode of the contract, *gasPrice* the amount of Ether required to execute the contract and *funds* the amount of ETH held by the smart contract.

  We furthermore consider a genesis state to model the initial state from where all Ether used in an instance of the Ethereum ledger are created. We denote this particular state $s_g \in \mathcal{S}_{\text{ETH}}$ by the tuple $((v_{\text{genesis}}, 0), \mathit{externalAccount}, x)$ where x denotes the total amount of Ether used in a snapshot of the Ethereum ledger. This does not fix a limit to the total amount of Ether supply, but only to the snapshot.
- **Witness**. Each witness $w \in \mathcal{W}_{\text{ETH}}$ is represented by a tuple $(\mathit{gasLimit}, \mathcal{F})$, where *gasLimit* denotes the maximum amount of Ether to be spent in the execution of a contract and $\mathcal{F}$ contains the set of fulfillment
- **Nullifier**. Each nullifier $n \in \mathcal{N}_{\text{ETH}}$ is a set of commitments $\{c_t, \dots, c_{t+i}\}$ for transaction t to $t + i$.
- **Edge**. Each edge $e \in \mathcal{E}_{\text{ETH}}$ linking a state to a witness can be defined as either producing or consuming edge. Moreover, nullifiers are read by withdrawal transactions involving Tornado Cash. We do not consider masking edges since the state consumption is always visible.

We also specify the validity of transactions per definition 3.5 via the below predicate that is likely unmodified from a TDAG [15]. To abstract away some processes, we define $\mathrm{VerifySender}\,(\mathit{sender}, \mathcal{F})$ that returns `true` if the confirmation $\mathcal{F}$ contains a valid signature from *sender*.

Definition 5.10 (Transaction predicate in Ethereum [15])**.** Consider a transaction $t := (\mathcal{V} \cup \mathcal{N} \cup \{w\}, \mathcal{E})$. Then, the *transaction predicate* $\mathbb{P}(t)$ returns `true` if the following conditions hold and `false` otherwise.

1. The witness must contain a signature from the sender of the transaction. Let **sender** be the $s.\mathit{sid.account}$ field for an arbitrary state s. Then

$$\forall s \in \star\mathcal{E}_C w : \mathrm{VerifySender}\left(\mathbf{sender}, w.\mathcal{F}\right)$$

2. All the externally owned accounts must receive a non-negative amount of Ether.

$$\forall s \in \star\mathcal{E}_C w|_{s.\mathit{type} = \mathit{externalAccount}} : s.\mathit{val} \geq 0$$

3. No ETH must be created or destroyed in a regular transaction.

$$\sum_{s \in \star\mathcal{E}_C w|_{s.\mathit{type} = \mathit{externalAccount}}} s.\mathit{val} = \sum_{s' \in w\mathcal{E}_P\star|_{s'.\mathit{type} = \mathit{externalAccount}}} s'.\mathit{val}$$

4. The gas limit provided in the transaction is enough to execute the contract.

$$\exists s \in w\mathcal{E}_P \star|_{s.\mathit{stype} = \mathit{contractAccount}} \implies w.\mathit{gasLimit} \geq s.\mathit{sval.gasPrice}$$

Now that we have defined the fundamental pieces of an Ethereum execution, we describe sample transactions, extended to the PDAG model, involving Tornado Cash in the next subsection.

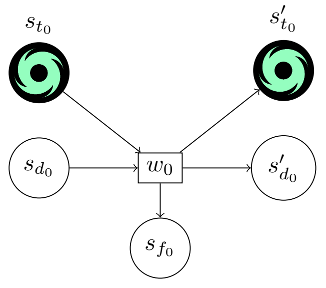
5.4.3 Sample transactions

In figure 5.7, we depict three different transactions involving Tornado Cash. The mixer is illustrated with the official logo (the green tornado) and the name of the state on top. The subfigure 5.7a represents a deposit, which is in fact a simple invocation of the smart contract. However, the very first withdrawal illustrated on subfigure 5.7b creates the nullifier set with a size of 1. Subsequent withdrawals read the existing nullifier set(s) and produce a new one, as shown in subfigure 5.7c. We proceed here with a PDAG description of these three sample transactions, while still giving a concrete interpretation. We assume an incremental block height here beginning with the first transaction at 1 and with the third at 3. Furthermore, we fix the deposit and withdrawal gas fees of Tornado Cash to 1 ETH for simplicity.

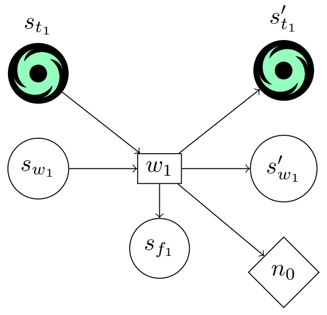
- $t_0 := \left(\left\{s_{t_0}, s_{d_0}, s'_{t_0}, s'_{d_0}\right\}, \left\{(s_{t_0}, w_0), (s_{d_0}, w_0), \left(w_0, s'_{t_0}\right), \left(w_0, s'_{d_0}\right)\right\}\right)$, where $\{(s_{t_0}, w_0), (s_{d_0}, w_0)\} \subset \mathcal{E}_C$ and $\left\{\left(w_0, s'_{t_0}\right), \left(w_0, s'_{d_0}\right)\right\} \subset \mathcal{E}_P$. A MIMO transaction takes place for Alice to deposit 20 ETH (out of her 50 ETH) into the mixer. The states and witnesses are described as follows:
  - $s := ((\mathit{account}, \mathit{blockHeight}), \mathit{contractAccount}, (\mathit{bytecode}, \mathit{gasPrice}, \mathit{funds}))$
  - $s_{t_0} := ((\mathrm{ADDR}_{\mathit{Tornado}}, 1), \mathit{contractAccount}, (\mathit{bytecode}_{\mathrm{Tornado}}, 1, 0))$,
  - $s_{d_0} := ((\mathbf{vk}_{\mathrm{Alice}}, 1), \mathit{externalAccount}, 50)$,
  - $s'_{t_0} := ((\mathrm{ADDR}_{\mathit{Tornado}}, 1), \mathit{contractAccount}, (\mathit{bytecode}_{\mathrm{Tornado}}, 1, 20))$,
  - $s'_{d_0} := ((\mathbf{vk}_{\mathrm{Alice}}, 1), \mathit{externalAccount}, 29)$,
  - $s_{f_0} := ((\perp, 1), \perp, 1)$, and
  - $w_0 := (1, \mathrm{CF})$,

  where CF is a set of confirmations including the signature from the sender. We assume here that not only Alice but also Bob does a similar deposit, resulting in Tornado's state, outlined below:
  - $s''_{t_0} := ((\mathrm{ADDR}_{\mathit{Tornado}}, 1), \mathit{contractAccount}, (\mathit{bytecode}_{\mathrm{Tornado}}, 1, 40))$.

