Understanding Penumbra: Privacy-Preserving Blockchain Architecture and Design

Master Thesis

Noah Maggio

Faculty of Science, University of Bern

January 2026

Prof. Dr. Christian Cachin
François-Xavier Wicht

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

Abstract

There exist many blockchain protocols, which are inherently transparent. Transparency provides a beneficial side effect of being traceable and auditable but poses a challenge for privacy. Penumbra follows a privacy-first design, making privacy the default. It solves the tension between privacy and auditability through various mechanisms. Penumbra is one of the first blockchain protocols in the Cosmos ecosystem that provides privacy and allows for selective disclosure in order to comply with regulatory requirements and auditability. This thesis analyzes the architecture, mechanisms, and cryptographic primitives of the Penumbra protocol, reviewing how it works and how it delivers privacy. Furthermore, a privacy and design analysis regarding confidentiality, integrity, accountability, and selective disclosure is performed, inspired by the CIA triad. The overall analysis is based on various sources, such as the official Penumbra documentation as well as some external audits. This thesis shows that Penumbra achieves a high level of privacy by combining an encrypted, multi-asset-capable shielded pool with zero-knowledge proofs, providing transaction validity. Penumbra combines and balances privacy, functionality, and auditability, although at the cost of protocol and implementation complexity.

Contents

1 Introduction

Cryptocurrencies gained increased popularity in the past decade. More and more blockchain-based financial systems are adopted by the general population. But because of the sheer amount of users, privacy has become a critical property for protecting both individuals and institutions, hence, privacy is no longer just a niche requirement. Nowadays, legal regulations and competitive business environments require, or rather demand, systems to be private and confidential while remaining verifiable and accountable. The two most famous blockchains are arguably Bitcoin and Ethereum. Both of them are public blockchains (so-called transparent blockchains) and decentralized, decoupling them from a single authority with control over everything. A single authority brings many benefits, especially in the range of controllability, verifiability, and auditability, but it demands full trust of its users and brings a very high degree of dependency (full power to the one in control). While decentralization frees users from this dependency and the transparency allows for verifiability and auditability in such an environment, the traceability aspect impacts the privacy negatively, since every transaction is permanently recorded on a public ledger, giving away the contained data. Many blockchains rely on pseudonymization in order to mitigate identity disclosure. This means users are represented by cryptographic addresses instead of their real-world identities. However, this only provides limited privacy, as transaction history is still publicly available and linkable. Thus, in this environment, privacy remains limited and susceptible to deanonymization processes. Additionally, many blockchains work in isolation, meaning they can only handle transactions using their own currency, guaranteeing their functionality and security but making them not versatile.

The two properties, privacy and transparency, represent contradictory goals, creating a tension that is difficult for both sides to satisfy. Furthermore, the interoperability of a protocol working with different currencies represents a challenge that not many blockchains have addressed or adopted. The tension between privacy and transparency has led to the development of privacy-preserving blockchain protocols, such as Zcash or Monero. Many of those protocols have some restrictions and limitations, such as providing only an opt-in/opt-out privacy (not default), restricted disclosure mechanisms (for regulatory compliance), or single-asset designs making the currency not interoperable across different currencies. Penumbra aims to solve all those mentioned challenges/restrictions.

Penumbra provides a privacy-first blockchain that acts in the Cosmos ecosystem, making it interoperable within. Privacy is enforced by default, and it uses a multi-asset-capable shielded pool, where every transaction occurs protected and privately across different assets. The main driving factor for privacy, as well as verifiability, is the usage of zero-knowledge proofs that allow private verification of transactions. The selective disclosure mechanism that Penumbra also provides makes it possible to reveal, in a controlled way, information for auditability and regulatory compliance. Additionally, Penumbra uses its own private decentralized exchange mechanism to prevent any form of maximum extractable value (MEV) attacks, such as front-running.

Despite its design and functionality, Penumbra has not received much academic attention. Information is spread across protocol specifications, external audit reports, developer documentation, and blogs. It lacks a consolidated and structured analysis of its architecture. This thesis will analyze the Penumbra protocol regarding its architecture and privacy-oriented design, guided by the following four research questions:

- *How does the Penumbra protocol, a privacy-preserving blockchain, achieve privacy at the architectural and cryptographic level?*
- *How does Penumbra balance privacy with verifiability and selective disclosure?*
- *How does the privacy design of Penumbra address confidentiality, integrity, and accountability requirements?*
- *What are Penumbra's limitations and trade-offs for a privacy-first design?*

The focus of this thesis lies on describing and analyzing the protocol-level design, i.e., the architecture, mechanisms, and cryptographic primitives that the Penumbra protocol uses and provides. Furthermore, privacy evaluation, comparison to other privacy-preserving blockchains, and the limitations and trade-offs will be discussed. The structure is as follows: In Chapter 2, some background knowledge and definitions will be introduced, which are used in the Penumbra protocol, in order for better understanding of upcoming primitives and their mechanisms. Chapter 3 will go over the whole protocol, analyzing its functionalities, mechanisms, and cryptographic primitives. Chapter 4 analyzes the privacy of the Penumbra protocol, a comparison between two other privacy-preserving protocols, and an evaluation of Penumbra's limitations and trade-offs. Chapter 5 concludes the thesis.

2 Background

2.1 Blockchain and distributed ledger fundamentals

Blockchains belong to the distributed-ledger technology (DLT), which are decentralized databases maintained by a network of users [2] [43]. In the classical view of decentralized databases, there is no single instance or actor that is responsible for everything (like validating and storing transactions), compared to a centralized database. It usually relies on a consistency mechanism called consensus, where multiple actors/users in the network agree on a single, verified version of data, even under the possibility of failures or unreliable network conditions [40] [43].

Many different consensus mechanisms exist, all with their own techniques to provide overall agreement on some data values. Popular examples of actively used permissionless consensus mechanisms are the Proof-of-X, like Proof-of-Work (PoW), Proof-of-Stake (PoS), or Delegated Proof-of-Stake (DPoS), where there is always some sort of risk/reward tied to it, making it unappealing to misbehave. Although different consensus mechanisms use different techniques, their main goal stays the same: reaching agreement in a network under the possibility of failures, unreliability, or misbehaving actors.

To consider a consensus protocol a correct and functioning protocol, it is required to fulfill four properties [9]: *agreement*, *validity*, *integrity*, and *termination*. Agreement ensures that all correct participants agree on a single state, validity ensures that any decided value was proposed by some process, integrity ensures that no processes decide twice, and termination ensures that the protocol eventually reaches a decision, even in the presence of failures.

Blockchains are a specific type of distributed ledger. The data gets structured into blocks, which are linked together in a sequential and immutable chain using cryptographic hashes [2] [43]. Since every block contains a hash of the previous one, it makes it nearly impossible to change the past without recomputing the whole chain (provided the majority of the network acts correct/valid). Actors in the network can play two kinds of roles: validator (also called miner), who essentially validates an existing chain and proposes a new block to be added at the end of the chain, and then there are the regular users, who perform transactions but do not participate in the consensus process [2].

Earlier developed blockchains, like Bitcoin and Ethereum, are examples of transparent blockchains [22][43]. Transparent means all transactions and account balances are publicly visible, as they are stored/recorded on a publicly accessible ledger. This has the side effect of traceability. This traceability can bring more trust and accountability due to every action being transparent, yet at the same time it can hurt the privacy of users, since the addresses and information of actions are publicly visible. Although the addresses of the users are pseudonyms (cryptographic addresses), meaning they do not directly reveal any information about the user (unlike a real username), it is technically difficult to trace an address back to a specific user. However, because something is difficult does not mean it is impossible. Since the addresses are only pseudonyms, not fully anonymous, it means with enough engagement and additional external information (for example, through network analysis or analyzing the behavioral pattern), it is possible to link a pseudonym address to a real-world identity [43]. Some heuristics and analysis techniques that can undermine pseudonymization are common-input-ownership heuristics, address change detection, amount correlation, and timing correlation, which all provide the possibility to deanonymize pseudonyms and link them back to real-world identities [4] [30] [44]. There also exist blockchains that do preserve privacy, for example, Zcash [18] or Monero [16]. Both use cryptographic mechanisms, like zero-knowledge proofs or ring-signatures. With those mechanisms, it is possible to hide information (like the total amount of a transaction, sender, and/or receiver) even when the actions (such as a transaction) are recorded and available publicly, making it fully anonymous [31] [46] [55].

Another aspect where blockchains can differentiate themselves from some others is by using different data structures. There are two common models (data structures), the UTXO-model (Unspent Transaction Output) and the account-model [2] [57]. In the UTXO-model (for example, used in Bitcoin), each transaction consumes an existing output and creates a new one. The blockchain tracks the ownership and status of those outputs. The benefits of this model are that it is modular and ideal for parallel processing [2]. In the account-model (for example, used in Ethereum), as the name implies, accounts are used, i.e., the blockchain tracks account balances, which simply get updated through transactions. This model is more intuitive but can be hard to parallelize [57].

The choice between these two models affects not only efficiency and parallelization but also the implementation of privacy. Depending on the used models, different information will become public (account balances in the account-model or transaction information in the UTXO-model).

Byzantine fault tolerance

In blockchains, or more generally in distributed systems, the so-called Byzantine fault tolerance (BFT) is often taken as the failure assumption model. BFT originated from the Byzantine generals problem [40], a scenario in historical days when multiple generals (participants), stationed away from each other, want to agree with each other (through messages, i.e., letters) if they will send their army into battle. Some generals might be traitors or unreliable and will either not answer or not act according to what they send, which can cause losing a battle due to uncoordinated attacks. The BFT essentially captures this problem in modern days, where faulty participants (Byzantine ones) may arbitrarily deviate from a protocol by either sending conflicting messages or purposely being absent, resulting in unpredictable behavior. The Byzantine fault tolerance consensus protocol was originally intended for permissioned systems that have a fixed set of known participants. Modern systems and networks, such as Tendermint-based networks, adapted these principles to a permissionless or semi-permissioned setting using stake-weighted validator sets.

One of the main results from the BFT theory is that at most $f < \frac{n}{3}$ Byzantine actors can be tolerated in consensus protocols operating in asynchronous or partially synchronous systems, where n represents the total number of participants and f represents the number of faulty ones. However, in stake-weighted systems, this classical bound does not apply to the number of validators but to the distribution of stake. In this setting, n represents the total voting power, and f represents the voting power that is controlled by Byzantine validators. Hence, as long as the adversarial stake stays below one-third of the total stake, the consensus safety and liveness are guaranteed. Rewriting $f < \frac{n}{3}$ gives us the consensus requirement of $n \geq 3f + 1$, a bound that resulted from the need to guarantee safety (agreement) and liveness (termination) in an adversarial environment [10].

Consensus mechanisms often rely on *quorum* intersections, where a quorum is the minimal number of participants required to agree on a state in order to be considered as valid and fault tolerant without needing all nodes to agree with each other. A quorum in classical BFT is defined as $q > \frac{n+f}{2}$, where n stands for the total number of participants and f represents the maximal number of Byzantine participants. For the minimal system size $n = 3f + 1$, this reduces to $q = 2f + 1$. Now for any two quorums Q_1 and Q_2, their intersection satisfies $|\mathsf{Q}_1 \cap \mathsf{Q}_2| \geq 2q - n$, which ensures that the intersection contains at least $f + 1$ participants when $n = 3f + 1$. This guarantees at least one honest participant, therefore safety. In case of $n > 3f + 1$, then $q > \frac{n+f}{2}$, and the intersection is respectively larger. This means if an adversary controls at most f Byzantine nodes, i.e., less than one-third of the total participants ($f < \frac{n}{3}$), then the consensus remains safe and live. If an adversary controls $f \geq \frac{n}{3}$ or more participants, then it is possible to break the overlap of quorums, violating safety or liveness [10].

2.2 Privacy in cryptocurrencies

Privacy is a fundamental part of cryptography and an important aspect for financial systems. In transparent blockchains, transaction data can reveal sensitive information about individuals and organizations [17] [41]. Information or data is of immense value today, which makes privacy crucial. One of the main reasons to introduce cryptocurrencies was to remove the centralization part (the dependency on one single entity that is in control of everything), like a bank [41]. Transparent blockchains provide decentralization and autonomy, but they also introduce transparency, which can be seen as a “blessing” and a “curse”.

As every transaction that is performed on and over the blockchain is permanently recorded on a public ledger, accessible to everyone, it provides verifiability. This brings clarity but also heavily impacts privacy, as everyone who has a way to access the blockchain can take a look at it (trace back the origin of the coin/currency, find out where it goes, and even how much was/is in a particular wallet) [18] [35] [41]. This infringement on privacy may be of big concern for users who do not want to disclose their financial activities. In general, users do not disclose their information to everyone, like how much money they earn, how much is in their bank account, and for what they spend their money. In addition, if such information were publicly available, it would make users that have a large sum in their accounts/wallets a popular target for cyberattacks or even susceptible to real-world threats. This is why there exists in traditional banking the concept of “banking secrecy”. Hence, privacy is a necessary condition in the world of cryptocurrencies [17] [18].

Enhanced privacy mechanisms such as zero-knowledge proofs can provide the needed degree of privacy while maintaining verifiability [25]. Although the data obtained from those blocks does not disclose a user’s identity directly (pseudonyms), as discussed in the previous section, deanonymization is possible, meaning it still can pose a privacy risk [35] [41]. Therefore, stronger cryptographic techniques are required to provide meaningful transaction privacy.

However, increasing privacy comes along with certain downsides that bring (unwanted) negative effects. For example, zero-knowledge proofs, simply speaking, bring the ability to prove something without affecting privacy (giving away information), but that comes at the cost of extra computations and storage compared to just using transparency. Depending on the scale of the whole environment, this can impact transaction speed and increase the maintenance cost [25]. Another downside of just increasing privacy to a point of (nearly) total anonymity is the possible increase of illegal activities. This creates a large challenge for law enforcement, due to tracing transactions becoming more difficult as well as making it very hard to confiscate illegal assets. Investigations into money laundering, tax evasion, or detecting the financing of terrorist organizations are becoming nearly impossible. This is why it is often necessary to disclose certain details in order to prove something in a legal context.

This greatly shows that total privacy is not something that can coexist with full traceability. There will always be a tension between those two properties. Thus it requires taking certain trade-offs in order to balance privacy, verifiability, and legal compliance [18].

2.3 Cryptographic primitives

For cryptocurrencies to stay private, secure, and also verifiable, they rely on many different cryptographic primitives. The following section will go over some of the most important primitives, which are commitments, zero-knowledge proofs, and pseudorandom functions [8] [26] [47].

First, we introduce a concept that will be used multiple times in this section: negligibility, or, in short, *negl*.

Definition 2.3.1. A function f is *negligible* if for every polynomial p there is an N such that for all $n > N$ it holds $f(n) < 1/p(n)$. In other words, as this is usually used to show a probability, the result is so small that it is irrelevant, i.e., *negligible* (negl) [34].

With this definition of negligibility, one can now properly define cryptographic primitives, beginning with commitments, which rely on negligible probabilities to ensure their security.

Definition 2.3.2. *Commitments* bind (cryptographically) a value and keep it hidden, with the ability to reveal a bounded value at a later point. This guarantees that a user cannot change the value afterwards when it was committed. This binding provides a guarantee that a value in fact exists and is valid without disclosing it. More formally, the commitment scheme is a pair of two algorithms called *commit* and *reveal*: $(\mathsf{Commit}, \mathsf{Reveal})$ operating over some message space $\mathcal{M}$ and a randomness space $\mathcal{R}$.

- **Commit**: When the sender commits a value m, he computes $C = \mathsf{Commit}(m, r)$, where $m \in \mathcal{M}$ and $r \in \mathcal{R}$ which is a uniformly at random chosen value. C is the commitment that will be sent to the receiver.
- **Reveal**: The sender opens C by revealing m and r to the receiver. The receiver computes $C' = \mathsf{Commit}(m, r)$ and checks if $C' = C$.

The following properties are required to satisfy a secure commitment scheme:

- **Hiding**: The commitment C must be hiding, meaning it reveals nothing about m, i.e., $\mathsf{Commit}(m, r)$ and $\mathsf{Commit}(m', r)$ are indistinguishable from each other (one cannot say to which commitment m belongs to, except with negligible probability).
- **Binding**: Once the commitment C is created, it should be infeasible for the sender to output a commitment that can be later on revealed to two different messages, i.e., the probability of finding arbitrary $\mathsf{Commit}(m, r) = \mathsf{Commit}(m', r')$ for two random r's, is negligible.

This way, a sender cannot change m after committing it, and it is hidden, as the commitment does not reveal information about m before revealing. One of the better-known commitment schemes is the Pedersen commitment. It relies on the discrete logarithm problem and provide properties of being perfectly hiding and computationally binding [8] [47].

Zero-knowledge proofs are a powerful cryptographic primitive, as they allow you to prove the validity of a claim without disclosing any crucial/private information about it [26]. A zero-knowledge proof is an interactive protocol, or rather an interactive proof system (IP) between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, where $\mathcal{P}$ proves that a statement is true without providing information about an underlying secret w. Formally expressed:

Definition 2.3.3. An interactive proof system $(\mathcal{P}, \mathcal{V})$ for a language $\mathcal{L}$ is zero-knowledge if, for every probabilistic polynomial-time (PPT) verifier $\mathcal{V}*$ there exists a PPT algorithm $\mathcal{S}$ called the simulator, such that $\forall x \in \mathcal{L}$, the distribution of the output $\mathcal{S}(x)$ of the simulator is indistinguishable from $\mathsf{View}_{\mathcal{V}*}(\mathcal{P}(x), \mathcal{V}*(x))$. Hence the verifier is not getting any knowledge beyond the validity of the statement. $\mathsf{View}_{\mathcal{V}*}(\mathcal{P}(x), \mathcal{V}*(x))$ denotes the distribution over transcripts generated by the interaction of prover strategy $\mathcal{P}$ and verifier strategy $\mathcal{V}*$ within the interactive proof system [26] [54].

The core properties of a zero-knowledge proof are:

- **Completeness**: If the statement is true ($x \in \mathcal{L}$), then an honest verifier (i.e., a correct one) will accept the proof (is convinced by the prover) with an overwhelming probability. $\Pr[\mathcal{V}\mathsf{accepts}(\mathcal{P} \leftrightarrow \mathcal{V})|x \in \mathcal{L}] \geq 1 - |x|^{-k}$, for any $k > 0$ and sufficiently large inputs x.
- **Soundness**: If the statement is false ($x \notin \mathcal{L}$), then no arbitrary (malicious) prover $\mathcal{P}*$ can trick the verifier into believing the statement is true, except with a negligible probability $\forall \mathcal{P}*$: $\Pr[\mathcal{V}\mathsf{accepts}(\mathcal{P}* \leftrightarrow \mathcal{V})|x \notin \mathcal{L}] \leq |x|^{-k}$.
- **Zero-Knowledge**: Already stated above in the definition. For completeness: if for every efficient verifier $\mathcal{V}*$ exists an efficient simulator $\mathcal{S}$, that can reproduce the view of the verifier, then the output of $\mathcal{S}$ is computationally indistinguishable from the real view; $\mathsf{View}_{\mathcal{V}*}(\mathcal{P}(x), \mathcal{V}*(x)) \approx \mathcal{S}(x)$, meaning the verifier does not learn any additional information about the secret w beyond the true statement $x \in \mathcal{L}$.

The first two properties belong to interactive proof systems, the last one makes it a zero-knowledge proof.

There exist many kinds of zero-knowledge proofs, however, this thesis will only focus on zk-SNARKs, as they are used in the Penumbra protocol. Zk-SNARK (Succinct Non-Interactive Arguments of Knowledge) is one of the first efficient zero-knowledge proof systems, it allows short proofs and fast verification, but in many constructs it presumes a trusted set-up in advance [28]. But more about zero-knowledge proofs in relation to Penumbra will be discussed later on in its dedicated section.

Besides commitments and zero-knowledge proofs, *encryption* also plays a central role in privacy systems. Symmetric encryption protects transaction content, such as amount or receiver information, in a way that it is only visible/readable for authorized users/parties. Hierarchical deterministic key derivation allows the creation of a multitude of keys from a master seed. This simplifies the process for backups, recovery, and portability of wallets. A *wallet* in this context is a software or hardware tool that enables users to store and manage their cryptographic keys. This gives them access and control over their funds or digital assets. As mentioned previously, privacy is often in high tension with legal/regulatory requirements, hence the introduction of viewing keys that allow the possibility of selective disclosure. There also exist full viewing keys that allow insight into all transactions of a user. Such mechanisms allow the compromise between privacy and traceability.

Another important cryptographic primitive is a so-called pseudorandom function (PRF). A common problem in cryptocurrencies is the possibility to double-spend due to the nature of how digital data/currencies are handled/transferred. A PRF is not the main factor in preventing the double-spend problem (this credit goes to the consensus protocol), but it plays an important role in it. PRFs are used to create commitment values, generate keys, or create unique identifiers, for example, so-called nullifiers, which play a significant role in preventing double spending in the upcoming Penumbra protocol. PRFs allow one to deterministically generate an output, given some secret information, which appears to be truly random for any outside observer (i.e., it is indistinguishable from random).

Definition 2.3.4. A *pseudorandom function* (PRF) $F : \{0,1\}^\lambda \times \{0,1\}^{in} \rightarrow \{0,1\}^{out}$ is a deterministic function that takes a key $k \in \{0,1\}^\lambda$ and an input $x \in \{0,1\}^{in}$ and outputs a value $y \in \{0,1\}^{out}$. $F(k,x)$ is computationally indistinguishable from a function chosen uniformly at random having the same domain and range [34]. (*Remark*: $F(k,x)$ is the same as $F_k(x)$ for fixed key k)

A PRF has two properties [34]:

- **Efficiency**: For every input $x \in \{0,1\}^{in}$, $F_k(x)$ can be calculated efficiently, meaning there is a polynomial-time algorithm that computes $F(k,x)$ given k and x.
- **Pseudorandomness**: The function $F_k(x)$ is a pseudorandom function if for all probabilistic polynomial-time distinguishers D, there is a negligible function *negl* such that:

  $$|Pr[D^{F_k} = 1] - Pr[D^f = 1]| \leq \mathsf{negl}$$

  where f denotes a truly random function.