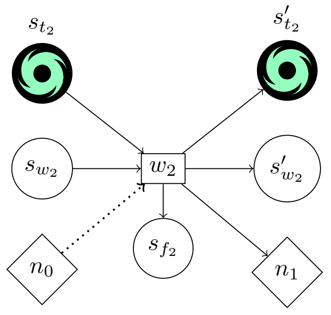
- $t_1 := (\{s_{t_1}, s_{w_1}, s'_{t_1}, s'_{w_1}, n_0\}, \{(s_{t_1}, w_1), (s_{w_1}, w_1), (w_1, s'_{t_1}), (w_1, s'_{w_1}, (w_1, n_0))\})$, where $\{(s_{t_1}, w_1), (s_{w_1}, w_1)\} \subset \mathcal{E}_C$ and $\{(w_1, s'_{t_1}), (w_1, s'_{w_1}), (w_1, n_0)\} \subset \mathcal{E}_P$. This MIMO transaction is the withdrawal from an unlinked individual, say, Ursula, with a fresh address. In this case, a nullifier is produced to store the commitment of Ursula's withdrawal. We hereby describe the different states, the witness, and the nullifier at stake.
  - $s := ((\mathit{account}, \mathit{blockHeight}), \mathit{contractAccount}, (\mathit{bytecode}, \mathit{gasPrice}, \mathit{funds}))$
  - $s_{t_1} := ((\mathrm{ADDR}_{\mathit{Tornado}}, 2), \mathit{contractAccount}, (\mathit{bytecode}_{\mathrm{Tornado}}, 1, 40))$,
  - $s_{w_1} := ((\mathbf{vk}_{\mathrm{Ursula}}, 2), \mathit{externalAccount}, 0)$,
  - $s'_{t_1} := ((\mathrm{ADDR}_{\mathit{Tornado}}, 2), \mathit{contractAccount}, (\mathit{bytecode}_{\mathrm{Tornado}}, 1, 20))$,
  - $s'_{w_1} := ((\mathbf{vk}_{\mathrm{Ursula}}, 2), \mathit{externalAccount}, 19)$,
  - $s_{f_1} := ((\perp, 2), \perp, 1)$,
  - $n_0 := \{c_{s_{w_1}}\}$ and
  - $w_1 := (1, \mathrm{CF})$.
- $t_2 := (\{s_{t_2}, s_{w_2}, n_0, s'_{t_2}, s'_{w_2}, n_1\}, \{(s_{t_2}, w_2), (s_{w_2}, w_2), (w_2, s'_{t_2}), (w_2, s'_{w_2}, (w_2, n_0))\})$, where $\{(s_{t_2}, w_2), (s_{w_2}, w_2)\} \subset \mathcal{E}_C$, $\{(w_2, s'_{t_2}), (w_2, s'_{w_2}), (w_2, n_1)\} \subset \mathcal{E}_P$ and $(n_0, w_2) \in \mathcal{E}_O$. This MIMO transaction is a second withdrawal from an unlinked individual, this time Ulrich, with – again – a fresh address. In this case, a nullifier is read to check if Ulrich does not attempt to withdraw a previously withdrawn deposit. A nullifier is also in turn produced to store the commitment of Ulrich's withdrawal. The different vertices are detailed below.
  - $s := ((\mathit{account}, \mathit{blockHeight}), \mathit{contractAccount}, (\mathit{bytecode}, \mathit{gasPrice}, \mathit{funds}))$
  - $s_{t_2} := ((\mathrm{ADDR}_{\mathit{Tornado}}, 3), \mathit{contractAccount}, (\mathit{bytecode}_{\mathrm{Tornado}}, 1, 20))$,
  - $s_{w_2} := ((\mathbf{vk}_{\mathrm{Ursula}}, 3), \mathit{externalAccount}, 0)$,
  - $s'_{t_2} := ((\mathrm{ADDR}_{\mathit{Tornado}}, 3), \mathit{contractAccount}, (\mathit{bytecode}_{\mathrm{Tornado}}, 1, 0))$,
  - $s'_{w_2} := ((\mathbf{vk}_{\mathrm{Ursula}}, 3), \mathit{externalAccount}, 19)$,
  - $s_{f_2} := ((\perp, 3), \perp, 1)$,
  - $n_1 := \{c_{s_{w_1}}, c_{s_{w_2}}\}$ and
  - $w_2 := (1, \mathrm{CF})$.



(a) An example of a deposit.



(b) An example of a very first withdrawal.



(c) An example of subsequent withdrawals.

Figure 5.7: Three examples of transactions involving Tornado Cash.

In such a scenario, an adversary may link any withdrawer to the n depositors with a probability of $\frac{1}{n}$. For instance, the probability of linking Alice and Bob to either Ursula or Ulrich is of $\frac{1}{2}$. This is the very purpose of Tornado Cash: obscure the link between multiple states. However, as described in section 4.4, an adversary may use heuristics to increase her chance to guess which states belong to the same entity. This means that she may identify a subset of k

depositors with $k < n$, to link a withdrawer with a chance of $\frac{1}{k} > \frac{1}{n}$. In the next section, we concretely explore some of these heuristics.

5.5 Using PDAG to Unify Deanonymisation Heuristics

In the following lines, we describe some common heuristics with an informal explanation as well with the PDAG notation. We will talk about states, witnesses, and edges to remain close to the graph abstraction but, to avoid confusion, keep the notion of address when necessary.

Naturally, heuristics are usually not universal to all types of blockchains. In UTXO-based systems (e.g., Bitcoin or unshielded Zcash), the same heuristics cannot be fully applied to account-based (e.g., Ethereum) or to – as we came to call it in chapter 3 – nullifier-based systems (e.g., Monero or shielded Zcash). We will expressively indicate which heuristics apply to which model in the following. Also, for clarity, we break down our explanations for each heuristic into paragraphs. We begin by recalling the trivial heuristic.

Trivial linking. *Trivial linking* happens when two states have the same underlying address. Most blockchains that do not perform address change by design – e.g., Bitcoin and Ethereum – are subject to this heuristic. It, therefore, links together all the states with the same address as follows:

$$\forall s_i, s_j \in \mathcal{S} : \mathbf{addr}\left(s_i, s_j\right) \implies \mathbf{link}\left(s_i, s_j\right). \tag{5.1}$$

Now, users tend to have multiple different addresses to break the link between several transactions they conduct. For that reason, in UTXO-based blockchains, considerable work has studied address clustering heuristics [58, 38, 23].

Input states grouping. One of the most common assumptions is that every consuming input state to a witness belongs to the same entity, thus *grouping input states* together. This is motivated by the fact that spending an amount requires the private key of the address and hence to be "in control" of the state. That statement translates to

$$\forall s_i, s_j \in \star\mathcal{E}_C w : \mathbf{link}\left(s_i, s_j\right). \tag{5.2}$$

However, CoinJoin transactions represent an exception to this assumption. This mechanism involves a group of users that creates a single transaction that simultaneously spends all their inputs into a (mixed) list of outputs. Thus, the input states cannot simply be linked together.

Coinjoin transactions detection Although such a mechanism makes the previous heuristic invalid, *CoinJoin transactions* can also be *detected*. We can indeed devise a heuristic that assumes that MIMO transactions with the same amount use this mechanism. States involved in such transactions are thus not linked [22, 25]. Here, the amount is crucial, since it must match each input and output state. In the transaction graph, the balance is encapsulated in each state, but the transacted amount is by itself not represented. To simplify the notations, we define the amount function $\mathbf{A}\left(s_i, s_j\right)$ that takes state s_i and s_j as input and returns the transacted amount γ. We can thus translate the Coinjoin heuristic more concisely. All the input states to a MIMO transaction with the same amount are not linked:

$$\forall s_i, s_j \in \star\mathcal{E}_C w_1, s_k, s_l \in w_2 \mathcal{E}_P S_O : w_1 = w_2 \land |S_O| \geq 2 \land \mathbf{A}\left(s_i, s_k\right) = \mathbf{A}\left(s_j, s_l\right) \implies \overline{\mathbf{link}}\left(s_i, s_j\right). \tag{5.3}$$

Unspent state detection. Another typical heuristic for UTXO blockchain is to assume that the *amount surplus* is sent back to the initial sender, i.e., that the produced state for the change belongs to the consumed state. That makes sense considering that coins usually cannot be spent partially. Which of the output state is really the one getting the change is unclear, thus the use of heuristics. The equation below describes the aforementioned heuristic,

$$\forall s_i \in \star\mathcal{E}_C w, \exists s_j \in w\mathcal{E}_P \star : \mathbf{link}\left(s_i, s_j\right). \tag{5.4}$$

As for account-based systems, we can reuse heuristic 5.2 since the same logic applies. The second one, however, is not relevant as there is no unspent state production in such systems. Victor [54] highlights a few different heuristics in his work [54].

Deposit address reuse. The first one, *deposit address reuse*, relies on the fact that crypto-assets create deposit addresses where users can transfer their coins. These addresses are usually created per customer, so multiple different addresses that send funds to the same deposit address may in fact refer to the same user [54]. We give the following equation to actually describe that scenario,

$$\forall s_i \in \star\mathcal{E}_C w_1 \mathcal{E}_P s_{d_1}, s_j \in \star\mathcal{E}_C w_1 \mathcal{E}_P s_{d_2} : \mathbf{addr}\left(s_{d_1}, s_{d_2}\right) \implies \mathbf{link}\left(s_{d_1}, s_{d_2}\right) \wedge \mathbf{link}\left(s_i, s_j\right). \tag{5.5}$$

Airdrop multi-participation. The *airdrop multi-participation* heuristic addresses a token airdrop where the multiple states beneficiaries tend to get merged into a single one after the airdrop, most likely for convenience [54]. The amount is – like in heuristic 5.3 – decisive here. For each merging transaction, the amount must match one of the airdrops for the heuristic to be valid. With the amount function defined above, we can translate the airdrop heuristic, into

$$\forall s_{a_1}, s_{a_2} \in \overbrace{w_1 \mathcal{E}_P \star}^{\text{airdrop}} \underbrace{\mathcal{E}_C w_2 \mathcal{E}_P s_d}_{\text{merging transaction}} : \mathbf{A}\left(s_{a_1}\right) = \mathbf{A}\left(s_{a_2}\right) \implies \mathbf{link}\left(s_{a_1}, s_{a_2}\right) \wedge \mathbf{link}\left(s_{a_1}, s_d\right) \wedge \mathbf{link}\left(s_{a_2}, s_d\right). \tag{5.6}$$

In addition, Ethereum possesses one mechanism to authorize accounts to spend tokens on behalf of another one. To abstract this mechanism, we define an $\mathbf{approve}\left(s_i, s_j, a\right)$ predicate that returns `true` if s_i has approved s_j to spend a tokens on its behalf.

Self-authorisation. Victor [54] makes the assumption of *self-authorization*, i.e., if one state approves another one, the two must belong to the same entity. Such behavior happens in the blockchain for either test purposes or risk distribution over several addresses [54]. We specify the aforementioned heuristic, as

$$\forall s_i, s_j \in \mathcal{S}, a \in \mathbb{R} : \mathbf{approve}\left(s_i, s_j, a\right) \implies \mathbf{link}\left(s_i, s_j\right). \tag{5.7}$$

One other mechanism to increase unlinkability in account-based blockchains is the use of mixing services. Mixing services allow users to mix their coins with other users in order to break links between addresses. Mixers operate through multiple smart contracts that accept different quantities of coins as deposits. When a user places coins, she must back up a deposit note to later be able to withdraw the original amount. Zero-knowledge proofs are usually used to ensure that the link between the depositor and the withdrawer remains hidden. Mixers are thus way more powerful in providing unlinkability than simple address duplication. Research has also studied mixers to devise heuristics in order to reconstruct links between states. We here give some of these heuristics outlined in the work by Wang et al. [56] [56].

The most obvious heuristic that can be devised first is the one where a user deposits and withdraws with the same address. That heuristic is in fact a direct implication of the trivial heuristic 5.1.

Subsequent transaction linking. We find a yet less trivial linking heuristic, where the withdrawal address a_w is used to perform *subsequent transactions* towards the initial deposit address a_d. So, after the withdrawal, there is a linking transaction from a_w back to a_d. We describe this heuristic in the following formula. For that we define the set of states that correspond to mixers as $\mathcal{S}_c$:

$$\forall s_d, s_{c_1} \in w_d \mathcal{E}_P \star, s_w, s_{c_2} \in w_w \mathcal{E}_P \star : \underbrace{s_w \mathcal{E}_C w \mathcal{E}_P s_d}_{\text{subsequent transaction}} \wedge \mathbf{addr}\left(s_{c_1}, s_{c_2}\right) \wedge\ s_{c_1}, s_{c_2} \in \mathcal{S}_c \implies \mathbf{link}\left(s_d, s_w\right). \tag{5.8}$$

Users are therefore encouraged to use a different address than their usual one to perform the deposit. They create several addresses to which they transfer coins from their regular address. Those addresses are later used for the deposit.