2.4 Existing protocols

Since the introduction of Bitcoin, privacy-preserving cryptographic currencies have developed significantly. Two of the more known approaches are Zcash and Monero, both of them use different cryptographic mechanisms to provide different levels of anonymity, untraceability, and privacy.

Zcash is an implementation of the *ZeroCash protocol* [3] that is based on the Bitcoin transparency model (UTXO-model) and its goal is to provide confidential transactions where sender, receiver, and amount are hidden. Zcash supports transparent as well as shielded UTXOs. The transparent UTXOs behave like Bitcoin, while the shielded UTXOs are cryptographically protected. In this system, each transaction consumes an existing, previously generated output and creates a new one. For shielded transactions, the process of consuming an output and creating a new one is accompanied by identifiers, essentially tracking nullifiers to prevent double spending. In ZeroCash, these nullifiers are derived using a PRF $F_k(x)$ that generates deterministic values for the spender, but for any outside observer, those values look completely indistinguishable from random and thus are unlinkable to any existing transaction. Besides PRFs and nullifiers, Zcash also uses commitments $C = \mathsf{Commit}(m,r)$ in order to hide values (m) but also bind them, providing consistency. This all happens in a shielded pool, a secure environment where actions, such as transactions, appear and are handled/stored, providing users with the ability to choose if they want to use transparent (like Bitcoin) or protected transactions. Transactions that are shielded (protected) hide various pieces of information such as sender, receiver, and transaction amount using commitments and zero-knowledge proofs. This provides strong privacy and unlinkability. Other transaction metadata, such as block inclusion and UTXO commitments, remain visible on-chain, while the content of the shielded transaction is cryptographically protected. With the help of zero-knowledge proofs, specifically zk-SNARK, it allows one to prove the correctness of a transaction without revealing any information about the transaction content itself (such as sender, receiver, or amount). Due to the properties of zk-proofs, i.e., completeness and soundness, every valid (honest) transaction is accepted with overwhelming probability, and correspondingly, any faulty (dishonest) transaction is rejected (except with a negligible probability that it could still be accepted). Hence, this allows everyone in the network to verify any transaction without the need to know the content of a transaction, making Zcash have strong privacy while maintaining verifiable transactions [3] [18] [31] [42].

Monero is the implementation of the *CryptoNote protocol* [55], a privacy-preserving protocol. Monero provides complete privacy by default, there is no option to opt in or out of it, hence, it does not have transparency. To provide this privacy, Monero uses a technique called ring signatures, which was introduced in CryptoNote, a way to hide a sender from a transaction. A sender gets put into a group (or "ring") of possible senders. Every signature then proves that someone from this group has the transaction signed without disclosing who [46]. This way it is not possible to tell with certainty who the sender of the group is. Additionally, receivers use a so-called stealth address, also specified in the CryptoNote protocol, which is derived from their public key and is only used once. This results in not being able to see on the blockchain which address actually receives the payment, making it unlinkable. Finally, transaction values are getting hidden by using commitments. Those commitments provide hiding and binding, as introduced earlier, allowing verification of inputs and outputs without revealing their contents. All together, this provides privacy/anonymity for the sender, receiver, and the content of a transaction, all while at the same time the incoming and outgoing transactions can be verified [36] [46].

3 The Penumbra protocol: design and mechanisms

3.1 Motivation

Most of the well-known and widely used cryptocurrencies and their respective systems, such as Bitcoin or Ethereum, are characterized by having publicly available transaction histories. This is useful as transparency increases trust through traceability. But at the same time, it introduces certain drawbacks, such as front-running (observing pending transactions and placing certain transactions in front, i.e., to be executed first to take an economic advantage), the monitoring of asset and activity data of individual addresses, or inferences about trading strategies. In short, transparency takes a toll on privacy. Through this tension between transparency and privacy emerges the motivation to develop blockchain systems that are private but at the same time verifiable. Hence the interest, or rather desire, for privacy in distributed-ledger technologies grows steadily. The Penumbra protocol exactly pursues those goals of being private and verifiable at the same time. It is a fully private, layer 1 (L1) and decentralized exchange (DEX) blockchain for the Cosmos ecosystem that allows inter-blockchain communication (IBC). It provides a privacy-by-default concept, unlike to most other protocols, where privacy is often just offered as an "opt-in" option. It combines elements of Proof-of-Stake (PoS), IBC, and zero-knowledge technologies in order to create an environment where transactions, staking, and governance can take place while respecting privacy. An L1 blockchain is a standalone network, i.e., it does not rely on other blockchains for data availability and security, that has its own cryptocurrency, provides its own consensus mechanisms, and maintains its own ledger of transactions. Some examples of L1 blockchains besides Penumbra (PEN) itself are Ethereum (ETH), Bitcoin (BTC), and Solana (SOL) [1] [39] [48] [52].

While Penumbra is not the only privacy-preserving blockchain, existing systems such as Zcash and Monero have several limitations that restrict their usability in broader decentralized finance ecosystems. A comparison between those two protocols and Penumbra is presented in Chapter 4 (section 4.7). Some key shortcomings are, however, already highlighted in here. Zcash provides privacy only optionally, leading to a smaller anonymity set, thus weakening privacy guarantees. Monero, although it provides privacy by default, falls short in the area of flexible selective disclosure mechanisms for regulatory compliance and auditability. Both Zcash and Monero are single-asset (only have one currency in play) and operate in isolation, without providing any interoperability with other blockchains. All those limitations motivate the design of Penumbra, providing privacy by default, working interoperably by incorporating interblockchain communication (IBC), supporting multiple assets, and having a fine-grained selective disclosure mechanism.

3.2 Overview

This section will give an overview of the architecture and conceptual structure of the Penumbra protocol, building upon the previous section about the motivation for privacy-oriented blockchains. The different mechanisms and components will be discussed in more detail in the later sections and subsections. The intention for this overview is essentially to give an outline of how those different components and mechanisms interact together, resulting in a unified and privacy-friendly system.

Penumbra can be viewed in a modular way based on a few key principles:

- **Privacy by design**: All interactions of users/accounts inside Penumbra are encrypted by default, meaning there is no choice in not having any privacy. The only "exception" is granted for authorities and audits, which can gain a selective disclosure of a user account using special cryptographic keys provided by that user. Those keys, however, only allow to gain a restrictive insight, not everything is revealed, only the necessary part, the rest is still kept private.
- **Verification through zero-knowledge proofs**: Every transaction or even a state change such as updating a private balance, redeeming a reward, or fulfilling a contract condition, can be cryptographically verified without needing to have knowledge or disclose crucial information, such as sender, receiver, or the total amount of value.
- **Interoperability in the Cosmos-ecosystem**: As Penumbra makes use of the Inter-Blockchain Communication protocol (IBC), it allows interaction with other chains. This lets Penumbra do transactions and exchanges between different blockchains.

Penumbra can also be disassembled into different layers for a better overview. Those layers lay out the mechanisms to provide/follow those principles. A rough classification into layers is [1] [39] [52]:

- **Shielded Pool**: The shielded pool in Penumbra represents the heart in terms of privacy. It is a protected area (also called *pool*), where all assets, such as tokens, staking positions or DEX transactions, are stored and transferred while always being encrypted.
- **Cryptographic subsystem**: The cryptographic subsystem contains all mechanisms that are responsible for a transaction to be properly validated (only correct transactions are allowed) and that no double-spending is occurring (prevention).
- **Application layer**: Based on the cryptographic subsystem, the application layer implements functions that all access the same encrypted infrastructure (e.g., staking, governance, and ZSwap/DEX).
- **Consensus layer**: The task of the consensus layer is to reach agreement between all participants in the network, guaranteeing that all users work with the same data (state). This agreement should also be reached reliably, i.e., even under the influence of malicious actors or unreliable network conditions. Penumbra is originally based on Tendermint (nowadays on a fork of it called CometBFT), which is a Byzantine fault tolerance consensus system (BFT) that achieves agreement in a distributed network even when some nodes are malicious or faulty.

Overall, Penumbra aims to pursue the benefits of *public* blockchains, where verifiability and traceability are guaranteed because of their transparency, together with a high degree of privacy and flexibility. The way it handles the combination of these two concepts shows that privacy and traceability, which are usually considered to be mutually exclusive, can be united in a consistent single protocol [1] [39] [48] [52] [56]. However, despite its versatility, Penumbra and specifically the Penumbra token have not gained much popularity and are relatively niche in the market. As of early 2026, the Penumbra token only ranked #4417 on CoinGecko [12], reflecting its limited adoption and visibility.

A quick remark regarding the notion of Penumbra's underlying token must be made beforehand to prevent confusion: in the official Penumbra documentation [39], the token used by Penumbra is called *PEN* (and the delegated token *dPEN*). In more recent articles, as well as in the Penumbra guide [52] the token is named *UM* (and, respectively, the delegation token *dUM*). It is unclear when or why this change in notion occurred, but these tokens are equivalent and represent the same underlying token used by Penumbra. In this thesis, therefore, PEN and UM represent the same token and will be used interchangeably, depending on the citation, in order to remain true to the original sources. An assumption for this change in name is that, in the earlier stages of Penumbra, they used PEN as the name for the token, as well as in the documentation about the protocol. However, they may have changed it to UM during the development while keeping the notion of PEN in their original documentation.

3.3 Cryptographic architecture

3.3.1 Shielded pool

One of the main elements in Penumbra's privacy design is the shielded pool. Penumbra's shielded pool is originally inspired by the Zcash design but modified to allow seamless interoperability across chains [52]. The shielded pool is a protected and private environment inside Penumbra on the blockchain, where all transaction activities and assets are stored in encrypted form. In case the reader is not familiar with the term "assets", these are basically digital representations of value or a right that can be transferred or stored electronically using distributed ledger technology [14]. All transactions inside this pool are represented through *note commitments*. At a high level, a note commitment is created using a commitment scheme $(\mathsf{Commit}, \mathsf{Reveal})$ that binds the value m with a random value r to create the commitment $C = \mathsf{Commit}(m, r)$. This means notes themselves (represented as m in the commitment scheme) are never published on the chain. Instead, the shielded pool records commitments C to notes in a tree structure called a state commitment tree (which will be introduced later on). A *note* is a central concept of Penumbra inside the shielded pool. Conceptually, notes function similarly to unspent transaction outputs (UTXO) in Bitcoin, which represent a spendable value that exists until it is consumed in a transaction. In Penumbra, a note represents the value and the control over the value in an encrypted manner. To create a new note (or rather, record a new note commitment on the state commitment tree), a transaction includes an *output* action, containing the commitment of a newly created note and a zero-knowledge proof that proves that the commitment was honestly created and is recorded in the state commitment tree. To spend an existing note, a transaction is created that includes a *spend* action and a zero-knowledge proof that proves that the note commitment was previously included in the state commitment tree, while not revealing which one. This allows a transaction to be validated without disclosing the sender, receiver, or amount. The commitment itself basically acts as an encrypted "envelope" that is placed in some sort of "shelf" and the zero-knowledge proof guarantees that this is an honestly created envelope [1] [39] [52].

What makes this shielded pool even more beneficial, besides being encrypted (shielded), is that it is also multi-asset-capable, and due to Penumbra's integration into the Cosmos IBC protocol, it can act as an interchain shielded pool, allowing any IBC-compatible asset to be transferred into a protected environment. This is possible due to the use of a state commitment tree (introduced later on) that summarizes all commitments cryptographically and makes the state of the pool efficient and verifiable without revealing any sensitive data. This allows users to perform transactions, swaps, asset exchanges, or staking inside this shielded pool without disclosing any identity or account information, unlike transparent chains where all information is visible publicly on-chain [1] [39] [52].

Inter blockchain communication (IBC)

The inter-blockchain communication (IBC) protocol is a protocol that takes care of authentication and the transport of data between two blockchains. IBC solves a common problem, cross-chain communication (exchanging data and transferring assets between multiple blockchains). It can transfer any kind of data encoded in bytes. IBC is secure in the sense that it ensures the authenticity and integrity of cross-chain messages, it is permissionless, and it is designed to connect independent blockchains into a single interoperable network. This protocol divides concerns into three layers, built on a set of modular abstractions: IBC Clients, IBC Core, and IBC Applications. Those layers guarantee that any pair of chains, without regard to their structure or consensus mechanism, can work and communicate together safely and reliably [49]. IBC is not just an interoperability feature, it is a core design element of Penumbra. Without IBC, Penumbra, or rather the shielded pool, would be limited to only handling one single native asset. This would significantly reduce the privacy benefits that Penumbra provides. Through the integration of IBC, Penumbra can provide a single interchain shielded pool that aggregates liquidity and transactions across multiple blockchains. This increases anonymity sets and makes privacy economically viable.

The IBC protocol allows communication between two blockchains that provide a minimal set of functions, specified in the Interchain Standards (ICS) [19]. Those specifications do not limit the network topology or consensus algorithm, meaning IBC can be used with a wide variety of blockchains [32]. Everstake released a Cosmos Staking Report, in which they stated that over 150 chains support IBC and that this number makes the Cosmos the largest interoperability network by the number of connected chains [21], hence the reason for Penumbra's integration into the Cosmos IBC protocol, providing a multi-asset-capable shielded pool. It should be noted that Penumbra is not limited to the Cosmos ecosystem. It can be used with any IBC-compatible blockchain (the Cosmos SDK just allows for easier integration of IBC as the IBC is tightly integrated into the Cosmos SDK architecture [1]). If the reader wants to learn more about the IBC protocol, they can read more about it on the official IBC website [49].

Functionality of the shielded pool

Penumbra provides interchain privacy by storing all value in a single (interchain) shielded pool, recording any asset from any IBC-connected chain. A user can *perform* three actions in the shielded pool [52].

1. **Deposit**: Any kind of token that is compatible with the IBC can be deposited into Penumbra's shielded pool. This is initiated by the user using a standard IBC transfer from any connected chain, providing a Penumbra address as the receiver. As soon as the funds are transferred to Penumbra, they become *shielded*. All actions performed inside the shielded pool are protected. When transferring funds using IBC, a randomized IBC deposit address is created to prevent linking different incoming transfers, thereby increasing privacy.
2. **Private transactions**: When the funds are transferred into Penumbra's shielded pool, a user can freely do transactions in private. They can transfer, trade, swap, stake, or vote without having to worry about revealing any information about themselves, as everything inside Penumbra is shielded.
3. **Withdraw**: When a user wants to take out their funds from Penumbra, they initiate an outbound IBC transfer to any connected chain. After withdrawing their funds, they lose their *shielded* property, i.e., everything they do afterward will not be shielded by Penumbra anymore, hence, depending on the target chain, their action may become public again.

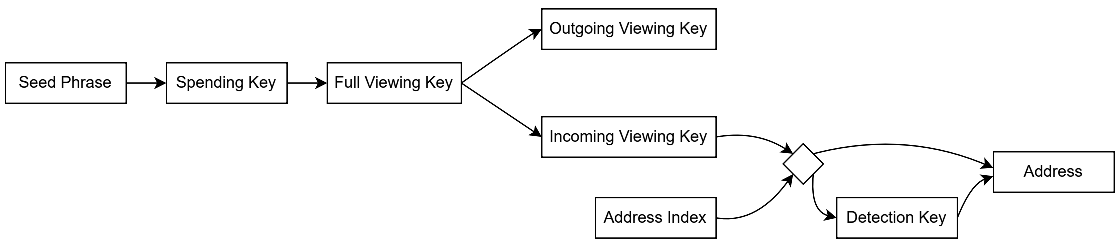

Figure 3.1: User-Visible Key Hierarchy

As mentioned in the “deposit” action, Penumbra uses IBC deposit addresses (which are automatically generated in Penumbra) to increase privacy when making IBC transfers into and out of Penumbra. This is done as an IBC deposit has to be performed outside, on the (public) counterparty chain. This can reveal the destination address and allow the possibility of linking addresses if the same address was used multiple times. Similar linking can happen when withdrawing without an IBC address. Generally, Penumbra recommends, in order to maximize privacy, to store funds on Penumbra, inside the shielded pool, and only withdraw once an action on another chain has to be performed [52].

In the upcoming subsections, different underlying cryptographic constructions will be explained in more detail, which are used inside this shielded pool for providing security and privacy.

3.3.2 Address and key derivation

The key and address architecture of the Penumbra protocol plays an important role in privacy. Penumbra deliberately separates cryptographic spend authorization, transaction monitoring, and addressing such that it prevents linkability, minimizing information leakage while supporting selective disclosure, delegation, and usability. Transparent blockchains, on the other hand, can reveal transaction information in different ways, depending on the model they use. Account-based blockchains, like Ethereum, record balances and transaction history for each address publicly on the ledger. Revealing an address, for example, to receive payment, can expose the complete account activity. UTXO-based blockchains, like Bitcoin, do not maintain balances per account but record transaction inputs and outputs publicly on-chain. This makes it possible to link addresses and reconstruct activity by analyzing UTXO flows. Penumbra prevents those privacy leaks by segregating cryptographic keys for spending, monitoring, and receiving funds, reducing the overall privacy impact in case of compromise. Penumbra allows multiple publicly unlinkable addresses that can be controlled by the same account [52]. These addresses are derived from spending keys via a series of intermediate keys representing different levels of capability attenuation. This provides high data protection without compromising the verifiability or consistency of the system. Additionally, since all addresses that belong to a single user use a shared incoming viewing key, scanning the blockchain does not increase the cost for the number of addresses in use [39].

Key hierarchy

The whole key hierarchy is deterministically derived from a single root secret (typically a seed phrase, which is a sequence of human-readable words, serving as a backup for the root secret). From this root secret, spending keys, various viewing keys, as well as diversified addresses and detection keys are generated. The design of the key hiearchy is inspired by the Zcash-sapling protocol [31] and adapted to suport Penumbra’s interoperability, IBC features and selective disclosure. Penumbra uses BLS12-377 elliptic curves (compared to BLS12-381 used in Zcash), Poseidon [27] as a hash and PRF (as noted in the Penumbra documentation [39]), decaf [29] (specifically decaf377) as the embedded group, and supports fuzzy message detection. The figure 3.1 shows only the user-visible parts of the key hierarchy, i.e., providing an external functional point of view.

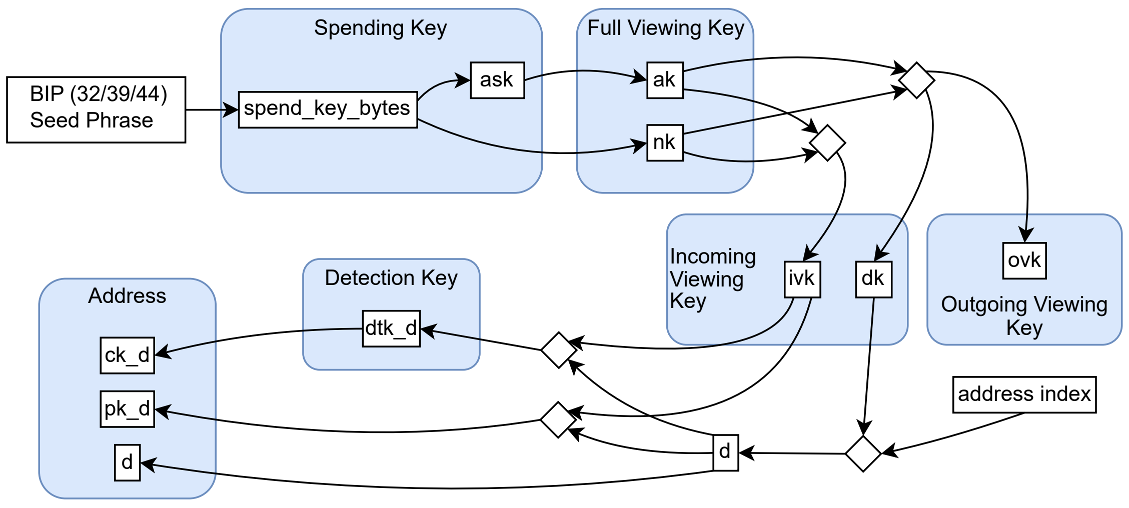

Figure 3.2: Internal view of the Key Hierarchy

Key components include:

- **Seed Phrase**: The seed phrase is the root key material. From this root seed phrase, multiple addresses with separate spending authority, can be derived.
- **Spending Key**: Represents the authorization to spend funds of the associated address. It is also used for deriving further keys.
- **Full Viewing Key**: Provides full insight into transactions of an address without providing spending capability.
- **Incoming/Outgoing Viewing Keys**: The full viewing key is split for selective disclosure. The outgoing viewing key provides the ability to view only outgoing transactions and conversely the incoming viewing key provides the ability to view only incoming transactions.
- **Detection Key**: Enables detection of transactions belonging to the associated address. It is partially derived from the incoming viewing key, therefore it has similar functionality. However, the detection key is only used for detecting, without giving any insights into the transaction details. As this key is not shared between diversified addresses, it allows for fine-grained control of delegation.
- **Arrows and diamonds** ◇: Arrows represent key derivation steps, and diamond represents key derivation steps that combine multiple components.

Spending keys

Internally, each of those keys has different components, visible in figure 3.2. As in the previous figure, arrows represent key derivation steps, and diamond boxes (◇) represent key derivation steps that combine multiple components. Every internal key component is represented with a box.

The spending key (containing *ask*) is the (private) key that represents full control over a user's wallet, responsible for all spending authorization and used to derive further keys as well as to create nullifiers. The *ask* part of the spending key is a private authorization key, which is used to authorize spends by enabling the creation of zero-knowledge proofs and spend authorization signatures. Now, since the spending key provides authorization to spend, it has to be handled with the highest confidentiality [39]. From figure 3.2

it is visible that Penumbra uses for the creation of spending keys a hierarchically deterministic procedure based on the BIP specifications/standards (such as BIP-32, BIP-39, and BIP-44). BIP stands for Bitcoin Improvement Proposal and is a technical document providing information about Bitcoin, describing a new feature for Bitcoin or its processes or environment. It contains general community guidelines up to major changes to the protocol. Each BIP represents one key change to the blockchain [15]. It might sound confusing that Penumbra uses BIP, as the description makes it seem like a Bitcoin-only standard. But many blockchains reuse these standards because they are well designed and widely adopted.