Direct coin transfer. However, these *direct coin transfers* are observable. They may in fact connect these deposit addresses to the original one(s). That means that the deposit states are actually at a distance from one witness to the original state. This pattern leads to the following heuristic:

$$\forall s_i \in \overbrace{\star\mathcal{E}_C w \mathcal{E}_P s_d}^{\text{direct coin transfer}} : \underbrace{s_d \mathcal{E}_C w_d \mathcal{E}_P s_c}_{\text{deposit}} \wedge s_c \in \mathcal{S}_c \implies \mathbf{link}\left(s_i, s_d\right). \tag{5.9}$$

Mixers set symmetry. Furthermore, mixers usually accept a fixed amount of coins, so multiple mixers must sometimes be combined to accumulate the desired deposit. Consequently, one address that is depositing to a fixed set of mixers and another address withdrawing from the very same set might well be linked.

$$\forall s_i \in \overbrace{\star\mathcal{E}_C w_d \mathcal{E}_P \mathcal{S}_c}^{\text{deposit}}, s_j \in \underbrace{\mathcal{S}_c \mathcal{E}_C w_w \mathcal{E}_P \star}_{\text{withdrawal}} : |\mathcal{S}_c| \geq 2 \implies \mathbf{link}\left(s_i, s_j\right) \tag{5.10}$$

A step further into unlinkability is an anonymity pool, which consists of a portion of the blockchain that follows the semantics of nullifier-based ones. Such a pool can be described as a subgraph G' of G, rather than, only one witness w for, the mixers. In the following, we use $G_z = (\mathcal{S}_z \cup \mathcal{N}_z \cup \mathcal{W}_z, \mathcal{E}_z)$ to indicate the subgraph that belongs to the pool and $G_t = (\mathcal{S}_t \cup \mathcal{W}_t, \mathcal{E}_t)$ the transparent one such that $G = G_z \cup G_t$. We refer to such systems as *hybrid* ones.

Zcash is a typical example of a blockchain that is composed of a shielded pool of transactions on top of a regular UTXO system. Kappos et al. [26] devise some heuristics that apply to a subgraph G_z.

Founders' reward. The first heuristic they devise is quite specific to Zcash where the initial founders of the blockchain get 20% of newly generated coins. This ratio is called the *founders' reward.* Per transaction, that means 2.5 ZEC out of the 12.5 ZEC mining fee. The authors notice that a transaction from G_z to G_t is often the same and from which the amount corresponds roughly to the reward for 10 blocks, i.e., 250 ZEC. They come to the conclusion that a transaction coming out of the pool, carrying more than around 250 ZEC in value, is done by the founders [26]. Yet, this heuristic is likely, not applicable any more, or at least not with the value of 250, since founders do not receive 20% of the miner's fee as of today. A fifth of the fee is instead divided between the new Major Grants Fund (8%), Electric Coin Co (7%), and the Zcash Foundation (5%) [11]. We define $\mathcal{S}_f$ to be the set of founders and v a generalized threshold value instead of the possibly outdated 250 ZEC. This statement, therefore, translates to

$$\forall s_i \in G'_z w \mathcal{E}_P \star G'_t, s_f \in \mathcal{S}, v \in \mathbb{R} : \mathbf{A}\left(s_i\right) \geq v \implies \mathbf{link}\left(s_i, s_f\right). \tag{5.11}$$

Mining pool retribution Another pattern that Kappos et al. [26] describe in their work refers to the *mining pool retribution* process. Usually, the mining reward of a pool is paid into a single address, controlled by the operator of the pool, and the pool operator then deposits some set of aggregated rewards into the shielded pool [26]. What happens next is that they pay each individual miner from G_z to G_t, resulting in a lot of output transactions. The authors conclude from this behavior, that if a transaction from the shielded pool to the outside has over 100 output states, one of which belongs to a known mining pool, all the states belong together. We denote by $\mathcal{S}_m$ the set of miner states and by k a generalized value for the number of outputs instead of 100 to propose the following translation in PDAG language:

$$\forall s_i, s_m \in G'_z w \mathcal{E}_P \star G'_t : |\star| \geq k \wedge s_m \in \mathcal{S}_m \implies \mathbf{link}\left(s_i, s_m\right). \tag{5.12}$$

Round-trip transaction. Still, in hybrid systems, Quesnelle [45] identifies so-called *round-trip transactions*, where an equivalent amount appears to enter the pool and leaves it soon after. The notion of time here is given by a certain amount of blocks by Quesnelle and also Kappos et al. [26]. We here take the liberty of formulating this notion of time by the size of witnesses of the pool $\mathcal{W}_p$. Moreover, we define two vectors A_I and A_O which correspond to amounts entering the pool and respectively leaving it, i.e., $A_I = [\mathbf{A}\left(s_i, s_j\right) \mid \forall s_i \in \mathcal{S}_t, s_j \in \mathcal{S}_p] = [a_{i,0} \dots a_{i,k}]$ and $A_I = [\mathbf{A}\left(s_i, s_j\right) \mid \forall s_i \in \mathcal{S}_p, s_j \in \mathcal{S}_t] = [a_{o,1} \dots a_{o,l}]$ for values $k, l \in \mathbb{N}$. We denote a value v being a coordinate of said vectors with the notation $v \in A$ and obtain the heuristic,

$$
\begin{aligned}
\forall s_i \in \underbrace{G'_t \star \mathcal{E}_C w G'_z}_{\text{input transaction}}, s_j \in \underbrace{G''_z w_k \mathcal{E}_P \star G''_t}_{\text{output transaction}}, \exists ! a \in A_I, b \in A_O :& \\
\mathbf{A}\left(s_i, \cdot\right) = a \wedge& \\
\mathbf{A}\left(s_j, \cdot\right) = b \wedge& \\
\underbrace{a \approx b}_{\text{amount matching}} \quad \wedge \left|\mathcal{W}_p\right| \leq k& \\
\implies \mathbf{link}\left(s_i, s_j\right).&
\end{aligned}
\tag{5.13}
$$

Now, contrary to our assumption about $\mathcal{H}^{\Pi}$, the list of heuristics summarised in table 5.3 is probably not exhaustive, but it gives a good summary of the different linking patterns studied in the literature. However, to the best of our knowledge, there exists no heuristic for nullifier-based blockchains. This can be mostly explained by the fact that these blockchains have no clear state consumption. Consequently, we assume that the lack of heuristics for such systems implies that they achieve unlinkability. We present our results in the form of comparison in the next section.

Equation	Name	Assumption	Domain	Reference
5.1	Trivial linking	All the states with the same underlying address are linked.	UTXO, account	-
5.2	Input states grouping	All the input states to a witness are linked.	UTXO, account	[40, 2, 36]
5.3	Coinjoin transaction detection	All the input states in a MIMO transaction where the amount matches are part of a Coinjoin transaction. States are thus **not** linked.	UTXO, account	[22, 25]
5.4	Unspent state detection	The unspent address belongs to one of the spender address.	UTXO	[36]
5.5	Deposit address reuse	The addresses assigned to users by an exchange are reused. Thus all the addresses that transfer coins to this address are linked with each other and to the assigned address.	account	[54]
5.6	Airdrop multi-participation	The addresses that are used for an airdrop are eventually merged into the main user address. All these addresses are linked.	account	[54]
5.7	Self authorisation	One state approving another one to spend any amount is linked to that very state.	account	[54]
5.8	Subsequent transaction linking	If a withdrawal address makes a transaction back to a deposit address; they belong to the same entity.	account	[56]
5.9	Direct coin transfer	The address that transfers to multiple deposit addresses is linked to them.	account	[56]
5.10	Mixers set symmetry	Two addresses that deposit and withdraw the same amount from the same set of mixers are linked.	account	[56]
5.11	Founders' amount	If an input transaction to the shielded pool exceeds a certain amount, the states involved belong to the founders.	hybrid	[26]
5.12	Mining pool retribution	If a transaction out of the shielded pool has multiple output states and one is known to be of a miner, then all the output states are of miners.	hybrid	[26]
5.13	Round-trip transaction	If there exists one transaction to the shielded pool and one – not too long after – subsequent transaction out of the pool with nearly the same amout, then those transactions are linked and so are the states involved.	hybrid	[45, 26]

Table 5.3: Summary of different heuristics

5.6 Comparisons

We end this thesis with a section comparing the blockchain systems discussed in this chapter. We take on the same order as in the previous sections, beginning with Bitcoin and then pursuing with Monero, Zcash and Tornado.

Bitcoin. Regarding Bitcoin, we consider the worst-case scenario for an adversary, where users utilize mechanisms to enhance their privacy. We, therefore, assume, as Biryukov and Feher [7] in their work that (1) new addresses are used to collect the change amount, (2) users own multiple addresses and (3) users frequently change their addresses. Still, with those measures, 40% of users remain vulnerable to linkability attacks [7]. We note, however, that users could use a coin mixer to enhance their privacy by invalidating the input state grouping heuristic. Yet, counter-heuristics exist to cope with that mechanism. We think, therefore, that the Bitcoin unlinkability property is not achieved in Bitcoin due to the vulnerability to the trivial heuristic but – with correct user behavior – is still achievable to some extent. We, therefore, believe that Bitcoin in its standard form does not provide unlinkability ($\epsilon = 1$) but may, to some extent, guarantee unlinkability properties with address changes and Coinjoin, i.e., $1 < \epsilon < \mathcal{O}\left(|\mathcal{S}|^2\right)$. As for untraceability and confidentiality, Bitcoin does not offer any built-in or add-on feature to provide such properties ($\epsilon = 1$).

Monero. Concerning Monero, this blockchain has built-in privacy mechanisms that aim to provide all three privacy notions studied in this thesis. For each transaction, a ring signature of size eleven is now mandatory, thus providing untraceability. However, as Egger et al. [19] note, this size is too small to provide acceptable anonymity against decomposition attacks. Vijayakumaran [55] conversely, show that only 0.24% of the total rings are traceable (in a given block interval), so although the size of the ring signature is theoretically not sufficient ($1 < \epsilon < \mathcal{O}\left(\frac{1}{k}\right)$, Monero seems empirically robust to decomposition attacks. As for unlinkability, Monero enforces address change at each transaction by design. This mechanism is quite effective at blurring the link between stealth addresses. We furthermore think that the lack of heuristics mirrors how robustly Monero enhances unlinkability. According to our definition, since there exists no linking heuristic for Monero, we think that an adversary can only use random guessing over the whole set of states to effectively jeopardize unlinkability, i.e., $\epsilon \leq \mathcal{O}\left(|\mathcal{S}|^2\right)$. Finally, RingCT provides confidentiality through encryption ($\epsilon \leq \mathsf{negl}\left(n\right)$) but also helps in promoting untraceability by increasing the traceability set.

Zcash. Zcash, in contrast, provides both untraceability and confidentiality via zero-knowledge SNARK in the shielded pool (z-to-z). That mechanism allows for a much larger traceability set than its counterpart, Monero, since it contains any potential shielded output. Thus, according to our definitions, in z-to-z and z-to-t transactions, an adversary is not likely to perform better than random guessing, i.e., $\epsilon \leq \mathcal{O}\left(\frac{1}{k}\right)$, given a subgraph of the entire blockchain. Yet, outside this pool, in a t-to-t transaction, Zcash functions much like Bitcoin and provides very little privacy. Moreover, in a t-to-z transaction and in a z-to-t transaction, the transaction is not confidential since the transacted amount appears in plain in the transparent outputs. Also, due to these interactions between both pools, it is possible, as Kappos et al. [26] and Quesnelle [45] show in their respective work, to devise linking heuristics for Zcash. To some extent and in a few settings only, it is thus possible to link addresses to the same entity. Consequently, an adversary may perform better than random guessing to jeopardize unlinkability in t-to-z and z-to-t transactions, i.e., $1 < \epsilon < \mathcal{O}\left(|\mathcal{S}|^2\right)$. Additionally, only a few of all transactions in Zcash are shielded. Thus, most transactions are transparent and do not provide the above privacy features.

Tornado Cash. Regarding Tornado Cash, it is part of blockchains that do not provide any privacy features by default but aims to provide unlinkability. Here, only the transactions involving Tornado Cash are considered for the experiment. We also assume that each user employs different addresses for deposits and withdrawals. Otherwise, the mixer would be subject to the trivial heuristic. However, as Wang et al. [56] shows, the mixer privacy still depends on multiple factors, such as the anonymity set of the mixer pool and, most importantly, the behavior of other users. They devise heuristics that, once combined, increase the probability of linking the withdrawer to the correct depositor on average by more than 50% [56]. The anonymity set of Tornado Cash is, therefore, smaller than advertised. An adversary may rely on a non-empty set of heuristics to link depositors and withdrawers, i.e., $1 < \epsilon < \mathcal{O}\left(|\mathcal{S}|^2\right)$. We also believe that the mixer's unlinkability promises are constrained due to the transparent blockchain it is part of. Compared to Zcash, which enhances privacy to a complete blockchain subset, Tornado Cash is only effective on single transactions.