Using those BIP standards, a mnemonic seed phrase (consisting of 12 or 24 words) is created and used to generate deterministic key derivations along a predefined path. This procedure allows for hierarchical key management as well as the ability to restore all keys created from the original seed phrase. Inside this scope, Penumbra defines a spend_key_bytes that is used as input material for the internal key hierarchy. From those bytes, the actual spending key *ask* is created, which in turn is used for the creation of two more central keys: the spend verification key *ak* and the nullifier key *nk*. The spend verification key *ak* is a public verification key corresponding to *ask* and is used by validators to verify spending authorization. The nullifier key *nk* is a secret key that is used to deterministically derive nullifiers by using it as an input to a pseudorandom function. Together, this key separation allows control over assets to be cryptographically separated from purely observable access [1] [39].

As mentioned, the spending key has the highest permission level, so compromising it would result in the loss of control of all associated sources. Using a well-established standard such as BIP ensures that users can safely recreate and restore their wallets on new devices, as long as the original mnemonic phrase is retained.

Viewing keys

The viewing keys allow insight into transactions without having the ability to spend funds. This allows privacy and transparency to be separated cryptographically from each other, allowing different levels of insight into an account [1]. Penumbra defines three types of viewing keys, which are all hierarchically derived from the spending key [39]:

- **Full Viewing Key**: The full viewing key contains the spend verification key (ak) and the nullifier key (nk). The spend verification key allows cryptographic verification (used in zero knowledge proofs) and the nullifier key is used for the unique identification of spent notes (via a PRF). The nullifier key is derived from the spend_key_bytes using a PRF-expand function (based on BLAKE2b-512). The full viewing key allows complete insight into the incoming and outgoing notes of an account, but does not grant authorization to spend.
- **Incoming Viewing Key**: The incoming viewing key (ivk) is derived from the full viewing key. This key is used for identifying and encrypting incoming transactions (scanning the chain). In combination with a diversifier key (dk), it is possible to create multiple diversified addresses that are publicly not linkable to each other. The key uses symmetric encryption to allow decryption of incoming notes.
- **Outgoing Viewing Key**: The outgoing viewing key (ovk) is derived from the full viewing key. This key enables transparency for the sender without the need for the transaction details to be publicly available on the blockchain. This key allows one to recreate outgoing transactions (-details), for example, for accounting purposes, while keeping all information hidden from third parties and without damaging the privacy of the receiver or the network itself. It is used for symmetric encryption of outgoing notes and memos.

The separation of the full viewing key into an incoming- and outgoing viewing key allows for selectively handing certain viewing keys to parties (such as the tax authorities or audit systems) to provide verifiable but limited insight into activities (an important concept for compliance-friendly private systems).

Addresses and detection keys

Each account can have multiple unlinkable addresses, referred to as diversified addresses, generated using a diversifier key. The diversifier key dk allows deriving multiple diversifiers d (combining dk with the address index, see figure 3.2). The diversifier, a 16-byte tag, serves as a parameter for generating an individual address. This allows creating up to 2^{128} different addresses for the same account, which are cryptographically independent of each other [39].

Every diversified address has a corresponding detection key that enables efficient detection if a note belongs to the address without disclosing the full viewing key. The detection key dtk_d is used to process "clues" included in each transaction (via the associated clue key ck_d) to detect relevant notes probabilistically. This mechanism ensures that users can identify their incoming notes while preserving the privacy of other transactions on the chain [39].

Penumbra allows outsourcing probabilistic transaction detection to third parties through fuzzy message detection (FMD), where a scan service can check a note without giving away the actual address, using this detection key. Similar to a bloom filter, there can be false positive detection but no false negatives. This means the detection will find all relevant transactions but also some amount of unrelated cover traffic. However, it is important to mention that giving out too many detection keys to a single detection entity might hold a subset of the privacy implications [39]. Although diversified addresses cannot be publicly linked cryptographically, in certain scenarios an empirical link can arise. If a user gives a detection entity multiple detection keys, the detection entity can use the known clue keys (ck_d) to derive that multiple addresses belong to the same user. This happens because the detection entity, when it detects transactions and reports them to the same user, empirically knows that the detection keys dtk_d are connected. When the detection entity recognizes that two addresses belong to the user, it can link them since the clue key is present in both addresses and is solely based on the detection key [39].

A simplified example is given in the Penumbra protocol documentation [39]: let's assume a user with two diversified addresses, addr_1 and addr_2, gives the associated detection keys $\mathsf{dtk}_{\mathsf{d}1}$ and $\mathsf{dtk}_{\mathsf{d}2}$ to a single detection entity. The detection entity detects relevant transactions using the clue keys $\mathsf{ck}_{\mathsf{d}1}$ and $\mathsf{ck}_{\mathsf{d}2}$ and reports the detected transactions for $\mathsf{dtk}_{\mathsf{d}1}$ and $\mathsf{dtk}_{\mathsf{d}2}$ back to the user. Now the detection entity can simply observe that transactions related to $\mathsf{ck}_{\mathsf{d}1}$ and $\mathsf{ck}_{\mathsf{d}2}$ are reported back to the same user, resulting in the knowledge that the addresses linked to these detection keys belong to the same user. This demonstrates the potential for linkability, since the diversified addresses can be linked by the detection entity through the detection keys, from which the clue keys are derived.

It is advised, when the user wishes to delegate the detection, to use multiple third parties (detection entities) using detection keys in order to mitigate the linkability of diversified addresses. This way, the detection keys are distributed between multiple detection entities (instead of just one), reducing the risk that a single entity has enough keys to link diversified addresses [39]. A central feature of Penumbra is that the scanning of transactions for all diversified addresses of an account can be done simultaneously using the same incoming viewing key. Because of that, it does not lead to additional cost for every single address, which makes Penumbra particularly efficient compared to other privacy-oriented blockchains [1] [39].

The (final) Penumbra address is created using three components:

$$\mathsf{Address} = (\mathsf{d}, \mathsf{pk_d}, \mathsf{ck_d})$$

The first component is the diversifier d (a 16-byte tag). With this diversifier, the diversified basepoint $\mathsf{B_d}$ can be created using $\mathsf{B_d} = \mathsf{H}^{\mathsf{d}}_{\mathbb{G}}(\mathsf{d})$, where $\mathsf{H}^{\mathsf{d}}_{\mathbb{G}} : \{0,1\}^{128} \to \mathbb{G}$ performs hash-to-group for decaf377. This diversified basepoint, together with the incoming viewing key (ikv), is used to derive the second component, the transmission key $\mathsf{pk_d}$, which allows the encryption of notes sent to the address. The last component is the clue key $\mathsf{ck_d}$ used for the probabilistic detection of transactions. Those addresses (the triple) are encoded in a secure format in order to prevent attacks through address manipulation (such as collision or prefix attacks). Besides the addresses being cryptographically completely different from each other (therefore, they cannot be publicly linked), they all can be controlled from a single account since they are derived from the same incoming viewing key (ivk) and detection key (dk) [1] [39].

Transaction cryptography and payload keys

The transaction cryptography represents an important element in the Penumbra protocol regarding confidential communication within the network. While zero-knowledge proofs provide correctness of “hidden” transactions, transaction cryptography encrypts notes, memos, and swaps using symmetrically derived one-time keys. A key exchange between sender and recipient happens using Diffie-Hellman to derive a shared secret that both sides can compute without revealing their private keys. Once both sides know the shared secret, Penumbra derives symmetric keys from that shared secret (and other inputs) and uses them to encrypt parts of the transaction [39]. This does not only keep movement and ownership history hidden, but also the user data stays private [39]. Memos are small message fields that can be included in a transaction. There is no need to use them in order for the protocol to work, but they provide a way to communicate between sender and recipient, similar to an annotation in a bank transfer.

The main component of this architecture is the Action Payload Key (APK). Every action, like creating or spending a note, generates a unique APK. This key is derived from a shared secret (using the Diffie-Hellman key exchange between sender and receiver). This key is used to encrypt action data from the corresponding transaction. Thanks to this key, only the receiver can decrypt relevant information, without others (third parties or validators) being able to gain knowledge of the data [39].

Additional keys are derived from the APK in order to provide more specific access rights. Such keys are [39]:

- The **Outgoing Cipher Key** is created using the APK together with the sender’s outgoing viewing key (OVK). The sender can use this key to decrypt their own past transactions. This allows the sender to collect information about their spending history without compromising the privacy of other involved users.
- The **Wrapped Memo Key** is used for the secure inclusion of memos in the transactions. The memo key is encrypted with the APK so that only the receiver can read the content of the memos. This allows for the creation of confidential communication between transaction partners.
- The **OVK Wrapped Key** encrypts the actual APK with the help of the OVK of the sender. Thanks to this, the sender can access the transaction details later on, even when the original temporary keys are not available anymore. At the same time, the protection of the spend keys is maintained, as the OVK does not give any permission to spend funds.

This multi-layered key structure allows for a fine-grained level of access, which guarantees data protection as well as data traceability. Only authorized parties, such as the receiver, sender, or auditors with special viewing keys, can decrypt and view certain parts of a transaction. Validators can only validate the cryptographic correctness of a transaction, without the specific need of information about participants and the amount of value.

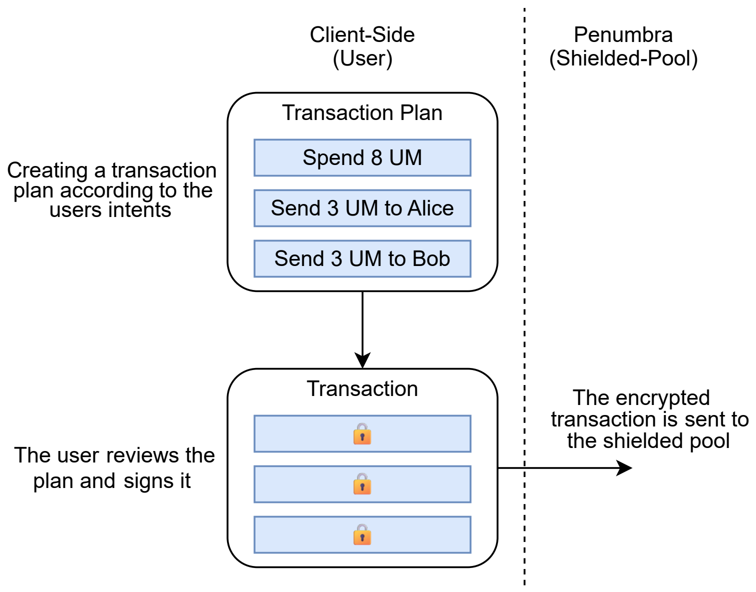
Client-Side
(User)
Penumbra
(Shielded-Pool)
Transaction Plan
Spend 8 UM
Send 3 UM to Alice
Send 3 UM to Bob
Creating a transaction plan according to the users intents
Transaction
The user reviews the plan and signs it
The encrypted transaction is sent to the shielded pool

Figure 3.3: Creation of a Transaction Plan

Transaction plan, perspective, and views

In the previous subsections, we discussed action payload keys and the related cryptographic mechanism that defines who can decrypt specific transaction data. Penumbra also defines what every participant can learn about a transaction. This is done through the concepts of transaction plan, transaction perspective, and transaction view.

The *transaction plan* describes everything a user wants to do in a transaction. It can contain information such as how much is to be sent, who it is to be sent to, if it is a swap or another action, and the content of the memo. That information is in plaintext and understandable by a human. This allows a user to see the plan and acknowledge the changes it would create. If they are in agreement with what would happen, they can approve it before they sign and submit the transaction [52]. After the approval from the user about the plan, they add their signature and create client-side proofs. Everything gets assembled together, and the transaction is sent to the network. Once the transaction has left the user, it is opaque, i.e., most actions and their information are shielded, encrypted, and private [52]. This is illustrated in the figure 3.3.

As discussed in the previous subsection, Penumbra allows for fine-grained control of privacy through the usage of different "level" keys. This selective disclosure of transaction content is achieved (besides using those specific keys) through the *transaction perspective*. In order to be able to decrypt certain content of a transaction, the transaction perspective contains a collection of scope data that enables this. Included are specific keys to decrypt certain parts of the transaction. For example, if the user only wants to disclose one single note of the whole output from the transaction, they can achieve this by providing only that corresponding key to the transaction perspective. The use of transaction perspective enables selective disclosure without giving away the full viewing key [52]. The transaction combined with the transaction perspective generates the *transaction view*, which shows the filtered, decrypted transaction according to the transaction perspective. This is visualized in figure 3.4.

This entire section on address, key derivation and architecture emphasizes the principle that Penumbra follows: "Privacy by design/default". Everything is encrypted. However, special mechanisms still allow for selective disclosure without compromising the user's or transaction's complete privacy.

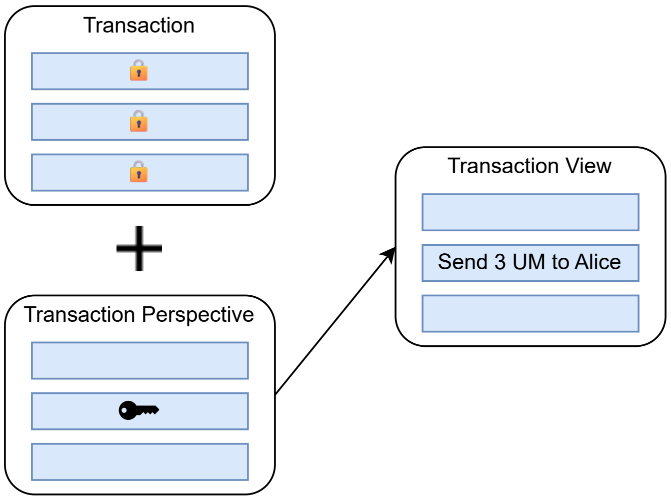

Figure 3.4: Transaction Perspective leads to Transaction View

3.3.3 Note-, balance commitments and nullifiers

We now have seen the encrypted transaction architecture of Penumbra. To preserve confidentiality while at the same time ensuring the correctness of state transition, Penumbra employs additional cryptographic primitives. The main goal of Penumbra is to keep transactions fully private without breaking integrity and the ability to double-spend. Penumbra achieves this through the combination of two cryptographic concepts: *Commitments* (hiding and binding values) and *Nullifiers* (for representing spent values) [1] [31] [52].

Note commitments

The basics of commitments were already discussed in section 2.3, definition 2.3.2. Penumbra implements and extends the note commitment scheme from the Zcash sapling protocol [31]. Notes are never stored in plaintext on the chain. Instead, they are only recorded in the state commitment tree (more details on this in 3.3.4) through note commitments. Every note commitment represents an encrypted unit of value and ownership in the shielded pool. This means the note commitments are publicly available on the SCT but through the commitment, the notes themselves (i.e., the content of the note commitment) are kept private (due to the hiding property of commitments). The note commitment is created using a Poseidon hash. Although the Poseidon hash is not a commitment, it is a hash function, however, as it is used together with a blinding factor, it fulfills the properties for a commitment scheme. The Poseidon hash guarantees binding through its collision resistance (making it computationally infeasible to find two notes that create the same hash), and hiding is provided through the use of a binding factor. Additionally, Poseidon hashes are zk-efficient (which presents the main reason for their usage). The note commitment in Penumbra is created as follows [39] [52] :

$$\mathsf{note\_commitment} = \mathsf{hash}(\mathsf{ds}, (\mathsf{rcm}, \mathsf{v}, \mathsf{assetID}, \mathsf{B_d}, \mathsf{pk_d}, \mathsf{ck_d}))$$

Let's break down what all those parameters actually mean. The *ds* stands for the domain separator, defined as the $\mathbb{F}_q$ field element, which is essentially a hashed constant that is used to separate cryptographic domains. For example, here the *ds* is derived using the so-called BLAKE2b-512 hash with the "penumbra.notecommitment" value. This is done in order for the Poseidon hashes to not collide with other Poseidon hashes used for other purposes in the protocol. The *rcm* stands for the note-blinding factor. It plays a crucial role in privacy and security. It is derived from a seed value in the note itself. The *rcm* acts analogously to a PRF key and introduces randomness, providing exactly what commitments do: hiding and binding. Additionally, it ensures uniqueness (even if a note has the same value and asset ID, the rcm differentiates them). The *v* stands for the value of the note, and *assetID* represents the asset ID (e.g., what kind of token). The last three parameters are B_d, which is the diversified basepoint (derived from the diversifier *d* from the address), pk_d, which is the transmission key, and ck_d, which is the clue key. The last two keys are part of the address of an account and represent the address. Those note commitments are recorded in the state commitment tree. As those note commitments are created using the Poseidon hash, they are not homomorphic, but this is not of concern as their only use is to show (without providing exact information) ownership of a note, so there is no need for (homomorphic) operations. The state commitment tree only contains the commitments, not the notes themselves. Hence, the blockchain can check the existence and correctness of a note without knowing the value or owner of a note [1] [39] [52].

Balance commitments

To prove a transaction's value correctness without disclosure, a different commitment, called a balance commitment, is required. Those balance commitments are homomorphic and based on the Pedersen commitments [47] (homomorphic refers to the property that operations on ciphertexts correspond to operations on plaintexts). Balance commitments ensure cryptographically that every transaction (containing a sequence of actions, such as spend, output, delegate, or swap) does not *create* or *destroy* value while keeping the values confidential, guaranteeing value conservation. Those balance commitments are constructed in such a way that they are additive homomorphic commitments, allowing to confirm that a transaction at the end has an overall balance of 0 (i.e., a balance commitment to 0, so no value is created or destroyed), without revealing the underlying values. Such a balance commitment could be expressed as $C_{\mathsf{bal}} = \mathsf{Commit}(\mathsf{value}, \mathsf{blinding})$, where value represents the value of the note and blinding is a random blinding factor. Those commitments are based on elliptic curve cryptography and use a blinding factor to ensure confidentiality. The balance commitments are calculated for all inputs and outputs of a transaction. Thus, for a transaction with multiple inputs and outputs, the balance commitments are summed:

$$\sum C_{\mathsf{in}} + (-\sum C_{\mathsf{out}}) = C(0)$$

where $C(0)$ is the commitment to 0. The prover demonstrates that the sum of all balance commitments is equal to the 0 commitment. Those balance commitments are essential for the security. They effectively link the validity of local state transitions, provided by each action's proof statement, to the validity of the global state transition [39].

Nullifiers

A common problem with cryptocurrencies is the potential for double-spending. Other cryptocurrencies prevent this through consensus mechanisms, which ensure that there is a single agreed-upon transaction history (e.g., Proof of Work in Bitcoin or Proof of Stake in Ethereum). In Penumbra, the previously shown commitments alone do not prevent double spending, as the same encrypted note could theoretically be handed in multiple times. Penumbra addresses this problem using so-called nullifiers, which are deterministically calculated, untraceable *identifiers* for each note [39] [52]. They function like a serial number, which is consumed when a note is spent (the nullifier will be registered on the ledger), while still keeping the note itself encrypted. The note commitments mentioned previously are so-called state fragments (introduced prior to their definition in 3.3.4), which are fragments of a state and stored locally on the client side. Only when they are used/spent are they committed to the SCT (using a state commitment).

Similar to the commitments, the Poseidon hash is used to create nullifiers. The nullifier *nf* is created as follows:

$$\mathsf{nf} = \mathsf{hash}(\mathsf{ds}, (\mathsf{nk}, \mathsf{cm}, \mathsf{pos}))$$

In the section 3.3.2, different kinds of keys were introduced. One of those keys was the nullifier key *nk* inside the full viewing key. This *nk* is used together with the state commitment *cm* and position *pos* inside the state commitment tree of that note. *ds* stands for the domain separator (but this time it is derived using the "penumbra.nullifier" value). The note commitments can be viewed as state fragments, pieces of the state stored locally on the client. This means nullifiers are not publicly derivable from the state fragment. Only with the use of the nullifier key *nk* is it possible to derive the nullifier associated with this state fragment. Using the nullifier key *nk*, the Poseidon hash functions as a PRF (definition 2.3.4), *nk* represents the secret key, and together with additional public inputs (*cm*, *pos*), they are fed into the hash to produce a pseudorandom field element. When a state fragment is created (i.e., a note commitment, but it could also be a swap), its state commitment is revealed on-chain, i.e., on the SCT, and when it gets consumed, the nullifier is revealed. This action is unlinkable without knowledge of the nullifier key. Each positioned state commitment has an associated nullifier. A state commitment is positioned when it has been included in the SCT and its position is fixed. So the nullifier is a specific instance of a state fragment included in the chain state and not just a property of the content of a state fragment [39]. Now, as soon as a nullifier appears on the blockchain, it means that the note belonging to that nullifier is spent. This means every future transaction containing the same nullifier will be rejected. This mechanism guarantees the non-reusability of all expenses [1] [31] [52].

A simplified example, for a transaction with sequences of events for better understanding, will be shown later in section 3.4, after the introduction of the state commitment tree and the zero-knowledge proofs.

3.3.4 State commitment tree

The state commitment tree (SCT) of Penumbra functions like a memory rather than as a controller. The SCT records private "state fragments", allowing them to be publicly verifiable while maintaining confidentiality. Penumbra records two kinds of state fragments: notes and swaps [39].

In Penumbra there are two different states, the public shared state, which is recorded on-chain in a key-value store, and the private per-user state, which is recorded on end-user devices and only committed to on-chain. The public state works similarly to blockchains that use the account-model, it is a global and mutable state, which will get modified by transactions during execution. The private state, on the other hand, similar to the UTXO-model, is composed of immutable, composable state fragments. Whenever a transaction is executed, existing state fragments will be consumed and new state fragments will be created, all while being private and using zero-knowledge proofs to prove that the newly created fragments are valid [39].

An important aspect of the SCT is that whenever new notes are created, they are never recorded in plaintext on the chain. Instead, a cryptographic note commitment is recorded in a private state fragment. The SCT manages (state) commitments to those private state fragments. To justify that a state fragment has been previously validated by the chain, a zero-knowledge proof demonstrates that the commitment to the state fragment is included in the SCT. This process allows access to earlier state fragments without disclosing which state fragment is referenced. However, this alone is not enough. It also must be shown that the state fragment that is planned to be consumed by a transaction has not already been consumed by another transaction, representing the double-spend problem. This is solved using the nullifiers, introduced in the previous subsection. Each state commitment is associated with a nullifier. Transactions reveal the nullifier of a state fragment whenever it is consumed. The chain itself holds a publicly available set of nullifiers that have been revealed in the public state in a jellyfish Merkle tree [24]. This allows checking if a note has already been spent (preventing double spending) by observing if the corresponding nullifier exists in this set [39].