Summary. We finally summarise our explanations in table 5.4 in which the symbols refer to each of the corresponding definitions on achieving notions, i.e., definition 4.5 for untraceability, 4.11 for unlinkability and 4.15 for confidentiality. We consider the four systems outlined above and their variations. That includes Bitcoin (standard) in which users do not necessarily perform address change, Bitcoin where users are hypothetically enforced to change their address at each transaction (address change) and use Coinjoin, Zcash with different transaction types, and Monero. As can be seen, Monero and Zcash are the sole systems to provide untraceability at least partially. The other systems have actual visible consumption and therefore offer no untraceability. Monero provides this property to some extent due to the limitations of ring signature, as discussed and referenced above. As for unlinkability, the standard Bitcoin does not provide unlinkability due to the reuse of addresses. An adversary playing the unlinkability game would immediately win using the trivial heuristic. With address change and other mechanisms such as Coinjoin, Bitcoin does provide unlinkability but only to a certain length due to linking heuristics. Other systems are subject to heuristics, such as Zcash (t-to-z and z-to-t) and Tornado Cash, and thus only partially provide unlinkability. Monero, conversely, offers strong unlinkability property. Finally, confidentiality is entirely achieved by both Monero, and Zcash shielded pool. We also consider that Zcash t-to-z and z-to-t does not provide confidentiality, as it is possible for an adversary to devise a function on the shielded amount by observing the plain ones (e.g., an upper bound). This comparison concludes our work on blockchain privacy notions. We here do not discuss fungibility but believe that untraceability and unlinkability may contribute to this property as mentioned by Kelen and Seres [28]. Also, as discussed, confidentiality is a strong pillar of those two properties and may, in fact, influence fungibility.

Finally, we did not evaluate the *scalability* of the different systems. Since both Monero and Zcash have a forever-growing UTXO set, their scalability is limited. As a result, users cannot store a compact blockchain representation (thin client). Two cryptocurrencies that address this are Quisquis [21] or Zether [13], which do not suffer these limitations by storing a relatively small amount of data and efficiency improvements on zero-knowledge proofs. Continuation of this work could therefore focus on modeling those two cryptocurrencies.

Scheme	**Untraceability**	**Unlinkability**	**Confidentiality**	**References**
Bitcoin (standard)	○	○	○	-
Bitcoin (address change and Coinjoin)	○	◐	○	[2, 36]
Monero	◐	●	●	[39, 60, 55, 19]
Zcash (t-to-t)	○	○	○	see Bitcoin (standard)
Zcash (t-to-z)	○	◐	○	[45, 26, 7]
Zcash (z-to-t)	●	◐	○	[45, 26, 7]
Zcash (z-to-z)	●	●	●	[11]
Tornado Cash	○	◐	○	[56]

● = provides properties, ◐ = partially provides property, ○ = does not provide property

Table 5.4: This is a summary table of the comparison of the system. Only a subgraph of the complete PDAG is passed to the experiment for the different systems.

Chapter 6

Conclusion

This work has tackled the definition of three blockchain privacy notions untraceability, unlinkability, and confidentiality. To do so, we first extended the TDAG [15] to capture privacy-preserving blockchains (PDAG) and, secondly, gave consistent definitions to these notions according to the PDAG notation. This allowed us to model and compare blockchain implementations and unify literature results. We thus answer each of our research questions. (1) First, we showed it was possible to extend the TDAG to privacy-preserving blockchains, regardless of the underlying technology. (2) Second, we defined privacy notions using the transaction graph model. (3) Last but not least, our model allowed the comparison of the different blockchain systems in terms of privacy guarantees.

During the course of this work, we also have discovered that the PDAG gives intuitive definitions to each notion and proposes a way to reason on the relation between them. It also provides enough formalism to prove and show complex properties in the blockchain (e.g., double-spending prevention). However, it is probably not a universal model since it only lies on the transactional level. For instance, it cannot capture network or off-chain privacy notions. Therefore, PDAG is not a one-size-fits-all solution to capture privacy notions. Yet, it is essential to build privacy solutions considering every layer of a blockchain system. One weak layer could indeed jeopardize the privacy of the ones above. Therefore, privacy in the public blockchain is a complicated, multifaceted concept.

Finally, the ongoing collaboration between research and cryptocurrency communities has proven to be fruitful for the privacy guarantees of blockchains. We thus believe that this cooperation is one essential pillar towards the improvement and widespread adoption of privacy-preserving cryptocurrencies. However, to which extent should we promote privacy is still an open question. It is well known that blockchains are also used to conduct illegal activities. Such practices are undoubtedly undesirable, but it is still unclear how to promote citizens' privacy while limiting criminal operations in a decentralized setting.

Bibliography

[1] Niluka Amarasinghe, Xavier Boyen, and Matthew McKague. “The Complex Shape of Anonymity in Cryptocurrencies: Case Studies from a Systematic Approach”. In: *Financial Cryptography and Data Security*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2021, pp. 205–225. DOI: `10.1007/978-3-662-64322-8_10`.

[2] Elli Androulaki, Ghassan O. Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. “Evaluating User Privacy in Bitcoin”. In: *Financial Cryptography and Data Security*. Ed. by Ahmad-Reza Sadeghi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 34–51. ISBN: 978-3-642-39884-1. DOI: `10.1007/978-3-642-39884-1_4`.

[3] Andreas Antonopoulos and Gavin Wood. *Mastering Ethereum: Building Smart Contracts and DApps*. First. O’Reilly Media, 2018.

[4] Nicola Atzei, Massimo Bartoletti, Stefano Lande, and Roberto Zunino. “A Formal Model of Bitcoin Transactions.” In: *Financial Cryptography and Data Security - 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26 - March 2, 2018, Revised Selected Papers*. 2018, pp. 541–560. DOI: `10.1007/978-3-662-58387-6_29`.

[5] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. *Zerocash: Decentralized Anonymous Payments from Bitcoin*. Cryptology ePrint Archive, Paper 2014/349. 2014.

[6] Alastair Berg. “The Identity, Fungibility, and Anonymity of Money”. In: *SSRN Electronic Journal* (2018). DOI: `10.2139/ssrn.3211011`.