Now, since the users have the private state fragments only on their side, it requires a role reversal relative to the conventional use of Merkle trees in blockchains. The user itself has to make the proof to a full node. This requires every user to have a synchronized local instance of the state commitment tree in order to get the required data for the proofs. This may sound like it would introduce some complications, as it seems that a user always needs to have an up-to-date instance of the full SCT, but this hurdle is solved by instantiating the SCT using a tiered commitment tree (TCT), an append-only, zk-friendly Merkle tree designed for efficient, filterable synchronization. Resulting in a user only having to synchronize the parts of the tree that are relevant to them (synchronizing selectively subtrees), without needing to maintain the full tree structure. For the other parts, it is enough to rely on the compressed representation using hashes [39].

The reason for Penumbra to use an append-only state commitment tree instead of an account-model presumably stems from the efficient inclusion proofs that the SCT provides, which are required by zero-knowledge circuits, while it preserves privacy at the same time. This design avoids revealing global balances and supports parallel verification of state fragments. Additionally, due to the fixed positioning of commitments, it allows us to derive unique nullifiers, which prevent double-spending without compromising anonymity.

Tiered commitment tree

The tiered commitment tree (TCT) is the data structure used for the SCT. It is an append-only, zk-friendly Merkle tree. The TCT provides some unique features [39] [48]:

- **Quaternary**: Every node of the TCT has four children. This reduces the hash operation overhead for each insertion, as the tree depth is shallower while having the same capacity for leaves. Research around the Poseidon hash, which Penumbra uses, also recommends using a quaternary tree specifically, as it balances the proof depth and proof size (binary tree $\rightarrow$ too many expensive hash operations and e.g., an octal tree $\rightarrow$ too large proofs).
- **Sparse**: A user/client does not need to have the entire view of the tree. It is enough to only represent the state commitments relevant to the client's state. All other parts ("forgotten" commitments and internal hashes) are summarized in a single summary hash.
- **Semi-Lazy**: Internal hashes (especially those along the frontier of the tree) are only computed when demanded, e.g., when a proof of inclusion is required or computing the root hash of the tree. This increases the performance during synchronization.
- **Tiered**: The tree is actually a nested tree of trees, to be precise a triply-nested tree (tree-of-trees-of-trees). The global tree contains epoch-trees (up to 65,536 epoch trees), where each of those trees contains at their leaves block trees (up to 65,536 block trees), where each of those trees again contains at their leaves note commitments (up to 65,536). The structure can be seen in figure 3.5. Due to this structure, it allows a client to avoid performing hashes to insert the state commitments in blocks or epochs when it knows it did not receive any notes or swaps. When an entire block or epoch does not contain anything of interest for that client, it does not need to construct the span of the commitment tree. It is enough to insert the single summary hash for that block/epoch.
- **Incremental persistence**: The tree can be implemented in a way that only requires serializing changed parts of the tree. The structural immutability of the tiered commitment tree can be taken as an advantage to serialize to disk only the portions that are changed, reducing memory and I/O overhead.

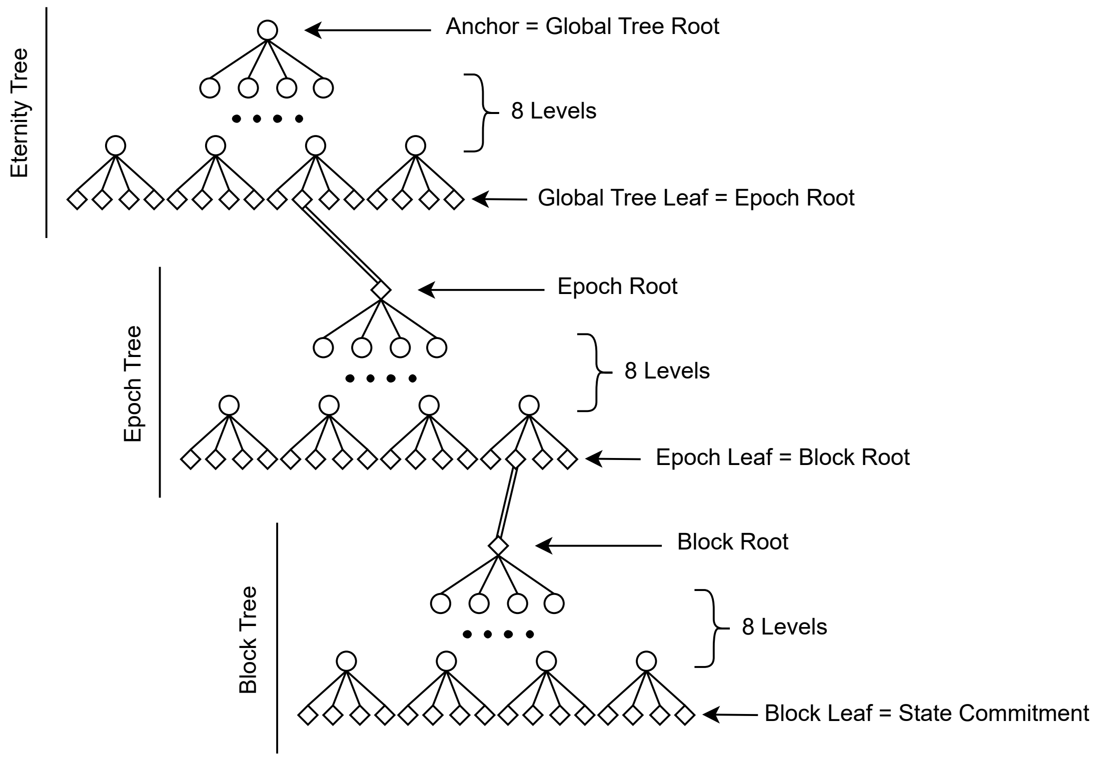

Figure 3.5: Tiered Commitment Tree structure

3.3.5 Zero-knowledge proofs (Groth16)

As Penumbra is a “privacy-by-default” protocol, ensuring transactions, staking, voting, and other actions are verifiable without being publicly traceable. To achieve this, it requires some sort of way to prove something without disclosing vital information such as sender, receiver, amount, or asset type. In short, a verifier should be able to verify the correctness of a transaction without the need to know specific information about that transaction. This is exactly the part where zero-knowledge proofs (ZKPs) come into play (see definition 2.3.3), as they provide the ability to prove a statement (meaning it can be verified) without disclosing any crucial (internal) information. ZKPs allow the coexistence of verifiable correctness and data privacy [48] [52].

Penumbra uses Groth16, which is a specific kind of zero-knowledge proof, a “succinct non-interactive argument of knowledge” zero-knowledge proof (zk-SNARK). It was developed in 2016 by Jens Groth [28]. Similar to the basic zero-knowledge proofs, it allows verifying a statement about secret information without needing to know the information itself. The benefit of Groth16 is its shorter proofs and fast verification compared to the basic zero-knowledge proofs, but this comes at a cost of requiring a trusted setup [31].

Groth16 proofs allow completely private transactions inside Penumbra’s shielded pool. The goal is to validate transactions without disclosing any sensible data, meaning the network can validate if a transaction is correct without giving away sender, receiver, or value. Originally Groth16 was developed with the Zcash-sapling protocol [31] in mind, but Penumbra has expanded the architecture for multiple assets, usage with the Cosmos-IBC protocol, and support for shielded environments [1] [31].

Penumbra uses multiple specialized zero-knowledge circuits (Groth16 circuits) for different transaction types. A zk-circuit describes the calculation rules that determine if a transaction is correct. Those rules are represented in a form such that they are cryptographically verifiable, proving that a transaction is valid without disclosing details about it. The three core proofs in Penumbra’s zero-knowledge system are:

- **Spend Proofs**: Spend proofs prove that a user owns a valid note and is eligible to spend it, without revealing the user’s identity or the note’s value. At the same time, the corresponding nullifier is checked if it has been correctly calculated and is not already published in order to prevent double spending [39] [52].
- **Output Proofs**: Output proofs guarantee that newly generated notes (the transaction output) are cryptographically valid and correctly bounded, i.e., they refer to correct asset types and values for the corresponding protocol balance [1].
- **Binding Proofs**: Binding proofs ensure that the sum of the spent value is equal to the generated value. This value balance principle prevents faulty transactions from generating illegitimate assets [52].

For secure and compact commitments, zero-knowledge proofs use elliptic curves. The most commonly used elliptic curves are the BLS curves (Boneh–Lynn–Shacham [5]). Those elliptic curves are special, as they support “pairings”. Those pairings allow for compact and efficient cryptographic methods, such as zero-knowledge proofs. Zcash, the “inspiration” of Penumbra, uses so-called BLS12-381 elliptic curves. Penumbra itself uses a different version, the BLS12-377 elliptic curves. This difference/decision is being made as BLS12-377 provides compatibility with future universal zero-knowledge proof systems (such as Halo2, Nova, or PLONK) and to make the underlying pairing operations more efficient [1] [52].

An important point in using Groth16 is that it requires a trusted setup. As Groth16 requires a circuit-specific setup, Penumbra uses a multi-party computation (MPC) procedure for generating a common reference string (CRS). This setup procedure orients itself from the Zcash “Powers of Tau”-Ceremony [45], which guarantees that not a single party controls the setup parameters. Looking to the future direction of Penumbra, the Penumbra developer team plans to change to a universal setup system that removes the dependencies on individual trusted setups [1] [53].

Through the combination of Groth16 proofs (zk-proofs), commitments, and nullifiers, Penumbra creates a verifiably correct and encrypted ledger architecture. This encrypted and verifiable ledger forms the foundation for key protocol components such as staking (3.5.1), governance (3.5.2), and decentralized exchange (3.6) [1] [39] [53].

3.4 Protocol rundown

Sufficient concepts and mechanisms have now been introduced to present a simplified example illustrating what a transaction between two users would look like. The example shows the steps involved when *Alice* wants to send *Bob* 10 UM. We assume that Alice and Bob already have a wallet (skipping the creation process).

Since Bob has a wallet, he also has one or more Penumbra addresses, each consisting of a diversifier d, a transmission key $\mathsf{pk_d}$, and a clue key $\mathsf{ck_d}$. Those three parameters represent a long-lasting diversified address $= (\mathsf{d}, \mathsf{pk_d}, \mathsf{ck_d})$. Such an address is publicly available, and Alice can use it to generate an encrypted note for Bob. We also assume that Alice already has a note: $\mathsf{note_{Alice}}(10, \mathsf{assetID}, \mathsf{rcm_{in}}, \mathsf{pk_d}^{\mathsf{Alice}})$, together wit the corresponding spending key (private), and the nullifier key (nk), which she will use for calculating the corresponding nullifier once the note is spent. We assume the note (respectively the note commitment) is already recorded in the SCT, i.e., $\mathsf{cm_{in}} = \mathsf{hash}(\mathsf{ds}, (\mathsf{rcm_{in}}, 10, \mathsf{assetID}, \mathsf{B_d}^{\mathsf{Alice}}, \mathsf{pk_d}^{\mathsf{Alice}}, \mathsf{ck_d}^{\mathsf{Alice}}))$.

Alice wants to send Bob 10 UM. She selects her note ($\mathsf{note_{Alice}}$) with a value of 10 UM as the input for the transaction. Then she calculates the nullifier for the input note, $\mathsf{nf} = \mathsf{hash}(\mathsf{ds}, (\mathsf{nk}, \mathsf{cm}, \mathsf{pos}))$, which prevents double spending. Additionally, she creates an (input) balance commitment C_{in}. Alice then creates a new note for Bob: $\mathsf{note_{Bob}} = (10, \mathsf{assetID}, \mathsf{rcm_{out}}, \mathsf{pk_d}^{\mathsf{Bob}})$, representing the note that Bob will later on own. She also computes the corresponding note commitment $\mathsf{cm_{out}} = \mathsf{hash}(\mathsf{ds}, (\mathsf{rcm_{out}}, 10, \mathsf{assetID}, \mathsf{B_d}^{\mathsf{Bob}}, \mathsf{pk_d}^{\mathsf{Bob}}, \mathsf{ck_d}^{\mathsf{Bob}}))$, and generates for this output, an (output) balance commitment C_{out}. Penumbra requires that no value is created or destroyed, i.e., that $C_{\mathsf{in}} + (-C_{\mathsf{out}}) = C(0)$, which is essentially a commitment to 0 (in the example $10 + (-10) = 0$). The sum (the total balance commitment) will be referenced in the zk-proof, showing that the equation holds without disclosing the values. Alice generates a zk-proof that is succinct and efficiently verifiable, ensuring practical performance in terms of storage and transmission over the network. The zk-proof shows:

1. Alice knows the private spending key for the note (legitimacy for spending)
2. The input note exists in the SCT (inclusion proof)
3. The nullifier is correctly derived from the nullifier key
4. The balance of the transaction is 0
5. The note commitment for Bob is correctly created

All information used inside this zk-proof is not disclosed, but due to the nature of zk-proofs, they can still be verified (no knowledge of the content needed). Finally, the transaction is constructed. It contains the nullifier for the input note, the note commitment for Bob, the balance commitments, the zk-proof, the Merkle path of the input note (inclusion proof in the SCT), and a clue generated from Bob's clue key (allowing Bob to detect the newly created note later on without exposing any secret information).

The transaction is then broadcast and validated by the validators, where they check for the validity of the zk-proof, that the nullifier has never been seen, that the note commitment is correctly created, and that the balance commitment is fulfilled. If all checks pass, the transaction is included in the blockchain. The new note commitment will be included in the SCT (representing the new owner of the 10 UM). Bob can then scan the blocks (periodically) using his clue key $\mathsf{ck_d^{Bob}}$ to detect new notes that belong to him. Once Bob recognizes that the newly created note is intended for him, he stores the note together with the proof data locally. Bob has now received the 10 UM from Alice, and when he wants to spend them, he does the same procedure as Alice did. This system allows transactions and ownership of notes to be publicly verifiable without disclosing any sensitive information that affects privacy, while most of the data, commitments, and proofs are performed locally, and only the validation and commitments occur on-chain.

3.4.1 Batching, threshold cryptography and flow encryption

In order to provide privacy-preserving transactions, Penumbra must overcome some challenges. Such challenges are computational efficiency, resistance to information leakage, and avoiding centralized trust assumptions. Penumbra does not record transaction values directly on the ledger, instead, it records the ownership through encrypted notes. Those notes encapsulate the values and are only referenced on the chain through commitments. This way the content is protected, but this is, however, not enough to prevent information leakage, which can occur through transaction patterns or aggregate value flows. Penumbra's goal is to provide a high degree of privacy without affecting integrity and traceability. But since Penumbra's ledger state must reflect that value is conserved across transactions, it creates a tension between the need to disclose the aggregated amount of value with each operation (as it is part of the public chain state) and not wanting to reveal the amount in separate transactions. To resolve this tension, Penumbra introduces a mechanism, built from an integer-valued homomorphic encryption scheme, that aggregates value flows (balance commitments) across a batch of transactions, revealing only the total amount and not each individual contribution. Combined with threshold decryption, validators jointly control a decryption key, which ensures that no single validator can access sensitive information independently. This results in the transaction flow being kept encrypted while only the aggregated total amount of value gets decrypted [39].

(Homomorphic) Threshold encryption

Threshold encryption is a technique for securely carrying out cryptographic operations in an environment with multiple participants. It splits a cryptographic key into multiple pieces and distributes those pieces among the participants. The general idea is that one assumes the participating entities can be malicious or susceptible to compromise. Thus, decrypting a shared secret or even signing a message requires the cooperation of a certain amount of participants (not all). Threshold encryption ensures that the participants, often called keyholders, can participate without seeing each other's part of the key. This allows to verify that each key holder has a valid piece of the key without revealing it to others. This distribution of key pieces ensures that not a single individual can decrypt something on their own. Only when enough key pieces are put together (the threshold is reached) is it possible to start decrypting. This greatly reduces the risk of a single point of failure, i.e., enhances the security. The secret remains secure, even if some participants work together. Additionally, since only a threshold is required to be reached, the system ensures that even if one or more individuals are unavailable, the secret can still be collaboratively unlocked, causing no bottlenecks, making the system fault-tolerant.

Going into more details, assume we have a secret S (in Penumbra, for example, an aggregated (batched) value flow). After S is encrypted using a public key, the corresponding private key for decrypting S is split up among a fixed number of participants N using a threshold secret sharing scheme, conceptually related to Shamir's Secret Sharing. The threshold cryptography introduces the variable K representing the minimum number of participants required (the threshold) to decrypt the secret. With $K < N$, the scheme provides fault tolerance, as any set of at least K participants can jointly decrypt S. The values N and K are public system parameters, however, individual participants do not know which other participants hold the remaining key shares. The distribution of the decryption key shares for the secret S is performed through a distributed key generator (DKG) protocol [39] [52]. This results in no single participant ever learning the full private key. Each participant receives exactly one key share in a confidential manner, and in order to decrypt S, at least K participants must cooperate with each other. For example, with $N = 10$ and $K = 7$, even if half of the participants are compromised, the secret S cannot be decrypted, as two additional participants would be required. Hence, the system stays secure as long as fewer than K participants act maliciously [51]. Besides fault tolerance, threshold encryption also helps mitigate Maximal Extractable Value (MEV).

Maximal Extractable Value: MEV is the value block producers (validators) can extract by reordering, front-running, or censoring transactions before they are added to the chain. Common MEV attacks include: *front-running*, where a specific pending transaction is executed first, often resulting in profit for the attacker, and *sandwich attacks*, where an attacker selects a transaction as its prey. An attacker puts a different transaction in front and back of the target, creating a disadvantageous outcome for the target, and a profit for the attacker.

A typical example of the sandwich attack involves impacting the exchange rate of currencies. The target wants to do a large exchange of currency A to B, let us assume an exchange of 10'000 A to B. The exchange rate before the attack is 1:1. An attacker could then place a transaction in front where the attacker buys 100 B for 100 A. This will increase the exchange rate slightly, for example, to 1.05. Now the target has to exchange 10'000 A at an exchange rate of 1:1.05, resulting in approximately 9'523 B (if the transaction had gone through first, it would have been 10'000 B). This large exchange further increases the exchange rate, let us assume to 1.10. The attacker then sells the previously bought 100 B at the new exchange rate of 1.10 and receives 110 A, making a profit of 10 A.

MEV is not inherently bad (for example, it has been used to balance token prices across decentralized exchanges), but in most cases, MEV extraction techniques are used with malicious intent to increase profit from a user's trade without their knowledge [51]. In typical blockchain networks, the validation of transactions and continued production of new blocks containing the validated transactions are guaranteed. However, there is no guarantee as to the order in which transactions are executed/published on-chain. Validators by default order transactions by the highest transaction fee in order to maximize their profits, thus allowing them to extract additional value by reordering transactions (MEV) [11]. In Penumbra, this is not possible, as all transactions are encrypted (they are placed in the shielded pool). MEV attacks require insight into transactions. Together with the threshold encryption, transaction details can only be gained when the block inclusion decision has occurred, i.e., the threshold of K has been reached. Only then can the validators decrypt it, but then the transaction has already been fixed. As a result, any kind of MEV attacks are effectively prevented this way.

Batching

Although Penumbra executes and validates transactions individually, there are many operations in Penumbra (such as validating certain flows or settling fees) that require the aggregated value (but not the individual transaction values). Hence, only aggregated values should be decrypted, never individual transaction values, in order to guarantee the privacy of the users. In such cases, transactions are batched, and those transactions that contribute value to a batch contain an encryption of the amount they contribute. In order to allow batching across block boundaries, the blocks are organized into epochs (as illustrated in figure 3.5). Changes to the validator set are only applied at epoch boundaries. Validators collect the ciphertexts of all relevant transactions for each epoch and then sum them homomorphically. The homomorphic aggregation in Penumbra is provided by an additive homomorphic encryption scheme (ElGamal encryption [23]), which allows validators to add up ciphertexts without learning their contents, i.e., the values of those ciphertexts. Then the validators compute an encrypted total of the batch, which is then jointly decrypted through threshold encryption and committed to the blockchain. This process does not require any additional action by users whose transactions are batched. Only the validators need to create and publish a threshold decryption key [39].

In the first block of each epoch, the encryption key is included, which the validator set derives from a jointly controlled decryption key produced from a distributed key generation (DKG). This ensures threshold cryptography, where no single validator controls or can derive the full decryption key individually. Because this encryption key is only available in the first block of an epoch, some transactions may not be included in this block. In the documentation, they estimate, under the assumption of a block interval similar to the Cosmos Hub, that there is a processing delay of around 8 seconds once per day. Additionally, a decryption key lives for the duration of an epoch, meaning that value flows can be batched between the creation of the first block and the end of an epoch. The documentation proposes an epoch boundary on the order of 1-3 days [39].

Flow encryption

Penumbra plans to extend its privacy features through flow encryption, allowing the use of distributed key generation (DKG) and threshold homomorphic encryption to encrypt and aggregate multi-party transactions, thereby overcoming the limitations of traditional zero-knowledge proofs for such scenarios. This feature, however, depends on the implementation of ABCI 2.0 in the Tendermint consensus protocol, and since this has been delayed, flow encryption is not part of Penumbra V1, as stated in the official Penumbra documentation [39]. With respect to this statement, it is unclear how up to date the documentation is, as they have since transitioned from Tendermint to a fork of it called CometBFT, and, at the time of writing, the current version of Penumbra is 2.1.0. According to the specification and tutorial on how to configure the system, it is stated that one should use CometBFT v0.37.x and not the newer (current) version v0.38 due to incompatibility [52]. CometBFT v0.37.x did not introduce the ABCI 2.0 (also known as ABCI++). This was only introduced in v0.38 and later versions [13] [50], which leads to the assumption that flow encryption is still not part of the current Penumbra protocol. However, given the detailed documentation about flow encryption, as well as the progress of CometBFT in implementing ABCI 2.0, it appears likely that it will not take much longer until Penumbra starts to use flow encryption as originally intended.