[7] Alex Biryukov and Daniel Feher. “Privacy and Linkability of Mining in Zcash”. In: *2019 IEEE Conference on Communications and Network Security (CNS)*. 2019, pp. 118–123. DOI: `10.1109/CNS.2019.8802711`.

[8] Alex Biryukov, Daniel Feher, and Giuseppe Vitto. “Privacy Aspects and Subliminal Channels in Zcash”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. London United Kingdom: ACM, 2019, pp. 1813–1830. ISBN: 978-1-4503-6747-9. DOI: `10.1145/3319535.3345663`.

[9] Alex Biryukov and Dmitry Khovratovich. “Equihash: Asymmetric Proof-of-Work Based on the Generalized Birthday Problem”. In: *Proceedings 2016 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2016. ISBN: 978-1-891562-41-9. DOI: `10.14722/ndss.2016.23108`.

[10] Bitcoin Wiki. *CoinJoin*. `https://en.bitcoin.it/wiki/CoinJoin`. 2023.

[11] Sean Bow, Taylor Hornby, and Nathan Wilcox. *Zcash Protocol Specification*. `https://zips.z.cash/protocol/protocol.pdf`. 2021.

[12] Benedikt Bunz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA: IEEE, 2018, pp. 315–334. ISBN: 978-1-5386-4353-2. DOI: `10.1109/SP.2018.00020`.

[13] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. “Zether: Towards Privacy in a Smart Contract World”. In: *Financial Cryptography and Data Security*. Vol. 12059. Cham: Springer International Publishing, 2020, pp. 423–443. DOI: `10.1007/978-3-030-51280-4_23`.

[14] Vitalik Buterin. *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. `https://ethereum.github.io/yellowpaper/paper.pdf`. 2014.

[15] Christian Cachin, Angelo De Caro, Pedro Moreno-Sanchez, Björn Tackmann, and Marko Vukolic. “The Transaction Graph for Modeling Blockchain Semantics”. In: *Cryptoeconomic Systems* (2020). DOI: `10.21428/58320208.a12c57e6`.

[16] Mauro Conti, E. Sandeep Kumar, Chhagan Lal, and Sushmita Ruj. “A Survey on Security and Privacy Issues of Bitcoin”. In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 3416–3452. ISSN: 1553-877X. DOI: `10.1109/COMST.2018.2842460`.

[17] Wei Dai. *Navigating Privacy on Public Blockchains*. `https://wdai.us/posts/navigating-privacy/`. 2022.

[18] A. L. Dulmage and N. S. Mendelsohn. “Coverings of Bipartite Graphs”. In: *Canadian Journal of Mathematics* 10 (1958), pp. 517–534. ISSN: 0008-414X, 1496-4279. DOI: `10.4153/CJM-1958-052-0`.

[19] Christoph Egger, Russell W. F. Lai, Viktoria Ronge, Ivy K. Y. Woo, and Hoover H. F. Yin. *On Defeating Graph Analysis of Anonymous Transactions*. Cryptology ePrint Archive, Paper 2022/132. 2022. DOI: `10.2478/popets-2022-0059`.

[20] Justin Ehrenhofer. *Response to ”An Empirical Analysis of Traceability in the Monero Blockchain”, Version 2*. `https://www.getmonero.org/2018/03/29/response-to-an-empirical-analysis-of-traceability.html`. 2018.

[21] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. “Quisquis: A New Design for Anonymous Cryptocurrencies”. In: *Advances in Cryptology – ASIACRYPT 2019*. Vol. 11921. Cham: Springer International Publishing, 2019, pp. 649–678. DOI: `10.1007/978-3-030-34578-5_23`.

[22] Steven Goldfeder, Harry Kalodner, Dillon Reisman, and Arvind Narayanan. “When the Cookie Meets the Blockchain: Privacy Risks of Web Payments via Cryptocurrencies”. In: *Proceedings on Privacy Enhancing Technologies* (2018). ISSN: 2299-0984.

[23] Bernhard Haslhofer, Roman Karl, and Erwin Filtz. “O Bitcoin Where Art Thou? Insight into Large-Scale Transaction Graphs”. In: CEUR Workshop Proceedings. 2016.

[24] Ryan Henry and Ian Goldberg. “Formalizing Anonymous Blacklisting Systems”. In: *2011 IEEE Symposium on Security and Privacy*. Oakland, CA, USA: IEEE, 2011, pp. 81–95. ISBN: 978-1-4577-0147-4. DOI: `10.1109/SP.2011.13`.

[25] Harry Kalodner, Malte Möser, Kevin Lee, Steven Goldfeder, Martin Plattner, Alishah Chator, and Arvind Narayanan. “{BlockSci}: Design and Applications of a Blockchain Analysis Platform”. In: *29th USENIX Security Symposium (USENIX Security 20)*. 2020, pp. 2721–2738. ISBN: 978-1-939133-17-5.

[26] George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. “An Empirical Analysis of Anonymity in Zcash”. In: *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. Ed. by William Enck and Adrienne Porter Felt. USENIX Association, 2018, pp. 463–477.

[27] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Third edition. Chapman & Hall/CRC Cryptography and Network Security Series. Boca Raton London New York: CRC Press Taylor & Francis Group, 2021.

[28] Domokos Miklós Kelen and István András Seres. *Towards Measuring The Fungibility and Anonymity of Cryptocurrencies*. 2022. DOI: `10.48550/arXiv.2211.04259`. arXiv: `2211.04259 [cs]`.

[29] Koe, Kurt M. Alonso, and Sarang Noether. *Zero to Monero*. Second. 2020.

[30] Christiane Kuhn, Martin Beck, Stefan Schiffner, Eduard Jorswieck, and Thorsten Strufe. *On Privacy Notions in Anonymous Communication*. 2019. arXiv: `1812.05638 [cs]`.

[31] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. “A Traceability Analysis of Monero’s Blockchain”. In: *Computer Security – ESORICS 2017*. Ed. by Simon N. Foley, Dieter Gollmann, and Einar Snekkenes. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 153–173. ISBN: 978-3-319-66399-9. DOI: `10.1007/978-3-319-66399-9_9`.

[32] Duc V. Le and Arthur Gervais. *AMR:Autonomous Coin Mixer with Privacy Preserving Reward Distribution*. 2021. arXiv: `2010.01056 [cs]`.

[33] Adam Mackenzie, Surae Noether, and Monero Core Team. *Improving Obfuscation in the CryptoNote Protocol*. 2015.

[34] Gregory Maxwell. *CoinJoin: Bitcoin Privacy for the Real World*. `https://bitcointalk.org/index.php?topic=279249.0`. Forum. 2013.