3.5 Application-level mechanisms

3.5.1 Staking and delegation

Penumbra is built on a delegated Proof-of-Stake model using the Tendermint/CometBFT consensus with additional modifications to support delegation privacy. In a typical Proof-of-Stake system, users can stake their tokens by delegating them to validators. *Staking* is the process where a user "locks" a certain amount of their assets or tokens in order to support the operations of the blockchain, helping the network to validate transactions and keeping it secure. In return, users earn staking rewards (similar to earning interest). Since most users will not operate their own validator node, staking is performed through *delegation*. A user delegates some of their assets or tokens to a validator. However, the ownership of the delegated value remains with the user. The validator only receives the right to use the delegation. The total amount of delegation (the delegation pool) of a validator (including the validator's own self-bonded stake) represents the voting power in the consensus process, i.e., the influence/chance of being selected to propose or validate blocks, which in return also earns rewards for the validator ("no one does something for free"). This responsibility introduces a risk: if a validator misbehaves (e.g., proposing/signing an incorrect transaction), it results in part of the delegated stake getting *slashed*, which means a portion of the asset or tokens that are bonded to that validator will be destroyed. This leads not only to losses for the validator but also for the delegators, and the validator loses trustworthiness. The delegators are incentivized to choose with caution whom they delegate their stake to. Conversely, correct behavior of the validator will get rewarded by distributing staking rewards proportional to each delegator's share. Misbehavior will also hurt validators in the future, as fewer delegators trust them, resulting in a smaller delegation pool, hence less voting power, thus a smaller chance to "win" the consensus, leading to fewer rewards earnable. This incentivizes validators even more to behave according to the protocol. So, by participating in staking, one helps to validate transactions, secure the network, and increase the cost of misbehavior and entitles stakers to receive rewards. Staking rewards are often calculated as an annual percentage yield (APY). This system overall resembles depositing money at a bank: a user deposits money in a bank they believe to be trustworthy, the bank uses this deposit as capital to earn money (e.g., through investments), and the user receives in return interest.

Now, Proof-of-Stake in combination with privacy introduces several challenges. Usually, delegations are public, i.e., recorded publicly on-chain. The majority of stake is bonded to validators, and privacy becomes uncommon. This makes it easy to distribute staking rewards, as the delegated amount and the duration of each delegation address are known. This is, however, not private. If delegations are private, then giving out staking rewards becomes very difficult as the chain no longer knows the amount and duration of each delegator address.

Penumbra addresses this problem by "completely" eliminating staking rewards and introducing a new mechanism that takes two different assets, unbonded and bonded stake, together with an epoch-varying exchange rate that functions similarly to staking rewards in other systems. All delegations to a particular validator are fungible, represented by a single delegation token mirroring a delegator's share in the validator's delegation pool. This delegation token is like other tokens recorded inside the multi-asset shielded pool of Penumbra, and only the total amount of stake bonded to each validator is part of the public state, determining consensus weight. This allows validators to be held accountable while providing privacy and flexibility for delegators, as they can handle delegation tokens like any other token. The fungibility, which provides privacy, comes with a sacrifice ("privacy is never free"). Delegators cannot realize income during the bonding period, only a capital gain (or loss) upon unbonding [39].

Staking tokens

As previously discussed, there are two types of stakes: bonded and unbonded stakes. Having two states representing the same token is impractical, hence, Penumbra introduces *delegation* tokens. Any regular token represents units of unbonded stake, they can be used freely or delegated. Once the token gets delegated to a validator, the delegated amount does not just get bonded (i.e., locked), instead, it is transformed into a delegation token, representing stake bonded with a particular validator. The following notion, shown for Penumbra's staking token PEN, applies to all other tokens. Penumbra's staking tokens represent unbonded stake, bonded tokens are called dPEN (d for delegation). It is crucial to know that there is no single dPEN asset: only delegations to a specific validator are fungible (represented as a single token that shows the share of ownership of that specific validator's delegation pool). If a stake is bonded with different validators, then the stake is not fungible, as each validator may have different commission rates and different risks of validator misbehavior. Thus, dPEN stands for a class of assets instead of a single-asset. The fungibility of dPEN is important for protection and privacy, as this prevents tracking how much stake individual users have delegated. As dPEN tokens are handled as all other tokens, it makes them flexible, meaning they can be traded, transferred, or even used in IBC. The exchange rate of PEN $\rightarrow$ dPEN (delegation) and dPEN $\rightarrow$ PEN (undelegate) is neither 1:1 nor fixed, it uses an epoch-varying exchange rate [39].

To calculate the exchange rate, first, the validator(v) specific commission rate is calculated. Each validator declares a set of funding streams, which specify both the destinations of the commission and the total commission rate $c_{\mathsf{v},e} \in [0, 1]$ (where v denotes the validator and e the epoch). This total commission rate is then subtracted from the base reward rate r_e to get the validator-specific reward rate:

$$r_{\mathsf{v},e} = (1 - c_{\mathsf{v},e})r_e$$

The base reward rate r_e is a parameter indexed by epoch e, which serves as a reference rate for the entire chain. Its value is set on a per-epoch basis, which involves the ratio of bonded and unbonded stake (increases when there is relatively less bonded stake and decreases when there is relatively more). This parameter is determined and adjusted by the governance [39].

Based on this, the base exchange rate between PEN and dPEN can be calculated:

$$\psi(e) = \prod_{0 \leq i < e} (1 + r_i)$$

where $\psi(e)$ measures the cumulative depreciation of stake PEN relative to the delegation token dPEN from genesis up to epoch e. Now, since the delegation token dPEN is not a single-asset but rather a class of assets (per-validator assets), this is only the base rate [39].

To get the actual exchange rate between the stake PEN and the validator-specific delegation token dPEN(v), the commission has to be taken into account. This is done by substituting the base rate r with the validator-specific rate $r_{v,i}$:

$$\psi_{\mathsf{v}}(e) = \prod_{0 \leq i < e} (1 + r_{\mathsf{v},i})$$

With this mechanism, it is not necessary to track the age of each specific delegation to compute the rewards. This is because discounting newly bonded stake by the cumulative depreciation of unbonded stake since genesis allows all bonded stake to be treated as if it had been bonded since genesis. Newly unbonded stake can always be inflated by the cumulative appreciation since genesis [39].

Through the use of delegation tokens as a separate token, Penumbra can integrate its privacy mechanism into delegation, being part of the shielded pool, making it impossible to publicly link them to addresses. Also, the process of tracking the duration of delegations has become easier, as this is integrated in the exchange rate, which already reflects accumulated rewards since genesis [39].

Delegation

Whenever staking tokens are delegated to a validator, those tokens are exchanged for delegation tokens (i.e., dPEN(v), the validator-specific bonded-stake asset introduced above), which are held by the delegator and can be traded similarly to liquid staking tokens on other blockchains. Every validator maintains a *delegation pool*, which represents the validator's total bonded stake and therefore their voting power in consensus. The total stake in a validator's delegation pool is public, while the individual delegations are private. As an incentive for correct behavior, validators also earn staking rewards. Each validator establishes a commission rate, which defines a percentage that the validator earns from the total staking reward from their entire delegation pool before the remainder is distributed across the delegators. The chain keeps track of the exchange rate between each validator's delegation token and staking token, which reflects the staking rewards earned by that validator. Whenever a validator misbehaves, it is "slashed", which means the value of the validator's delegation tokens is reduced, reducing the exchange rate $\psi_v(e)$, penalizing both the validator and their delegators. A delegator can undelegate their tokens and will receive the originally staked tokens (provided that the validator was not slashed) and their earned rewards (through the exchange rate of dPEN to PEN). Thus, the higher the stake of the delegator and the performance of the validator are, the greater the increase in value.

A delegator cannot, however, immediately undelegate their stake. This is done to prevent the evasion of slashing penalties. A validator could escape the slashing penalty by quickly withdrawing their stake. Penumbra introduces an unbonding delay, during which undelegated stake enters an intermediate *unbonding state* (shown in Figure 3.6). Whenever a delegator undelegates their tokens, they are first converted to

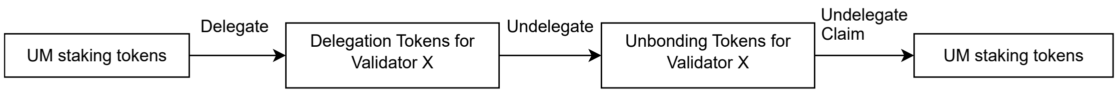

Figure 3.6: State transition of staking tokens

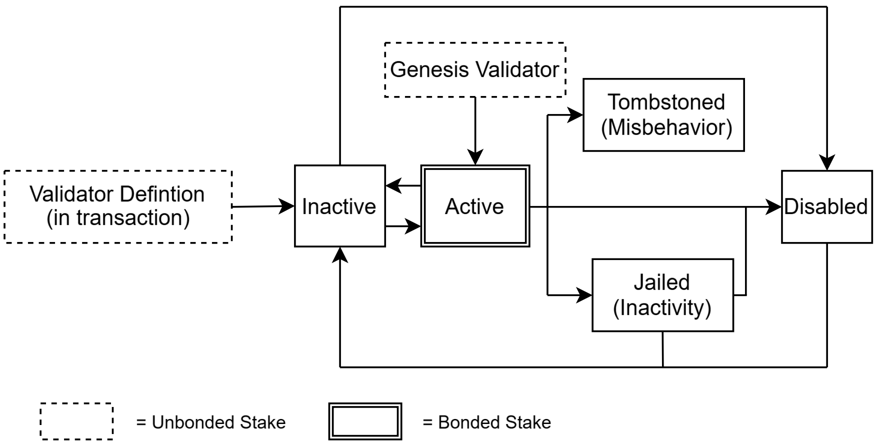

Figure 3.7: States of Validators

temporarily unbonded tokens (an intermediate state between PEN and dPEN), which are still tied to the validator's identity (i.e., the specific validator) in order to remain exposed to slashing penalties in case the validator misbehaves. After the unbonding delay, the unbonding tokens can be converted back to staking tokens, taking into account the staking rewards and the validator's exchange rate (including slashing penalties). This unbonding delay is currently set to 120,960 blocks, which is approximately 7 days [52]. This concept also has the advantage that unbonded tokens, while still tied to the validator, can be transferred or traded. Thus, if detect a risk of slashing, they can sell their tokens at a discount, which results in the slashing risk remaining associated with the unbonded tokens, representing the uncertainty of the corresponding validator (capturing the risk profile of that validator) [52].

Validators

The term validators has been used frequently in the previous sections. One may ask, how can one become a validator? In principle, every participant in the Penumbra protocol can become a validator with the necessary infrastructure to run an active validator node setup (being able to run and maintain software and infrastructure). If this is the case, one initially becomes an inactive validator. Only the top 100 validators by total stake in their delegation pools at any epoch can participate in the consensus and earn rewards, forming the *active* set of validators [52]. The other validators remain inactive, until they enter the top 100. In order to become an active validator, a validator must contribute a certain amount of self-stake to the delegation pool in order to demonstrate self-commitment, as the validator also bears risk in case of misbehavior. Validators are known to the chain either at genesis or by issuing a transaction with a *ValidatorDefinition* action in it. Validators specified at genesis begin in the active state with their allocated stake in their delegation pool at genesis. Other validators created later start in the inactive state, with no stake in their delegation pool. It is possible to stake to/delegate to an inactive validator's delegation pool, but this will earn no rewards. In addition, there is no unbonding delay in effect, i.e., the stake is not bonded, so undelegation is effective immediately. Validators can transition through five states (some of which have already been mentioned), also shown in figure 3.7 [39]:

- **Inactive**: A (newly created) validator whose delegation pool is too small to participate in the consensus set. Every stake contributed to it will not earn rewards and there is no unbonding state.
- **Active**: When a validator has a nonzero balance and its voting power (total stake in the delegation pool) is in the top 100 (the *active* set of validators [52]), then an inactive validator becomes active during the next epoch transition. An active validator participates in the consensus. The stake delegated to an active validator can earn staking rewards, but this also implies that the unbonding delay is active. An active validator can exit the consensus set in four ways. One way is that it gets displaced by another validator with more stake in its delegation pool. This transitions the validator back to the inactive state, and the stake in its delegation pool stays in the unbonding state (at the time the validator becomes inactive). After that the validator is inactive, and it can become active again. The remaining transitions are described below.
- **Jailed**: If for some reason an active validator behaves incorrectly due to inactivity, it transitions into the *jailed* state and gets slashed for inactivity. Jailed validators are immediately removed from the consensus set. The validator's rates are updated to reflect the slashing penalty and are then held constant. Jailed validators are not permanently banned from participating in consensus, as inactivity cannot be seen as something done on purpose (for example, due to network or technical issues). However, if this happens too often, then the state can change to the next explained state, *tombstoned* (as this could be a sign of malicious intent or the validator is not suited to be a validator anymore). An operator of a jailed validator can re-activate it by re-uploading the validator definition. As long as the validator is jailed, no stake can be delegated to a slashed validator. The stake in the delegation pool that has been contributed before enters the unbonding state in order to hold the validator accountable for any Byzantine behavior during the unbonding period. Re-delegation can occur after the validator enters the inactive state.
- **Tombstoned**: An active validator can become *tombstoned* and respectively slashed for Byzantine misbehavior. This transition is of high importance, therefore, it can occur in any block, triggering an unscheduled epoch transition, where the validator is immediately removed from the consensus set and banned from participating in consensus permanently. Any open undelegations from the slashed validator are terminated: the quarantined output notes are deleted, and the quarantined nullifiers are taken out of the nullifier set. The validator's rates are adjusted to reflect the slashing penalty and are subsequently maintained unchanged. Stake can no longer be delegated to a tombstoned validator. Any stake that has been delegated to the tombstoned validator's delegation pool is not bonded (as the validator has already been slashed and tombstoned), which means undelegations can occur immediately, i.e., there is no unbonding delay.
- **Disabled**: Finally, an active validator can enter the *disabled* state if the operator of the validator manually disables it. The validator then no longer participates in the consensus and does not earn staking rewards (the validator's rates are held constant). At the time the validator transitions to the disabled state, the stake in the delegation pool will enter the unbonding state. After switching to the disabled state, the validator can only transition back to the inactive state if the operator re-activates it by updating the validator definition.

Staking dynamics

The Penumbra documentation [39] provides a illustrative example that shows the dynamics of validators and delegators in the system. The scenario is as follows: there are three delegators, Alice (a), Bob (b), and Charlie (c), and two validators, Victoria (v) and William (w). The consensus requires at least four validators and no single party controls more than $1/3$ of the stake (otherwise, the BFT consensus security guarantee fails, a party with control over $1/3$ of the stake has a voting-power over $1/3$ and could prevent the block finalization, blocking the consensus). The example only uses a few parties to illustrate the dynamics. For simplicity, the base reward rates and commission rates are fixed over all epochs at $r = 0.0006$ and $c_{\mathsf{v}} = 0$, $c_{\mathsf{w}} = 0.1$. The PEN and dPEN holdings of participants a, b, and c are denoted by x_i and y_i respectively (where i refers to a, b, or c). In the beginning, Alice starts with $y_{\mathsf{a}} = 10\,000$ dPEN of Victoria's delegation pool, Bob starts with $y_{\mathsf{b}} = 10\,000$ dPEN of William's delegation pool, and Charlie starts with $x_{\mathsf{c}} = 20\,000$ unbonded PEN.

- At genesis, the distribution of total stake between the participants Alice, Bob, and Charlie is $25\%, 25\%$, and 50% respectively. The total voting power is distributed between them as $50\%, 50\%$, and 0% respectively (Charlie holds no delegation tokens).
- At epoch $e = 1$, the holdings of Alice, Bob and Charlie do not change, but their unrealized notional values have changed.
  - As Victoria charges no commission ($c_{\mathsf{v}} = 0$), we get that $\psi_{\mathsf{v}}(1) = \psi(1) = 1.0006$. Thus, Alice's $y_{\mathsf{a}} = 10\,000$ dPEN(v) is now worth $10\,006$ PEN.
  - William charges 10% commission ($c_{\mathsf{w}} = 0.1$), so $\psi_{\mathsf{w}}(1) = 1.00054$. Bob's $y_{\mathsf{b}} = 10\,000$ dPEN(w) is now worth $10\,005.4$ PEN, and William receives 0.6 PEN.
  - The commission William receives can be used to cover expenses, or self-delegate. In the example, it is assumed that the validators self-delegate their entire commission, illustrating staking dynamics. William self-delegates 0.6 PEN, to get $0.6/\psi_{\mathsf{w}}(2) = 0.6/1.00054^2 = 0.59935...$ dPEN in the next epoch.
- At epoch $e = 90$:
  - Alice's $y_{\mathsf{a}} = 10\,000$ dPEN(v) is now worth $10\,554.67$ PEN and Bob's $y_{\mathsf{b}} = 10\,000$ dPEN(w) is now worth $10\,497.86$ PEN. The self-delegations of the commissions that William accumulated over the epoch has resulted in $y_{\mathsf{w}} = 53.483$ dPEN(w).
  - Victoria's delegation pool size remains the same at $10\,000$ dPEN(v). William's delegation pool size has increased to $10\,053.483$ dPEN(w). Their respective adjustment factor (which will be introduced in section 3.5.2, a value used to calculate the actual total voting power of a validator) has now reached $\theta_{\mathsf{v}}(90) = 1$ and $\theta_{\mathsf{w}}(90) = 0.99462$, this means the voting power of their delegations is respectively $10\,000$ and $9\,999.37$. The minor loss of voting power by William's delegation pool occurs because he self-delegates rewards with a one epoch delay, resulting in the missing of a compound interest for one epoch.
  - Charlie's unbonded $x_{\mathsf{c}} = 20\,000$ PEN remains unchanged, but the value relative to Alice and Bob's bonded stake has declined. Additionally, William's commission from Bob slightly reduces Bob's voting power relative to Alice's.
  - The distribution of stake between Alice, Bob, Charlie and William is now at $25.67\%, 25.54\%, 48.65\%, 0.14\%$ respectively. The distribution of voting power is $50\%, 49.74\%, 0\%, 0.27\%$ respectively.
  - Charlie now decides to bond his stake evenly between Victoria and William. This gives him $10\,000/\psi_{\mathsf{v}}(91) = 9\,485.85$ dPEN(v) and $10\,000/\psi_{\mathsf{w}}(91) = 9\,536$ dPEN(w).

- At epoch $e = 91$:
  - Charlie has now 9 468.80 dPEN(v) and 9 520.60 dPEN(w), which is worth 20 000 PEN. For the same amount of unbonded stake, Charlie gets more dPEN(w) than dPEN(v), because the exchange rate ψ_{w} prices in the cumulative effect of commission since genesis. Charlie is not charged for commission during the time he did not delegate to William.
  - The commission for William in this epoch is now 1.233 PEN, up from 0.633 PEN in the previous epoch.
  - The distribution of the stake between Alice, Bob, Charlie, and William is now 25.68%, 25.54%, 48.64%, 0.14% respectively. Since all stake is now bonded (except William's commission for this epoch but that is insignificant in size), the distribution of voting power is identical to the distribution of stake.

This example could be continued for many more epochs, but this is enough to illustrate the dynamics of staking.

3.5.2 (Decentralized) Governance

A major advantage of delegation in Penumbra is the link to on-chain governance. It uses a delegated Proof-of-Stake model, which connects staking and delegation mechanisms to governance and enables privacy with delegated voting. *Governance* is the term used in blockchain systems to describe the process by which the network collectively makes decisions. Since in blockchains, including Penumbra, there is no central authority, it means that changes to the system, such as protocol, parameter, or infrastructure changes, are driven through a formal voting procedure. Participants submit their votes in encrypted form, while the total result is publicly aggregated. This is possible due to threshold encryption and the aggregation of votes by validators at the end of every epoch. The blockchain can then publish the results without revealing individual votes. This principle enables transparent governance results without compromising the privacy of individual participants. The governance system is similar to the one used in Cosmos Hub, with the difference that Penumbra uses only three voting options: *yes*, *no*, and *abstain*. In the Cosmos Hub there is also a vote called *NoWithVeto*. This is not included in Penumbra, but its effect is partially implemented via a threshold (80% by default) for *no* votes, i.e., if a sufficient number of *no* votes is reached, the proposal is slashed, and the deposit is burned. This is a prevention mechanism for spam proposals [1] [39] [52].

Proposals

Similar to staking, everyone can submit a governance proposal by depositing the required amount of tokens (the current required proposal deposit is 10 PEN). A user creates a transaction with a *ProposalSubmit* description, depositing an amount of PEN that is escrowed (this deposit is used to demonstrate commitment and assume risk). Proposals are handled like normal transactions, thus, they remain private. This ensures that the user's address remains hidden and unlinkable, only the proposal itself is publicly visible. The control over the escrowed proposal deposit is mediated by a per-proposal family of bearer NFTs. An NFT (*Non-Fungible Token*) is a unique digital token, representing a specific asset or right on a blockchain, and unlike regular tokens, it cannot be exchanged for another. In Penumbra, an NFT encodes the right to reclaim a proposal deposit or receive a commemorative token, depending on the outcome. For creating a proposal N, the user receives a *proposal_N_Voting* NFT. This can be redeemed for the original proposal deposit (if not slashed) and a commemorative *proposal_N_passed*, *proposal_N_failed*, or *proposal_N_slashed* NFT. A proposal can either pass or fail, but regardless of their outcome (as long as it did not get slashed), the user can reclaim their deposit. A proposal can also be withdrawn prior to the end of the voting period by creating a transaction with a *ProposalWithdraw* description. Withdrawn proposals cannot be accepted,

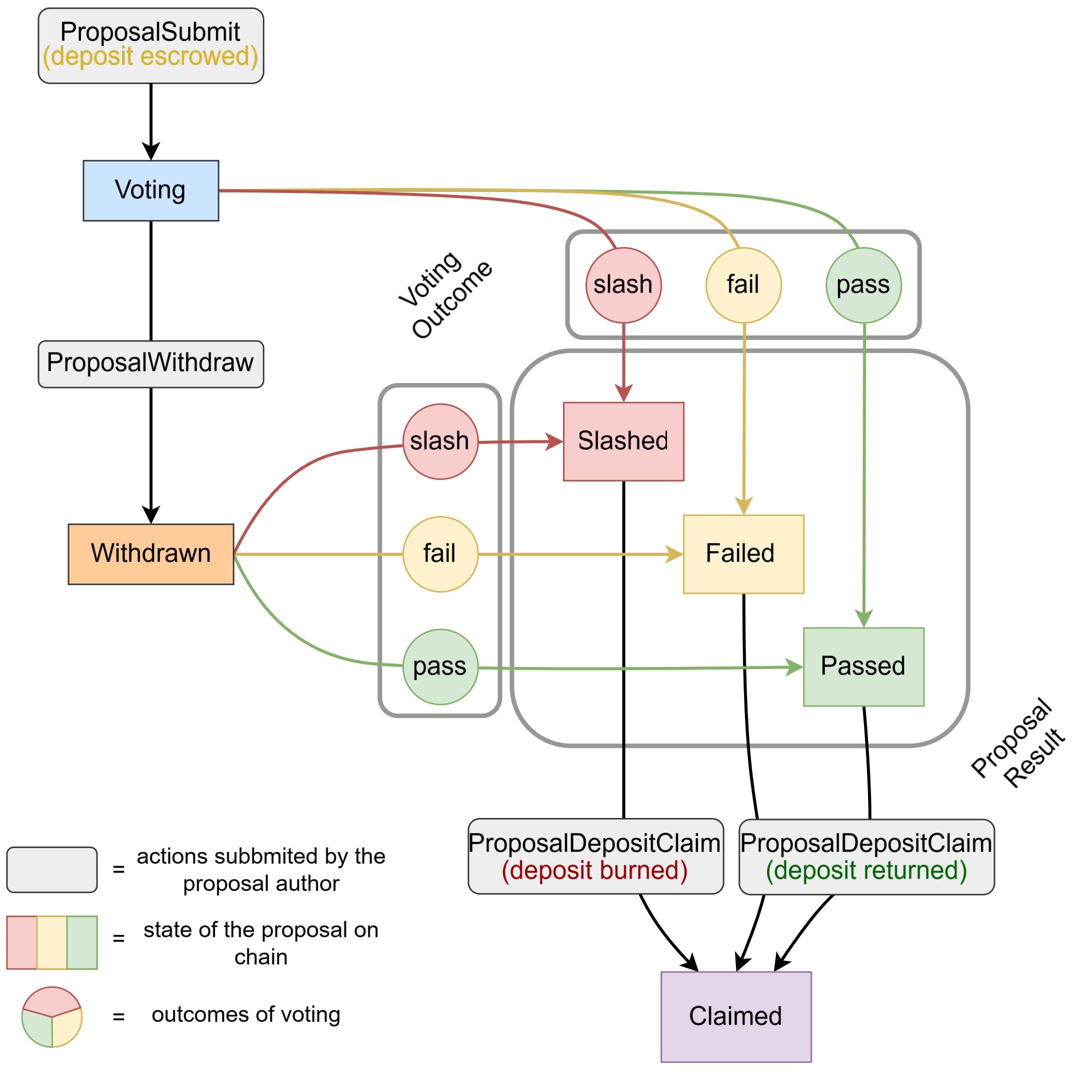

Figure 3.8: Proposal life cycle

even if the vote would have passed. However, they may still be slashed (i.e., it is not possible to create "fake" proposals and then immediately withdraw to avoid being penalized). The deposit, however, cannot be returned once the number of *no* votes exceeds the proposal slash threshold. From the proposer's point of view, the lifecycle of a proposal begins when it is submitted and ends when the deposit is claimed. Depending on the action, the NFT the proposer receives is named differently. For proposal N, the NFT of denomination *proposal_N_submitted* is returned to the proposer. If the proposer decides to withdraw (in time), the *proposal_N_submitted* NFT is consumed, and a *proposal_N_withdrawn* NFT is returned. When the proposer wants to claim the escrowed deposit, the *proposal_N_submitted* NFT is consumed and the *proposal_N_claimed* NFT is returned along with the original proposal deposit, provided the proposal has not been slashed. This is the final state of the proposal lifecycle. The life cycle of proposals can be seen in figure 3.8 [39] [52].

There are several kinds of proposals, and a proposal can either be a normal one or an emergency proposal. In both cases, the voting period begins immediately in the next block after the proposal is committed to the chain. The difference between normal and emergency is essentially how long it takes for a proposal to be accepted. A normal proposal has a fixed-length voting period, while an emergency proposal has a shorter voting period, as it only requires a 1/3 majority of the stake to be reached in order to be accepted. To anticipate a bit, validators provide default votes for their delegation pool, hence, an emergency proposal could be accepted immediately, without any input from delegators. This way it is possible to solve a time-critical emergency, for example, deploying a 0-day hotfix, as any stake-weight minority holding strictly more than one-third of the total active stake is enough to halt the chain [39] [52]. The following is a list of possible proposal types in Penumbra [39] [52]:

- **Signaling**: A signaling proposal is a proposal that does not have a mechanized effect on the chain when passed, it is used to *signal* community consensus on a topic (e.g., to agree on a code change).
- **Upgrade**: The upgrade proposal is a special kind of signaling proposal. It coordinates a chain halt, where full node operators (anyone who runs the Penumbra node software, i.e., listing and indexing nodes and not just validators) can choose if they want to apply an *upgrade* to a new software version.
- **Emergency**: As already mentioned previously, emergency proposals are ones that require *immediate* action (to prevent a potential crisis). They should conclude earlier than a normal proposal, as soon as a 1/3 majority of all active voting power votes *yes*. The emergency proposal can also coordinate a chain halt. If this occurs, then off-chain coordination between validators is required to restart the chain. It is also important to know that an emergency proposal has limited power, which is that it cannot actually do anything. This replicates the ability of a strictly greater than $1/3$ stake-weight minority to halt the chain.
- **Parameter**: A parameter proposal changes the chain parameters when it passes. Chain parameters specify things like staking reward rate, slashing penalties, and other properties/numbers that determine how the chain should behave. A parameter change proposal contains the (at the time of proposing) current parameters and the new proposed ones. Only if the current parameters in the proposal match the actual parameters, and if the proposal is accepted, then the parameters are changed immediately, otherwise, nothing happens. This is done to prevent two simultaneous parameter change proposals from overwriting each other or merging together into an undesired state.
- **(Un)Freeze IBC Client**: This proposal allows freezing an IBC client (or unfreezing) for a counterparty chain. This is, for example, used when the community collectively agrees that a counterparty chain might be compromised (or not anymore in the unfreeze proposal).
- **Community Pool**: This proposal allows the community to submit a transaction plan that may spend funds from the community pool if passed. The community pool spend transaction has two exclusive actions, which are limited to the community pool only: *CommunityPoolSpend* and *CommunityPoolOutput*. Those actions spend funds from the community pool and mint funds

transparently to an output address (unlike regular output actions, which are shielded). Those spend transactions are not able to use the regular shielded outputs, cannot spend funds from any other source except from the community pool, and cannot perform swaps, submit, withdraw, or claim governance proposals. This proposal is mainly used for on-chain funding. The community pool acts as a funding mechanism (similar to how Cosmos-based chains use community pools to fund, for example, ecosystem growth or incentives). Only spending proposals are accepted that do not overdraw the current funds in the community pool at the time of the submitted proposal. However, spending funds from the community pool is currently disabled to minimize protocol risk and will be activated in the future once it is in a sensible and secure state.

Regarding the community pool, anyone can contribute any amount to it (instead of declaring a penumbra address as a recipient, the community pool is specified). However, this is not a delegation or deposit, meaning it cannot be withdrawn except through a successful community pool spend governance proposal [39].

Voting

The voting process of proposals functions similarly to transaction validation through (active) validators and their delegators. Voting can therefore be separated into validator voting and delegator voting.

The validator votes are public, voting is a transparent action and attributable. The votes from validators act as default votes for their delegators. The delegator votes, on the other hand, are anonymous and override their portion of the validator's vote. They must have delegated some tokens to the (active) validator when the proposal they want to vote for was submitted. Thus, a delegator vote consists of a spend proof for a given delegation note to some validator combined with an inclusion proof that shows that the delegation was included in the delegation pool of the validator before voting started on the proposal they wish to vote for. Similarly to spending a note, a delegator could delegate the same note to cast double votes on the same proposal. This double-voting problem is addressed similarly to the double-spending problem. The delegator vote reveals the nullifier for the note that was used to justify the vote (but not revealing any information about the note itself). The delegator votes (also sometimes referred to as stakeholder votes in the documentation) are of the following form:

$$(x_y, x_n, x_a)$$

This represents the weights for *yes*, *no* and *abstain*. The delegator will set the value they want to vote with to 1 and the rest to 0 (for example, voting with *no* would look like (0,1,0)). In order for their vote to count, they also must prove ownership of some amount of bonded stake (or delegation equivalently) prior to the beginning of the voting period. This is done through a transaction with a vote description, which identifies the validator v and the proposal, proves spend authority over a note recording y dPEN(v) (the delegation token delegated to validator v), and reveals the note's nullifier. At the end, the vote consistency $y = x_y + x_n + x_a$ is proven to produce a new note with y dPEN(v) and includes $\mathsf{Enc}_D(x_i)$, an encryption of the vote weights to the validators decryption key [39].

The encrypted votes from the delegation pool of each validator will get collected at the end of each epoch. Those encrypted votes are aggregated into encrypted tallies, which then are decrypted to only reveal the intermediate aggregated result. Those intermediate tallies are revealed because it is not possible to batch value flows over time intervals longer than one epoch. However, this is not a privacy risk as only the tallies are public, not the votes themselves, preserving individual vote privacy. At the end of the voting period, those per-epoch tallies are summed up. For every validator v, the votes for each choice are added up in order to calculate the share of the delegation pool that participated. The validator's vote serves as the default vote for the remainder of the delegation pool. The validator's voting power is equal to the total delegated voting power at the time a proposal started voting. A delegator's voting power is equal to the unbonded staking token value of the delegation tokens they had staked to an active validator at the time the proposal started voting. The delegator's voting power is initially attributed to the validator, but as soon as a delegator votes, their voting power is subtracted from the voting power of the validator to whom they had staked delegation notes at the time of the proposal start, and their stake-weighted vote is added to the total of the votes. At the end, the subtotals for each validator are multiplied by the voting power adjustment function $\theta_{\mathsf{v}}(e)$ to obtain the final vote total (since the size of each validator's delegation pool cannot be used directly because they are based on validator-specific conversion rates ψ_{v} and are therefore making direct comparisons impossible) [39].

The adjustment function $\theta_{\mathsf{v}}(e)$ is calculated using the base exchange rate $\psi(e)$ and validator specific exchange rate $\psi_{\mathsf{v}}(e)$ as well as the (validator-specific) reward rate r_e, respectively $r_{\mathsf{v},e}$ (all of them have been already defined in subsection 3.5.1):

$$\theta_{\mathsf{v}}(e) = \frac{\psi_{\mathsf{v}}(e)}{\psi(e)} = \prod_{0 \leq i < e} \left(\frac{1 + r_{\mathsf{v},i}}{1 + r_i} \right)$$

This adjustment function accounts for the compounded effect of the validator's commission on the size of the delegation pool. The validator v whose delegation pool has y_{v} dPEN in epoch e has voting power $y_{\mathsf{v}}\theta_{\mathsf{v}}(e)$.

The proof statements of a *vote* are almost identical to a *spend* proof, but with two key differences. The first difference is that in a spend proof, the note's nullifier is checked against the global nullifier set. However, in the vote proof, the nullifier is only checked against a snapshot of the nullifier set at the time the corresponding voting has begun (making sure the note was not already spent), as well as against a per-proposal nullifier set (to ensure the note has not already been used for voting). This marks a note as spent only within the context of voting on a specific proposal. It does not mark the note as spent in general. The second difference lies in the zero-knowledge proof, which now additionally proves that the note was created before the proposal started and that the note has not been spent after the proposal has started [39]. With those two differences, delegators can vote on multiple proposals concurrently, at the cost of linkability. This means the same note can be used to vote on multiple proposals, but the votes, as well as the subsequent spend of the note, will have the same nullifier and thus will be linkable to each other. However, because the descriptions of the votes are shielded (encrypted), an observer will only learn that two opaque votes were related to each other [39].

3.6 Penumbra decentralized exchange (DEX)

Penumbras decentralized exchange (DEX) is a state-of-the-art DEX engine that enables private and high-performance trading of any IBC asset. It allows users to *swap* between different assets without ever needing to leave the shielded pool of Penumbra. A central element of Penumbra's private DEX is ZSwap. ZSwap is the privacy-preserving decentralized exchange mechanism that enables users to perform asset swaps inside the shielded pool of Penumbra without disclosing information such as trade amounts, asset pairs, or trading strategies. The DEX in Penumbra features a batched swap/intent system and enables fine-grained control over liquidity positions [52]. ZSwap allows to trade anonymously inside the shielded pool. It uses concentrated liquidity pools and processes transactions in batches to reduce maximal extractable value (MEV). Processing transactions in batches means grouping multiple transactions together and executing them simultaneously in a single block (as described in section 3.4.1 about batching). The combination of batch swaps and concentrated liquidity pools increases efficiency and also enhances privacy [1] [39].

In the following paragraphs, batch execution (with regard to swaps) and concentrated liquidity positions will be looked at in more detail.

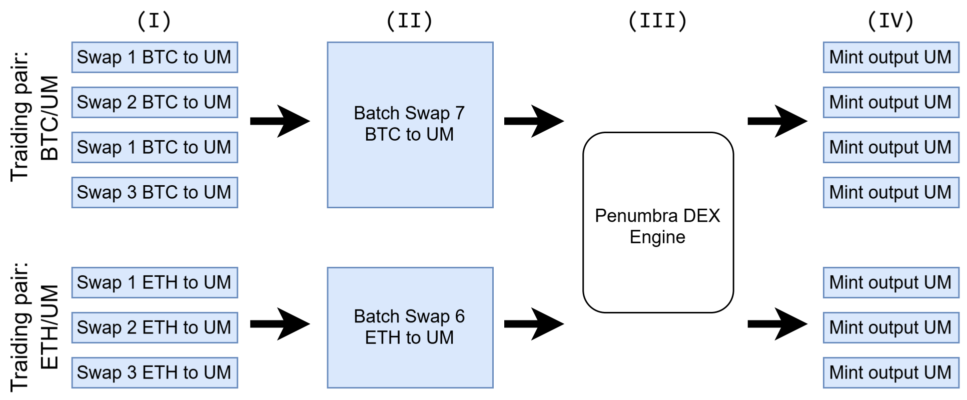

Figure 3.9: Batch Execution

Batch execution

Although batching was introduced earlier in a general context, it is revisited here to explain its application to swaps in more detail. A swap, which is essentially an exchange between two currencies, is triggered by users, and multiple swaps are batched together with all other swaps for the same trading pair (for example, all swaps for BTC $\rightarrow$ UM are batched together). Those batches are then executed at the end of each block. Each swap (transaction) order is encrypted client-side and then included in the blocks in encrypted form. Validators can verify the correctness of the swaps through zero-knowledge proofs but do not learn anything about the plaintext of the swap. Through parts of the balance commitment, it is possible to aggregate all swaps into a batch. The encrypted swaps, or equivalently the complete batch, are encrypted using threshold key encryption. A single validator is never in possession of a complete private decryption key, instead, the decryption key is split into confidential shares via a distributed key generation (DKG) protocol at the start of each epoch. These shares are distributed to each validator in the participating set. To decrypt a batch, a certain threshold of validators must cooperate and combine their key shares. Only then is it possible for the validators to learn the aggregated sum of the batch, ensuring that no single validator can decrypt a batch independently.

Additionally, reordering swaps is simply not possible, as the validator does not know the individual swaps in the first place. The order of swaps within the batch is determined and committed before decryption, hence, when the batch is decrypted collectively, all validators see the same batch in the same order. Therefore, the batch and its swaps are finalized atomically, thus, no validator can act alone (or below a threshold) and reorder swaps within the batch. Hence the validator cannot profit by reordering swaps (i.e., no intra-block ordering), as the batches only represent the overall aggregated amount of swaps (in order to preserve the privacy of individual swaps). This ensures that no validator can extract any MEV (discussed in section 3.4). A trade is executed approximately every 5 seconds, which is Penumbra's block time [52]. Overall, this shows that the batching mechanism, combined with threshold encryption and DKG, is not only very useful for privacy reasons but also for cryptographic fairness, preventing any information leakage and the possibility of exploiting reordering (i.e., extracting MEV) that could be exploited by adversarial validators.

Figure 3.9 shows how swaps are handled and batched together (inside a single block) and further processed through the DEX engine. The steps in the figure are labeled with Roman numerals, referenced in the following text to indicate the corresponding action.

Each block receives swaps (*Swap* transactions) that have been issued by users (I). Those issued swaps, which are encrypted client-side, contain information such as the input asset type, the desired output asset type, and the amount of the input asset to be exchanged (swapped). In a *Swap* transaction, a user publicly burns their input assets and privately mints the output assets [39]. The individual swaps are batched together (summed up) for each trading pair, resulting in a single batch for such a specific trading pair (II). Those batched swaps are encrypted using a threshold key encryption scheme provided by the DKG. Only when a certain threshold of validators approves the block are they able to decrypt the batch(es), seeing only the aggregated sum (and not the individual swaps). For example, if Alice, Bob, and Charlie want to swap 1, 2, and 3 ETH respectively, into UM, then the batch swap consists of 6 ETH to swap into UM. This greatly shows that it does not even matter which swap came in first or how they are going to be ordered, as the sum remains the same and the block, or rather the aggregated batches, must have already been accepted by all validators in order to be able to decrypt them (thus entirely preventing the possibility to do MEV). After batching, validation, and encrypting, the batch swaps are executed through Penumbra's DEX engine (more on that later) per trading pair (III). The batch results are written to the public chain state. A batch swap can either pass or fail (for example, due to insufficient liquidity). The difference lies in the output. If successful, the output would be the swapped asset, in case of failure, it would be the original input. A user can, in subsequent blocks, mint the output funds of the batch swaps directly into the shielded pool using a *SwapClaim* transaction (IV). The minted output funds for a user are their pro-rata contribution to the batch swap (if the swap was successful) or the original input of the swap (in case of failure). In the previous example, Charlie initiated a swap of 3 ETH into UM, thus contributing half of the batch swap input funds. This means if the batch swap is successful, then the *SwapClaim* transaction will mint half of the batch output funds to Charlie. In case of failure, the *SwapClaim* transaction will mint 3 ETH to Charlie. Notably, the *SwapClaim* transactions are automatically issued by wallets. The fees are prepaid in the original *Swap* transaction, and the *SwapClaim* is pre-authorized by the *Swap* transaction, hence does not require any additional signatures from the user.

Concentrated liquidity positions

The liquidity for trades, i.e., swaps, is provided by market makers, who are users providing liquidity by agreeing to buy and sell specific trading pairs at a price they specify. This is done in Penumbra through the creation of liquidity positions. Moreover, Penumbra's DEX supports concentrated liquidity positions. Essentially, each liquidity position is implemented as a constant-sum Automated Market Maker (AMM). Fees are not set via predefined tiers, instead, they are determined on a per-position basis, allowing market forces to set fees dynamically. Active market makers can adjust their prices as often as once per block. The positions of those liquidity positions are public. However, since they are funded from and withdrawn back to Penumbra's shielded pool, the owners remain anonymous. This allows to see the aggregated state of the market without seeing each individual's position and who is in control of it [52].

DEX execution

Now, as there are multiple market makers involved in swaps, this means that there can be different trading paths, some of which are better than others. Penumbra's DEX automatically finds the optimal trading path across all available liquidity. The Penumbra guide [52] explained it in an intuitive analogy:

Each liquidity position represents a road connecting two assets at a specified price. The DEX engine acts as a GPS, that finds the best route for trading the input assets to the desired output asset. A route can be direct, or it can involve multiple liquidity positions in order to get the best price. This process is fully transparent to the user.

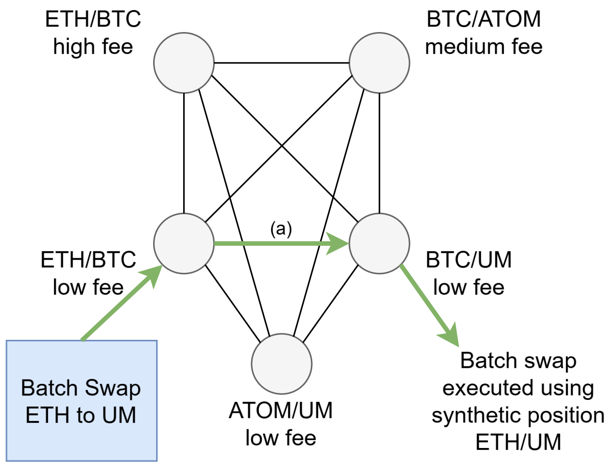

Figure 3.10: Penumbra DEX Engine: (a) Optimal arbitrage performed to ensure prices are consistent

A visualization of the Penumbra DEX engine can be seen in figure 3.10. All of those liquidity positions can be represented as a weighted graph (like a road network) where Penumbra's DEX engine finds the best path for executing trades. Whenever a block arrives at the DEX engine, it finds the optimal path across the graph for each batched trade for each trading pair. Those trades can also be split across multiple pools to provide the best prices for swaps. To find the optimal path, however, is not as simple as just finding the shortest path, for example with the Dijkstra algorithm, since the edges (the exchange prices) are not linear and may change. Additionally, as mentioned, splitting may be beneficial. Penumbra's DEX finds the optimal path by solving an optimization problem for maximizing the output amount for a given input under consideration of the account pool liquidity and exchange fees. The DEX engine also performs optimal arbitrage to ensure that the prices are consistent between liquidity pools. This is done, for example, by systematically swapping UM for various assets and then converting them back to UM to profit from those discrepancies. The generated profit is burned during this process, eliminating the price inconsistency, resulting in constant prices across all pools [52].

3.7 Consensus layer (Tendermint/CometBFT)

The Penumbra protocol uses as its consensus layer the Tendermint consensus [37], more specifically a fork of it called CometBFT [7] [13]. CometBFT is software used to replicate states securely and consistently across multiple machines. It operates under a partially synchronous network model, where under this assumption *agreement* is guaranteed independently of network synchrony, as long as less than $1/3$ of machines fail in arbitrary ways and less than $1/3$ of machines are offline, and *termination* follows once the network behaves synchronously. Consistently means that every non-faulty machine sees the same transaction log and computes the same state. Those properties for replication play a critical role in the fault tolerance of a broad range of applications, from currencies to elections to infrastructure orchestration and beyond. Even though the consensus participation is only limited to validators, Penumbra is still a permissionless system based on staking rather than centralized control. This means the validator membership is open for everyone, it does not require centralized approval, i.e., as long as a validator (suitable node) is willing to stake native tokens, they will get included in the validator set. CometBFT consists of two technical components: the blockchain engine, based on the Tendermint consensus algorithm [37] and a generic application interface, called the Application BlockChain Interface (ABCI), which delivers the transactions to applications for processing. CometBFT is designed to be easy to use, simple to understand, highly performant, and useful for a wide variety of distributed applications [13]. The fast finality and deterministic block production of Tendermint, or rather CometBFT, are well-suited for Penumbra, since it enables reliable batch decryption and execution of shielded transactions all while minimizing information leakages through reordering.