[35] Sarah Meiklejohn and Claudio Orlandi. “Privacy-Enhancing Overlays in Bitcoin”. In: *Financial Cryptography and Data Security*. Ed. by Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2015, pp. 127–141. ISBN: 978-3-662-48051-9. DOI: `10.1007/978-3-662-48051-9_10`.

[36] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. “A Fistful of Bitcoins: Characterizing Payments among Men with No Names”. In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. Barcelona Spain: ACM, 2013, pp. 127–140. ISBN: 978-1-4503-1953-9. DOI: 10.1145/2504730.2504747.

[37] Malte Möser, Rainer Böhme, and Dominic Breuker. “An Inquiry into Money Laundering Tools in the Bitcoin Ecosystem”. In: *2013 APWG eCrime Researchers Summit*. 2013, pp. 1–14. DOI: 10.1109/eCRS.2013.6805780.

[38] Malte Möser and Arvind Narayanan. “Resurrecting Address Clustering in Bitcoin”. In: *Financial Cryptography and Data Security*. Vol. 13411. Cham: Springer International Publishing, 2022, pp. 386–403. DOI: 10.1007/978-3-031-18283-9_19.

[39] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. “An Empirical Analysis of Traceability in the Monero Blockchain”. In: *Proceedings on Privacy Enhancing Technologies* (2018).

[40] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf. 2008.

[41] Shen Noether, Adam Mackenzie, and The Monero Research Lab. “Ring Confidential Transactions”. In: *ledger* 1 (2016), pp. 1–18. ISSN: 2379-5980. DOI: 10.5195/ledger.2016.34.

[42] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. “Structure and Anonymity of the Bitcoin Transaction Graph”. In: *Future Internet* 5.2 (2013), pp. 237–250. ISSN: 1999-5903. DOI: 10.3390/fi5020237.

[43] Rafael Pass and Elaine Shi. “Rethinking Large-Scale Consensus”. In: *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. Santa Barbara, CA: IEEE, 2017, pp. 115–129. ISBN: 978-1-5386-3217-8. DOI: 10.1109/CSF.2017.37.

[44] Andreas Pfitzmann and Marit Hansen. *A Terminology for Talking about Privacy by Data Minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf. 2010.

[45] Jeffrey Quesnelle. *On the Linkability of Zcash Transactions*. 2017. DOI: 10.48550/arXiv.1712.01210. arXiv: 1712.01210 [cs].

[46] Fergal Reid and Martin Harrigan. “An Analysis of Anonymity in the Bitcoin System”. In: *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*. 2011, pp. 1318–1326. DOI: 10.1109/PASSAT/SocialCom.2011.79.

[47] Ronald L. Rivest, Adi Shamir, and Yael Tauman. “How to Leak a Secret”. In: *Advances in Cryptology — ASIACRYPT 2001*. Vol. 2248. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 552–565. DOI: 10.1007/3-540-45682-1_32.

[48] Dorit Ron and Adi Shamir. “Quantitative Analysis of the Full Bitcoin Transaction Graph”. In: *Financial Cryptography and Data Security*. Ed. by Ahmad-Reza Sadeghi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 6–24. ISBN: 978-3-642-39884-1. DOI: 10.1007/978-3-642-39884-1_2.

[49] Tim Ruffing and Pedro Moreno-Sanchez. “ValueShuffle: Mixing Confidential Transactions for Comprehensive Transaction Privacy in Bitcoin”. In: *Financial Cryptography and Data Security*. Vol. 10323. Cham: Springer International Publishing, 2017, pp. 133–154. DOI: 10.1007/978-3-319-70278-0_8.

[50] Riccardo Spagni. *Monero 0.13.0 ”Beryllium Bullet” Release*. https://www.getmonero.org//2018/10/11/monero-0.13.0-released.html. 2018.

[51] U.S. Department of the Treasury. *Cyber-Related Designation*. https://home.treasury.gov/policy-issues/financial-sanctions/recent-actions/20220808. 2022.

[52] U.S. Department of the Treasury. *U.S. Treasury Sanctions Notorious Virtual Currency Mixer Tornado Cash*. https://home.treasury.gov/news/press-releases/jy0916. 2022.

[53] Nicolas van Saberhagen. *CryptoNote*. https://bytecoin.org/old/whitepaper.pdf. 2013.

[54] Friedhelm Victor. “Address Clustering Heuristics for Ethereum”. In: *Financial Cryptography and Data Security*. Vol. 12059. Cham: Springer International Publishing, 2020, pp. 617–633. DOI: 10.1007/978-3-030-51280-4_33.

[55] Saravanan Vijayakumaran. *Analysis of CryptoNote Transaction Graphs Using the Dulmage-Mendelsohn Decomposition*. Cryptology ePrint Archive, Paper 2021/760. 2021.

[56] Zhipeng Wang, Stefanos Chaliasos, Kaihua Qin, Liyi Zhou, Lifeng Gao, Pascal Berrang, Ben Livshits, and Arthur Gervais. *On How Zero-Knowledge Proof Blockchain Mixers Improve, and Worsen User Privacy*. 2022. DOI: 10.48550/arXiv.2201.09035. arXiv: 2201.09035 [cs].

[57] Dimaz Ankaa Wijaya, Joseph Liu, Ron Steinfeld, and Dongxi Liu. "Monero Ring Attack: Recreating Zero Mixin Transaction Effect". In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. New York, NY, USA: IEEE, 2018, pp. 1196–1201. ISBN: 978-1-5386-4388-4. DOI: 10.1109/TrustCom/BigDataSE.2018.00165.

[58] He Xi, He Ketai, Lin Shenwen, Yang Jinglin, and Mao Hongliang. *Bitcoin Address Clustering Method Based on Multiple Heuristic Conditions*. 2022. DOI: 10.48550/arXiv.2104.09979. arXiv: 2104.09979 [cs].

[59] Jiangshan Yu, Man Ho Allen Au, and Paulo Esteves-Verissimo. "Re-Thinking Untraceability in the CryptoNote-Style Blockchain". In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. Hoboken, NJ, USA: IEEE, 2019, pp. 94–9413. ISBN: 978-1-72811-407-1. DOI: 10.1109/CSF.2019.00014.

[60] Zuoxia Yu, Man Ho Au, Jiangshan Yu, Rupeng Yang, Qiuliang Xu, and Wang Fat Lau. "New Empirical Traceability Analysis of CryptoNote-Style Blockchains". In: *Financial Cryptography and Data Security*. Ed. by Ian Goldberg and Tyler Moore. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 133–149. ISBN: 978-3-030-32101-7. DOI: 10.1007/978-3-030-32101-7_9.