Goals and core features

The design goal of CometBFT makes it a perfect fit for permissionless Proof-of-Stake systems. It provides deterministic and immediate *finality* under *byzantine fault tolerance*. Additionally, it provides *modularity* through its ABCI. In CometBFT, a block is finalized once validators representing $\geq 2/3$ of the total voting power have signed a precommit (more on precommits in the next paragraph). This finality of a block is deterministic and non-probabilistic, i.e., absolute, and cannot be reversed. Thus, applications can fully rely on this finality without having to bother about any possible changes. Compared to the probabilistic finality in Proof-of-Work, CometBFT does not rely on statistical guarantees. However, it still depends on timely message delivery under partial synchrony. In applications with complex logic, such as aggregation, CometBFT is very beneficial as it provides consistency. This consistency is particularly useful in Penumbra, as it ensures that all nodes operate on the same state for zk-proofs. CometBFT also provides Byzantine fault tolerance (BFT), which guarantees as long as less than one-third of the involved processes, i.e., in Penumbra validators, are Byzantine (acting malicious or error-prone). The robustness of this consistent system against Byzantine behavior makes CometBFT a favorable consensus protocol in public networks, where participant behavior according to protocol cannot always be guaranteed. Lastly, CometBFT provides modularity through its Application Blockchain Interface (ABCI). This conceptually separates the consensus layer and the application layer. The application logic, such as account balances, privacy mechanisms, or transactions, is handled in the application layer, which is linked to the ABCI, while the consensus layer is strictly responsible for the Tendermint consensus, guaranteeing finality. Whenever a block is finalized, the application layer is called (through ABCI) to deterministically execute the block and create a persistent state. This guarantees that all nodes will work with the same state, disregarding the programming language or concrete implementation of an application. This provides stability for the consensus protocol and allows modifying or extending the application without the need to adjust the consensus mechanism [13] [37]. All these properties make CometBFT a very well-suited consensus protocol for blockchains that have special requirements, like, for example, data protection or complex transaction logic, as it is present in Penumbra.

Sequence of a consensus round

CometBFT operates in consecutive block heights, and at each height, in rounds. For a given height, validators repeatedly execute rounds until a block is finalized. At the beginning of every round, a *proposer* is deterministically chosen, based on the validator set and their respective voting power, who proposes a new block. The consensus process is roughly divided into *proposal*, *prevote*, and *precommit* (illustrated in figure 3.11 [33]). In the *proposal* step, the proposer creates a block consisting of transactions from the pool (shielded pool in Penumbra's case) and broadcasts a proposal message to all validators (containing all parts of the block). In the *prevote* step, after the validators have checked the proposal for (syntactic/semantic) correctness and validity, they send a prevote for that block, or if invalid or not received in time, they prevote *nil*. As soon as more than two-thirds of all validators have prevoted, a "polka" is formed (which is a supermajority prevote). This threshold of $2/3$ for the supermajority follows from the Byzantine fault tolerance assumption, where less than $1/3$ of validators may behave maliciously. Now we are in the *precommit* phase. If the block contains such a "polka", the validators proceed to broadcast their precommits (effectively locking the block that has reached a polka for that validator). Only once the proposed block

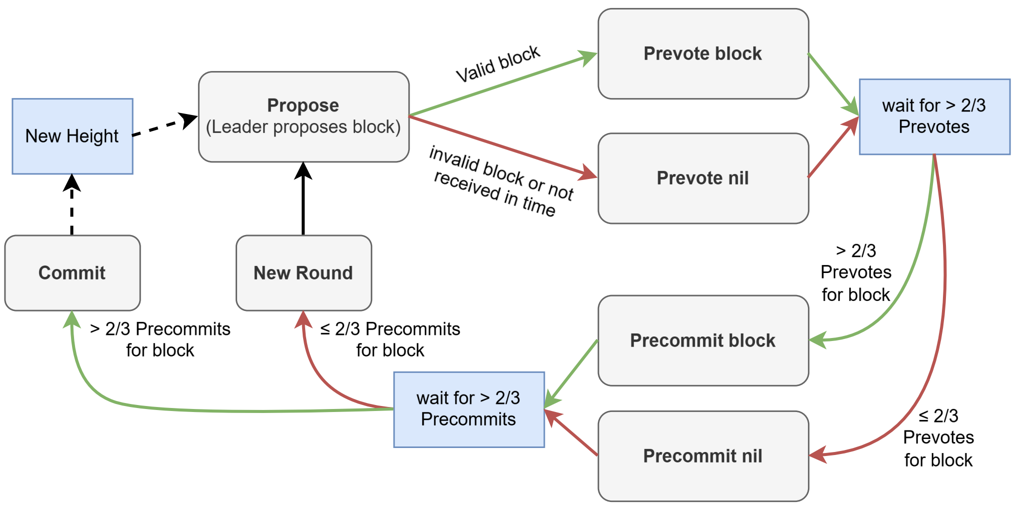

Figure 3.11: Sequence of a Tendermint/CometBFT consensus round

reaches at least two-thirds of precommit votes will it be accepted as final and committed by all nodes (validators), i.e., executed according to the content (transactions) of the block to their local state machines. Whenever no such agreement is reached, i.e., the threshold is not met, the procedure continues into a new round, where a new proposer is selected and the whole process repeats itself until a block has been finalized. The way the process is structured reduces the communication complexity, as the protocol only requires two voting phases per round, plus the initial broadcast of the proposed block at the beginning, and can still guarantee that valid proposed blocks are safely finalized (under the assumption of BFT). Once a block has been committed, this block is binding to all validators, and no one can commit a conflicting block at the same height [6] [13] [33] [37].

Relevance for the Penumbra protocol

CometBFT provides a stable and secure basis as the consensus protocol for Penumbra, which requires enhanced privacy features and has complex zero-knowledge constructions. The deterministic finality guarantees that a state that has been confirmed is consistent and cannot be reversed (providing the foundation for Penumbra's zero-knowledge proofs). A key feature for Penumbra is the modularity provided through CometBFT's ABCI. This clearly separates application and consensus from each other and allows further development on the application layer independently of the consensus layer. This is very useful for Penumbra, as it is still in development and has some planned features, meaning it does not need to be concerned with the underlying consensus mechanism.

4 Privacy and design analysis

The Penumbra protocol provides a secure blockchain system through the usage of various mechanisms that have been introduced in previous sections at a conceptual level, with a focus on privacy-preserving design principles. The goal of Penumbra is to provide comparable functionalities to other well-known blockchain systems (which are often transparent) while enhancing privacy, addressing the limitations of existing systems. This chapter will analyze the privacy-preserving mechanisms that the Penumbra protocol uses, looking at how it fulfills privacy and security, with the main focus on *confidentiality*, *integrity*, *accountability*, and *selective disclosure*. Throughout this chapter, the privacy and security guarantees rely heavily on zero-knowledge proofs (introduced in chapter 3, section 3.3.5). They allow the protocol to verify the correctness without revealing any underlying private information [28] [31]. This chapter does not, however, intend to go over a formal security proof or a comprehensive threat model, establishing cryptographic security guarantees in a formal sense. Instead, it will solely discuss how each individual component and its design achieve specific privacy properties. The cryptographic constructions of Penumbra are assumed to be secure and correct based on third-party audits, peer-reviewed research, and existing primitives. It is also assumed that an honest majority set of validators exists, as required by the CometBFT, and that we are in a partially synchronous network model. Doing a full adversarial analysis is outside the scope of this thesis.

4.1 Methodology of the analysis

The analysis is based on the CIA-triad (Confidentiality, Integrity, Availability), an approach from information security but adapted and extended to blockchain systems [9] [34]. In the context of Penumbra, the privacy and security goals are:

- **Confidentiality**: The protection of information (such as transaction and state information) from unauthorized actors and ensuring only authorized actors can access it.
- **Integrity**: The guarantee that transactions and global states are correct, complete and reliable as well as the protection against manipulation and double-spending.

- **Accountability**: Closely linked to transparency and trustworthiness, accountability represents traceability and linkability of actions in the system.
- **Selective disclosure**: The ability to verify certain information (disclosure) without compromising overall privacy.

The evaluation of Penumbra's privacy and security is based on the protocol specifications, the mechanisms and cryptographic primitives introduced in chapter 3, as well as external audits [38] [58]. Penumbra had many third-party security audits, which all focused on the cryptographic construction and protocol implementation. Those audits have found throughout the (ongoing) development of Penumbra a number of implementation-level issues, but never was a fundamental flaw in the underlying privacy design reported (at the time of writing), hence, it supports the assumption that the protocol design aligns with current best practices. A discussion of the individual findings of those audits would be again out of scope for this thesis.

4.2 Confidentiality

In Penumbra, confidentiality is an important concept, especially in the privacy-DEX, where being decentralized entails significant privacy risks. Confidentiality is a core requirement for privacy. It prevents linkability, protects crucial information such as transaction values, the identity of the participants, or internal state changes inside the shielded pool. It also hides those information from network observers, assuming the protocol provides correct cryptographic primitives and an honest execution. If it did not enforce this confidentiality, then crucial information in transactions or activities in general would be transparent, essentially representing classical blockchains (like Bitcoin), which Penumbra aims to improve by providing privacy. Its goal is to prevent such transparency by providing and enforcing privacy.

Penumbra provides confidentiality mainly through three mechanisms: commitments, encryption, and zero-knowledge proofs (which play a central role). The confidentiality is provided through its multi-asset-capable shielded pool, where all activities occur in shielded form, using commitments and encryption. Since all actions happen inside this pool, it enforces this privacy (privacy-by-default), boosting confidentiality even further [28] [31] [39].

- **(Note- and Balance-) Commitments and encryption**: The amount of currency and to whom they belong are in Penumbra represented through notes. Those notes are never published in plaintext, essentially, only the owner has them stored locally (in private). Only commitments to those notes (note commitments) and actions/operations with them (balance commitments) are revealed/published into the global state inside the shielded pool. As those commitments provide hiding and binding, it means confidentiality is ensured. Also, the encryption of payloads enforces confidentiality of the contained information, since encryption in general provides confidentiality under standard cryptographic assumptions. This means that only entities who have the corresponding secret key can decrypt the ciphertext. Since the ciphertexts are computationally indistinguishable from random for any efficient adversary makes them secure and private.
- **Zero-knowledge**: A primary reason why confidentiality in Penumbra can be ensured while still allowing for it to be operable (i.e., perform transactions, swaps, or other operations) are the zero-knowledge proofs, specifically SNARK-based zero-knowledge proofs (Groth16), adapted to the multi-asset shielded pool. The properties of it guarantees that no additional (private) information besides the truth of the statement is revealed. In Penumbra, the existence of a note, if it is unspent, and the ownership of the note, or that a transaction/vote proceeds according to the protocol, can all be proven without disclosing the content and identities.

Penumbra provides a high level of confidentiality. This becomes especially clear when compared to other typical transparent blockchains. Penumbra addresses the privacy weaknesses of those transparent chains and enforces privacy by default, elevating itself from some other privacy-preserving blockchains, such as Zcash, that only provide it as an opt-in feature. Penumbra, however, does not fully eliminate information leakage, despite it providing strong cryptographic guarantees. Metadata leaks can still be observed on the network layer through transaction timing and network-level metadata (for example, through participation patterns). As Penumbra does not address this form of leakage directly in the protocol itself, rather it relays it to external systems such as network-level privacy tools, the privacy guarantee that the Penumbra protocol provides should be understood as primarily protecting transaction content, not all possible side channels.

With regard to information leakage, the incorporation of IBC for the multi-asset capability in Penumbra also results in possible information leaks. However, this is somewhat inevitable, as transferring assets from a transparent public chain into and out of Penumbra requires a transfer request that provides a destination address. As those transfer requests come from outside or leaf Penumbra's shielded pool, it means the addresses are publicly visible and thus can be linked. Penumbra circumvents this by randomizing the IBC addresses, however, the fact that assets enter or leave Penumbra via IBC remains observable on the counterparty chain. This is a perfect example that shows a necessary trade-off in terms of privacy in order to gain interoperability, something that cannot be completely mitigated by the protocol. Those leaks are out of scope for the Penumbra protocol, as they are inherited through the underlying networking and interoperability (especially as such leaks are not explicitly addressed in the documentation itself). A way to prevent such leakages would require using further anonymity systems, such as, for example, *Tor* [20]. Further challenges bring the zero-knowledge proofs, which are very powerful but introduce complexity, making implementations error-prone. A critical failure, for example, in a spending transaction, could have severe consequences. Also regarding keys, loss or theft of viewing keys (due to user error) can compromise wallet confidentiality over the long term.

4.3 Integrity

For a private blockchain like Penumbra to maintain integrity, the system must always remain in a valid state. This requires prevention of invalid state transitions, that transactions cannot be manipulated and no double spending is possible, all while not disclosing the necessary information for validation. Those properties of integrity are also closely related to safety/agreement and depend on the guarantees provided by the underlying consensus protocol. Mechanisms in Penumbra that guarantee this integrity are:

- **Tendermint/CometBFT**: Penumbra uses CometBFT as its consensus protocol, which implements the Tendermint consensus, guaranteeing that all participants in the network will agree and operate on the same state [13] [33] [37]. Due to the deterministic finality of blocks that CometBFT guarantees, meaning once a block is committed and included in the chain, it cannot be reverted or manipulated. This results in strong consistency across all participants and, consequently, strong integrity, since blocks are only included on chain if they are valid and approved by a supermajority valdators.
- **Nullifier-set**: Penumbra prevents double-spending, similar to Zcash, through nullifiers and the publicly available nullifier-set (record of all published nullifiers) [31] [39]. Every note has its own unique nullifier, which will get revealed/published when spent on the public nullifier-set to prevent the same note from being spent twice. Nullifiers are publicly verifiable without revealing the specific note, since nullifiers are derived from a spend proof. This makes the underlying note private while still ensuring uniqueness and allowing verification.

- **Spend- and Output-Proofs**: Those two proofs guarantee correctness of the state transitions without disclosing, or the need to know, the underlying information, due to zero-knowledge proofs [39]. Zero-knowledge proofs verify that a proof is correctly constructed, that the nullifier corresponds to a valid and unspent note owned by the prover, and that value is conserved between inputs and outputs (i.e., the sum of those is equal to zero), without disclosing any private information. Additionally, it is not possible to modify the content of the proof without invalidating the whole proof itself, making the proof tamper-resistant and thereby ensuring correctness and strong integrity [5].
- **Private DEX consistency**: The private DEX in Penumbra aggregates orders through a batching mechanism (mitigating front-running and MEV). This allows to verify the correctness of executions and that state transitions are consistent with the underlying matching rules through zero-knowledge proofs, all while keeping the individual orders private. This prevents manipulation and guarantees correctness of pricing and liquidity accounting.

Integrity essentially ensures that the protocol follows the protocol rules (staying in a correct state), all while not needing to know any private information. The integrity in Penumbra relies on well-established mechanisms. Correctness and soundness of the used cryptographic primitives, especially the zero-knowledge proofs [5] [28], as well as the collision resistance of the commitment hash function [47], are assumed. Integrity depends mainly on the underlying consensus protocol CometBFT, which requires an honest-majority set (the assumption that fewer than one-third of validators behave maliciously) and a partially synchronous network (i.e., the network eventually reaches synchronicity).

Through the DEX mechanisms (such as batching and threshold encryption) and cryptographically preventing invalid state transitions and double-spending, Penumbra ensures integrity and safety. However, the protocol does not provide censorship resistance, meaning validators are technically able to delay or even exclude valid transactions without being able to decrypt their contents. This shows that Penumbra prioritizes correctness and safety over guaranteed liveness under adversarial conditions.

Regarding the use of zero-knowledge proofs, those introduce significant complexity. Thus it is very important that the implementation of zero-knowledge proofs is done with utmost care and guarantee of correctness. The same can be said about the batching logic. Otherwise, any error emerging due to those components can no longer guarantee the integrity or safety of the protocol, despite the soundness of the underlying cryptographic assumptions.

4.4 Accountability

Accountability is most likely one of the hardest properties to fulfill in a privacy-oriented system, as privacy and accountability are opposites of each other. To hold someone accountable for something requires evidence, which is difficult to get if everything is kept private. In contrast to confidentiality and integrity, accountability in Penumbra is not achieved through public verifiaction and traceability, but through cryptographically controlled disclosure. Penumbra aims to solve this tension through its governance model (private single votes, publicly aggregated total votes) and different keys (key hierarchy), which together allow optional insight into certain aspects without compromising the whole privacy [39] [48].

- **Viewing Keys**: Penumbra has a key hierarchy that provides different levels of access and visibility. The one most prominent for accountability is the viewing key, which can be even further specified into outgoing and incoming viewing keys. Those viewing keys allow only the viewing of transactions of an account (or, respectively, only the incoming or outgoing ones). Since the addresses are unlinkable and not associable with a global identifier, this also protects the identities involved in a transaction, so only the content is viewable, which allows holding someone accountable, exactly what accountability represents. Account owners can give away those viewing keys.

- **Governance mechanisms**: Penumbra implements a governance mechanism that provides the necessary transparency for deciding on a proposal while keeping individual votes of delegators hidden. The delegator votes assigned to a specific validator are essentially collected together as one and given to the validator, who then publicly votes based on the total amount of their delegation. This way, validator votes are public, while the delegations stay private. This provides accountability, and through various mechanisms explained in the previous chapter, delegators and validators can be held accountable for their actions, even when private (e.g., through *slashing* of delegation tokens).
- **Accountability through selective disclosure**: Through the various keys (such as the viewing keys), a user can selectively disclose certain transactions in order to comply with potential legal requirements (such as preventing illegal activities or tax fraud). This allows holding a user accountable but still provides a strong level of privacy as only selective information is made visible (e.g., a user can have multiple addresses, and each of those has different viewing keys, so only one address will be viewable).

The accountability in Penumbra is designed in a way that it is possible to disclose information without compromising the whole privacy. It was mentioned throughout this thesis that accountability and privacy are two properties that are in high tension with each other. The approach Penumbra decided to take allows it to provide both, although in slightly reduced ways, at the cost of some trade-offs and primarily in favor of privacy. Through its key architecture, Penumbra allows accountability on demand by providing selective disclosure. The governance model, coupled with its delegation mechanisms, allows traceability in this regard too while keeping the privacy of individuals. However, as already hinted, there are some drawbacks related to the design choice of Penumbra. Probably the biggest sacrifice that Penumbra has taken in favor of privacy first, is that it does not provide a mandatory global accountability, only a conditional one. Each user has control over their keys, hence, they control (selectively) what they want to disclose. Thus, Penumbra is not suited for an environment that requires mandatory, systemwide auditability. This may seem like a technical flaw, but it is a deliberately designed choice in order to put privacy upfront. As a result, the accountability in Penumbra inherently depends on the user's behavior and key management. This in return means that global auditability can only be possible if users or institutions deliberately disclose specific keys or transaction views. This highly limits the ability of third parties to perform independent audits in case of non-disclosure. However, this does not mean it is impossible to comply with legal requirements. Due to the possibility of selectively disclosing certain information (selective disclosure, discussed in more detail in the next section), it is possible to still provide some degree of accountability and auditability. Lastly, another drawback is related to viewing keys. In Penumbra, a user can have multiple addresses, each having their own set of keys. When disclosing, losing, or leaking a specific viewing key without care, it can lead to long-term insight into all transactions related to the corresponding address. Although this does not compromise the privacy of all addresses belonging to a user, it still compromises the privacy related to the address of the key. This then leads to another crucial challenge Penumbra faces: user behavior/error. Since the user is the one in control over the keys, it means user discipline is crucial.

4.5 Selective disclosure

Selective disclosure is strongly linked to accountability, hence its mention there as well. However, it is included here as a separate property, as it is an important aspect in its own right: how Penumbra enables user-controlled privacy-preserving disclosure in the context of regulatory law enforcement. Selective disclosure allows providing pieces of information without giving away the full information about all related/linked transactions for regulatory purposes. This selective disclosure is implemented through various levels of "access rights", a hierarchical key structure, and so-called transaction views [39] [52]. Important to point out is that such disclosure is always controlled and initiated by the users, no other entity or third party is able to access transaction data from other users in Penumbra without their consent.

- **Transaction perspective/view**: Users can generate perspectives, containing decryption keys that allow to decrypt and generate views that disclose certain notes, individual transactions, or metadata (for example, what kind of transaction it is (spend, swap, delegation) or memo fields).
- **Key hierarchy**: As already mentioned in the accountability section, there are the viewing keys, but there exist more keys. They can be roughly separated into three levels. The strongest one with respect to an address (belonging to an account) is the spending key, which allows spending and therefore should not be disclosed. Then in the middle are the viewing keys. Lastly, the weakest ones are the detection keys that allow detecting transactions belonging to the corresponding key without knowing any details about the address or the transaction itself. This allows to selectively disclose what information can be seen or made public.
- **Structured selective disclosing mechanisms**: With the above-mentioned mechanisms, users are able to control with high granularity what they want to (or have to) disclose. This is very useful for the law enforcement, such as tax audits, but also for evidence of compliance or internal company accounting.

Selective disclosure serves an important role in Penumbra, as it allows the reveal of selective information in a controlled way, without compromising the whole privacy. It achieves this by giving away certain levels of access/viewing rights. This may impact privacy to a certain degree but is a necessary compromise in order to provide some way of accountability, auditability, and compliance with law regulations/enforcements. Since any form of information leakage (deliberately or not) can introduce some risks. Especially since in Penumbra, the responsibility of selective disclosure is shifted towards the users. The users can give away their specific keys for disclosing information, but this should always be done only to trusted entities, as misuse or getting into the wrong hands can cause long-term privacy compromise for the corresponding address (as, for example, the viewing key allows permanent insight to that address). Thus it is important, or even required of the users, to carry out careful key management and to have contextual awareness. This is especially the case for detection keys. By giving away multiple different detection keys to the same detection entity, it can cause linkability, resulting in privacy implications. Overall, this greatly shows that the selective disclosure in Penumbra is a powerful and useful mechanism, yet it has to be handled with the corresponding care. It enables auditability and regulatory compliance, but it is not automatic (user-driven) and comes not without any risks. This greatly shows again the trade-off that has to be taken in order to provide both worlds, privacy and accountability.

4.6 Threat model, availability and network attacks

In the CIA-triad, the A stands for availability. However, as the Penumbra protocol does not introduce mechanisms that address availability specifically, rather, the underlying consensus protocol is responsible for this, availability has been replaced by accountability in this analysis. Additionally, since the Penumbra protocol does not formally specify a global adversary model, the analysis assumes a partially synchronous network, an honest-majority validator set, and secure cryptographic primitives (as in other SNARK-based systems, such as Zcash). Hence, on the protocol level, Penumbra does not protect against global passive network adversaries. However, *availability* is not something to completely ignore in this analysis. In section 4.3, which discussed integrity, validator censorship was mentioned. Penumbra does not provide any additional censorship resistance beyond what the underlying consensus protocol, the CometBFT consensus, does. This means that, if a sufficiently large group of validators censor specific transactions, even without being able to decrypt them and know the contents, censorship can occur. With regard to network-level attacks, such as timing attacks or network correlation, the Penumbra protocol does not provide any built-in defenses against such global passive network adversaries. Thus, this is out of scope for the Penumbra protocol and this analysis, hence will not be further discussed.

4.7 Comparison, limitations and trade-offs

Penumbra is not the only privacy-oriented blockchain. Two such systems have been mentioned in this thesis, which are similar to Penumbra. Those are Zcash [3] and Monero [55].

Comparison with Zcash

Zcash was one of the earliest widely deployed cryptocurrency systems that made use of zk-SNARKs in order to provide privacy in the world of transparent blockchains. Many mechanisms that Penumbra provides are taken as inspiration from Zcash, hence, there are a lot of similarities. Zcash uses commitments and nullifiers as well to prevent double spending, and it provides selective disclosure through viewing keys. It also uses zk-SNARK to guarantee privacy without disclosing any secret information. However, there are some differences. The first main difference is that Zcash provides privacy but not as the default, in general it still operates like a transparent blockchain but users can opt in to use privacy mechanisms. In Penumbra, there is no choice, privacy is enabled by default. Another big difference is that Penumbra provides a multi-asset model, whereas Zcash only has a single-asset model, hence, in Penumbra, commitments and nullifiers are conceptually similar but more generalized for multi-asset usage. Zcash was developed to only provide protection (privacy) in its own crypto system, i.e., for ZEC transactions. Penumbra aimed to be a universal crypto system where different assets can be transferred into the Penumbra environment, allowing operations to be done in its own protected environment, i.e., shielded pool. This provides very high flexibility but introduces large cryptographic complexity. With regard to selective disclosure, Penumbra builds on this concept further by providing even more fine-grained access levels, for example, through detection keys or transaction perspectives. The governance mechanism follows a similar improvement in Penumbra. In Zcash, the governance is mainly public and transparent, it provides the possibility for some actors to stay anonymous, but the whole process stays public. Penumbra essentially took this possibility for anonymity and enforced it on the whole governance mechanism, such that individuals (i.e., delegators) are kept private/anonymous, but the complete vote/process, through validators and their aggregated votes, is kept public in order for it to remain auditable and verifiable. Finally, Zcash does not use or have its own DEX, it relies on external DEX protocols. Penumbra, on the other hand, uses its own DEX engine that allows for private swaps and batching of transactions directly in its own shielded pool. Overall, the differences between those two protocols can be summarized with which design principle they follow. Penumbra follows a privacy-first design principle, whereas Zcash uses a currency-first design, focusing more on payments (with additional privacy). Penumbra takes some concepts from Zcash and expands those further, leading to higher privacy but at the cost of more complexity [3] [31] [42].

Comparison with Monero

Monero is taken as a second candidate for comparison as it is also a privacy-oriented blockchain, but it achieves it differently without the usage of SNARKs. Monero's privacy relies on ring signatures (which provide the sender anonymity), uses stealth addresses (to provide anonymity for the receiver), and uses RingCT (Ring Confidential Transactions) [46], which hide the transaction values. Both Monero and Penumbra follow the same design principle, privacy-first, keeping all assets private. Hence, Monero as well as Penumbra do not have a transparent mode (unlike Zcash), resulting in transactions always being private, there is no opt-in mechanism. Monero and Penumbra use commitments to hide (and bind) values and remain consistent at the same time. Thus, the way double spending is prevented is also similar. Monero uses so-called key images that act as a unique one-time identifier to prevent double spending without disclosing the identities of the spending action, similar to nullifiers in Penumbra, but Penumbra uses those in a SNARK-based model. Monero obviously has many differences from Penumbra, mainly due to how it achieves privacy without using SNARK. Hence, its anonymity essentially depends on the ring size, meaning small rings or pattern recognition can lead to a reduction in anonymity. Penumbra does not have such a problem, as privacy is mainly achieved through its completely shielded pool.

A big difference is that Monero, like Zcash, only supports a single-asset model. Monero also, in a way, provides selective disclosure, but only in a restricted manner. It uses a viewing key, but with that key all transactions are viewable, technically not resulting in ”selective” disclosure (only disclosure). Penumbra provides significantly more granularity in what to reveal and what not, through its diversified addresses, key hierarchy, and transaction perspectives, without disclosing every single detail. Another difference is how they handle correctness and integrity. Monero uses RingCT, which is effective but produces larger proofs that depend on complex cryptographic assumptions. Penumbra uses SNARK proofs, which provide smaller proofs and are better suited for usage with multi-assets. Overall, Monero provides strong privacy without the need to have a trusted setup, but it has its limitations because of the ring sizes and the restricted disclosure. Penumbra provides a theoretically stronger privacy, can provide more controlled disclosure of information, and supports multi-asset operations, but all those features again come at the cost of complexity (more complex cryptography needed) [36] [46].

Limitation and trade-offs

Penumbra provides many beneficial properties and functionalities, but it is not perfect. Due to the inherent difficulty of achieving privacy while still providing auditability and accountability, it comes with some limitations and risks. Penumbra solves the issue with privacy and verifiability through the use of zero-knowledge SNARK proofs and extensive proof logic mechanisms, but this introduces more complexity, thus bringing new classes of potential vulnerabilities, making formal verification harder [8] [28] [31]. The introduction of additional access keys for selective disclosure, like viewing keys, detection keys, and spend keys, requires disciplined handling, since the users are the ones in control and are never expected to act perfectly, i.e., errors are part of human nature. Incorrect disclosure or leaks can result in long-term privacy compromise. Additionally, regarding disclosure, since users are in control of those keys, auditability is only possible if the users voluntarily disclose information (or rather, comply with legal requirements). This is especially true if it is required for a global, system-wide analysis, which represents a limitation of Penumbra, namely its restricted global auditability. Finally, besides the complexity of the proof system, it also introduces performance costs, as the creation and verification of zero-knowledge proofs are more computationally intensive than classical, transparent transactions.

Since Penumbra acts in a field of tension between privacy and auditability it introduces or rather requires some trade-offs in order to provide aspects of both worlds. Through selective disclosure, it is possible to provide accountability while keeping some level of privacy (trade-off towards auditability), but again this is only possible on a voluntary basis by users in Penumbra, hence, global transparency (or any mechanism providing it) is not possible (trade-off towards privacy). Also, providing this level of privacy, i.e., security in the form of zero-knowledge proofs, introduces more complexity and requires more computational power, hence, efficiency is lower compared to classical transparent approaches (trade-off between security and efficiency). Another trade-off Penumbra must accept is in its trading capabilities, which cannot be real-time, as it uses a DEX and batching mechanisms that prevent MEV but introduce a necessary delay [11] [39] [48]. Additionally, the decision of Penumbra to be multi-asset capable introduces further difficulties in terms of implementation and auditing, hence, the Penumbra system is more difficult and complex than classical protocols [39] [58], but is therefore more powerful at the same time.

5
Conclusion

The main purpose of this thesis was to analyze the Penumbra protocol systematically and show how it provides such strong privacy and which mechanisms and cryptographic primitives it uses to achieve this while still being verifiable at the same time. Furthermore, an evaluation of Penumbra's privacy with respect to the properties of confidentiality, integrity, accountability, and selective disclosure has been conducted, as well as a comparison between Penumbra and other privacy-preserving blockchains, and Penumbra's limitations and trade-offs have been discussed. Guided by the research questions introduced in chapter 1, this chapter summarizes the results of the thesis.

Privacy, verifiability, and design analysis

The analysis and evaluations have shown that Penumbra achieves a strong level of privacy without neglecting auditability, i.e., a degree of transparency. With regard to the first research question, *how does the Penumbra protocol, a privacy-preserving blockchain, achieve privacy at the architectural and cryptographic level?* We have seen in the analysis that Penumbra provides strong privacy. In Penumbra, all transactions and/or state changes appear shielded inside the multi-asset-capable shielded pool, which, in combination with zero-knowledge proofs, commitment schemes, and nullifiers, allows the verification of correctnes without the necessity of needing to know confidential information. By recording note commitments in the state commitment tree, transaction values, assets, and participants are ensured to remain confidential.

Privacy and transparency, or verifiability, are inherently two conflicting properties. However, Penumbra has shown that it does not entirely require excluding one in order to (partially) provide the other. With regard to the second research question, *how does Penumbra balance privacy with verifiability and selective disclosure?*, Penumbra provides privacy and transparency, allowing auditability and verifiability without completely sacrificing privacy or the other way around. This, although, comes with certain restrictions in place, since providing complete privacy and complete traceability is not possible due to their inherent contradiction. By providing a fine-grained key hierarchy, as well as a transaction perspective, Penumbra is able to hold a high level of privacy while at the same time being (to some extent) transparent, in the sense that it is possible to verify transactions without knowledge of the underlying information, and through the selective disclosure mechanism it is possible to disclose specific information for regulatory or compliance purposes. However, this selective disclosure cannot provide global auditability. But this restriction, although a slight drawback, is chosen deliberately, i.e., is a design choice and trade-off, in order to prioritize the protection and privacy of users (privacy-first). Importantly, through encrypted state representations and zero-knowledge proofs, the validity and correctness can be verified (by validators) without even needing to have knowledge of sensitive data (i.e., it is not even necessary to disclose information in order to be able to check correctness). This essentially enforces privacy and allows for cryptographical verifiability. The privacy-by-default approach that Penumbra pursues does not affect the protocol or the privacy and security properties negatively. Through the deterministic finality provided by the consensus layer, through the usage of commitments and nullifiers, and through the cryptographic verification, it guarantees the consistency and correctness of the global state of the protocol.

Related to the second research question is the third one: *how does the Penumbra privacy design address confidentiality, integrity, and accountability requirements?* In the analysis, we have seen that the design choice of Penumbra having a privacy-first architecture still addresses confidentiality, integrity, and accountability (as well as selective disclosure). The shielded pool of Penumbra ensures confidentiality primarily. It represents a place where everything occurs and is recorded in encrypted form through state commitments inside the state commitment tree. Those state commitments are verified by zero-knowledge proofs, preventing the recording of faulty commitments and preventing any necessity of information disclosure. Integrity is provided by the cryptographic commitment schemes, the nullifiers, and the state commitment tree. The combination of those guarantees value conservation, prevents double-spending, and enforces correct state transitions. All this is done while hiding sensitive information yet still being verifiable by the protocol through checking correctness by validators. Lastly, accountability (and, related to it, selective disclosure) is provided through the key hierarchy, especially through the viewing keys and transaction perspectives. Penumbra does not provide global transparency in order to protect its users, rather, it shifts accountability from public traceability to cryptographically verifiable disclosures initiated by users.

Limitation of the analysis

All those findings and conclusions highlight how Penumbra's design addresses privacy-related requirements. However, it is important to point out that the analysis performed in this thesis has several limitations. The first limitation is that the analysis is primarily based on protocol specifications, official documentation, and some external audits. There is no formal verification or implementation of the Penumbra protocol. Thus, the conclusion drawn only reflect the intended design and documentation behavior of the protocol. Another limitation is that the evaluation of the complexity, the performance cost, and the deployment costs is only discussed qualitatively. This means that no empirical performance evaluation is performed. The last limitation to mention is that the analysis focuses only on the protocol design and the used cryptographic primitives. Implementation vulnerabilities, such as those found in external audits, or user key management risks and network-level insecurities (beyond what is discussed in the thesis), are not explicitly taken into consideration. With these limitations in mind, the previous as well as the upcoming discussion about Penumbra's limitations and trade-offs focuses only on the inherent privacy-first design choices that Penumbra makes.

Limitations and trade-offs of Penumbra's privacy-first design

That brings up the last stated research question: *what are Penumbra's limitations and trade-offs for a privacy-first design?* The previously mentioned disclosure of information initiated by users means that compliance regarding law regulation is only possible if all users willingly participate with it. This essentially shows that trade-offs are unavoidable if one wants to provide privacy as well as accountability and auditability, as those can only exist separately in complete form. Additionally, the benefits of privacy that the privacy-first design provides come at the cost of increased complexity of the protocol. Due to complex zero-knowledge proofs and circuits, additional logic is required to operate with multi-assets, and the dependencies, as well as trust in proper key management, result in a higher implementation effort and the reliance on advanced cryptographic primitives. Along with complexity also comes the increased execution time, due to the required additional cryptographic verification processes. A very notable trade-off due to Penumbra's design choice of how it prevents MEV through batching to ensure privacy and fairness is that it is not suitable for a real-time trading platform (due to transactions not going through individually and immediately). Most of those limitations and trade-offs are, however, made consciously in order to maintain the design of privacy-first, meaning they do not necessarily represent a weakness of the protocol.

In comparison to other privacy-preserving blockchains, such as the two discussed ones, Zcash and Monero, Penumbra stands out due to its multi-asset capability and, linked to that, the interoperability through the IBC. Many protocols, transparent or private ones, often rely solely on their own single asset, which obviously reduces the complexity, and regarding privacy-preserving blockchains, reduces the difficulty of keeping that asset private. But this highly limits their usability. Through Penumbra's choice to be embedded in the Cosmos ecosystem, where the majority of protocols support IBC [49], it allows the transfer of any transparent asset into Penumbra's shielded pool, allowing transactions inside Penumbra with its level of privacy guarantee. Thus making Penumbra a very versatile protocol in the area of privacy-preserving blockchains.

Future work

The analysis of the Penumbra protocol in this thesis focused on the understanding of the (intended) design, architecture, and documentation behavior of the protocol. There are many more areas/focuses that deserve more analytical observation, which, however, were out of scope for this thesis. Hence, this brings up possible future work ideas. One area to look into in more detail would be the performance, i.e., perform a quantitative analysis by implementing the Penumbra protocol and playing with it in a toy scenario, measuring various metrics, like proof generation time, cost of verification, or throughput under realistic network conditions. Another direction for future work would be to do more of a formal cryptographic verification, going through the cryptographic mechanisms and doing an in-depth security analysis with proper threat modeling, observing and pointing out concrete attack scenarios, analyzing vulnerabilities shown in external audits, and discussing economic attack vectors.

Final remark

All together, this analysis of the Penumbra protocol showed a possible design choice for privacy-preserving blockchains with interoperable capabilities. It showed that it is, in fact, possible to provide mandatory privacy with some disclosure mechanism to provide at the same time some degree of transparency, more specifically, providing a way for auditability and accountability, even though privacy lies in high tension with those properties. With appropriate cryptographic mechanisms and taking in some limitations and trade-offs, Penumbra showed that it is possible to provide them, coexisting together. The development of the Penumbra protocol is still ongoing at the time of writing, this means it may be possible that the addressed limitations and trade-offs will be improved, or even changed, in the future. This is especially likely as research in the area of cryptographic mechanisms, such as zero-knowledge proofs, formal verification techniques, and even the development of new hardware, is still thriving. The Penumbra protocol combined many mechanisms, some of which are available only in isolation in other privacy-preserving blockchains, and added its own functionality by being multi-asset capable and interoperable, making it a versatile blockchain, applicable as inspiration for future privacy-preserving blockchains.

Bibliography

[1] Stellar Magnet (Lead Analyst) and Black Sky Nexus. Penumbra protocol. `https://nexus.blacksky.network/zine/00000001/penumbra-protocol`, 2024. Accessed: 2025-10-27.

[2] Andreas M. Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain*. O'Reilly Media, Sebastopol, CA, 2nd edition, 2017.

[3] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474. IEEE Computer Society, 2014.

[4] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of clients in bitcoin P2P network. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 15–29. ACM, 2014.

[5] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319, 2004.

[6] Ethan Buchman, Rachid Guerraoui, Jovan Komatovic, Zarko Milosevic, Dragos-Adrian Seredinschi, and Josef Widder. Revisiting tendermint: Design tradeoffs, accountability, and practical use. In *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2022, Supplemental Volume, Baltimore, MD, USA, June 27-30, 2022*, pages 11–14. IEEE, 2022.

[7] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018.

[8] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 315–334. IEEE Computer Society, 2018.

[9] Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.

[10] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In Margo I. Seltzer and Paul J. Leach, editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999.

[11] Chainlink. Maximal extractable value (MEV). Chainlink Education Hub, `https://chain.link/education-hub/maximal-extractable-value-mev`, 2025. Accessed: 2025-11-21.

[12] CoinGecko. Coingecko: Cryptocurrency prices, charts, and market capitalization. `https://www.coingecko.com/`, 2026. Accessed: 2026-01-25.

[13] CometBFT. *CometBFT Documentation, v0.38*. Informal Systems / CometBFT, 2025. Accessed: 2025-11-18.

[14] European Commission. Crypto-assets – a comprehensive framework for crypto-assets and related services to ensure that the union financial services are fit for the digital age. `https://finance.ec.europa.eu/digital-finance/crypto-assets_en`, 2025. Accessed: 2025-11-14.

[15] Bitcoin Developer Community. bips.dev – bitcoin improvement proposal. `https://bips.dev/`, 2025. Accessed: 2025-11-15.

[16] Monero Community. Monero documentation. `https://docs.getmonero.org/`, 2026. Accessed: 2026-01-25.

[17] Electric Coin Company. Zcash basics. `https://zcash.readthedocs.io/en/latest/index.html`, n.d. Accessed: 2025-10-12.

[18] Electric Coin Company. Zcash technology overview. `https://z.cash/learn`, n.d. Accessed: 2025-10-12.

[19] cosmos/ibc. "interchain standards (ics) — ics-001: Ics specification standard". `https://github.com/cosmos/ibc/tree/main/spec/ics-001-ics-standard`. Accessed: 2025-11-14.

[20] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320. USENIX, 2004.

[21] Everstake. Cosmos (atom) staking insights & analysis: H1 2025. `https://everstake.one/crypto-reports/cosmos-staking-insights-and-analysis-h1-2025`, 2025. Accessed: 2025-11-14.

[22] Ethereum Foundation. Ethereum documentation. `https://ethereum.org/en/developers/docs/`, 2023. Accessed: 2026-01-25.

[23] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985.

[24] Zhenhuan Gao, Yuxuan Hu, and Qinfan Wu. Jellyfish merkle tree. *Diem Association, Tech. Rep.*, 2021.

[25] Rodrigo Dutra Garcia, Gowri Sankar Ramachandran, Kealan Dunnett, Raja Jurdak, Caetano Mazzoni Ranieri, Bhaskar Krishnamachari, and Jo Ueyama. A survey of blockchain-based privacy applications: An analysis of consent management and self-sovereign identity approaches. *CoRR*, abs/2411.16404, 2024.

[26] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[27] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 519–535. USENIX Association, 2021.

[28] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[29] Mike Hamburg. Decaf: Eliminating cofactors through point compression. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 705–723. Springer, 2015.

[30] Martin Harrigan and Christoph Fretter. The unreasonable effectiveness of address clustering. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse, France, July 18-21, 2016*, pages 368–373. IEEE Computer Society, 2016.

[31] Daira Hopwood, Sean Bowe, Taylor Hornby, Nathan Wilcox, et al. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 4(220):32, 2016.

[32] Interchain Foundation (ICF). What is ibc? `https://tutorials.cosmos.network/academy/3-ibc/1-what-is-ibc.html`, n.d. Accessed: 2025-11-14.

[33] Interchain Foundation. Tendermint core documentation. `https://docs.tendermint.com`, 2025. Accessed: 2025-10-11.

[34] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Chapman and hall/CRC, 3 edition, 2021.

[35] Philip Koshy, Diana Koshy, and Patrick D. McDaniel. An analysis of anonymity in bitcoin using P2P network traffic. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, volume 8437 of *Lecture Notes in Computer Science*, pages 469–485. Springer, 2014.

[36] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of monero's blockchain. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*, volume 10493 of *Lecture Notes in Computer Science*, pages 153–173. Springer, 2017.

[37] Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1(11):1–11, 2014.

[38] Penumbra Labs. Penumbra security audits q2 2024. `https://www.penumbra.zone/blog/2024-audits`, 2024. Accessed: 2025-12-08.

[39] Penumbra Labs. The penumbra protocol. `https://protocol.penumbra.zone`, 2025. Accessed: 2025-10-28.

[40] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. In Dahlia Malkhi, editor, *Concurrency: the Works of Leslie Lamport*, pages 203–226. ACM, 2019.

[41] 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 Konstantina Papagiannaki, P. Krishna Gummadi, and Craig Partridge, editors, *Proceedings of the*

2013 Internet Measurement Conference, IMC 2013, Barcelona, Spain, October 23-25, 2013, pages 127–140. ACM, 2013.

[42] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 397–411. IEEE Computer Society, 2013.

[43] Arvind Narayanan, Joseph Bonneau, Edward W. Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction*. Princeton University Press, 2016.

[44] Jonas David Nick. Data-driven de-anonymization in bitcoin. Master's thesis, ETH-Zürich, 2015.

[45] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security - 22nd International Conference, ACNS 2024, Abu Dhabi, United Arab Emirates, March 5-8, 2024, Proceedings, Part III*, volume 14585 of *Lecture Notes in Computer Science*, pages 105–134. Springer, 2024.

[46] Shen Noether and Adam Mackenzie. Ring confidential transactions. *Ledger*, 1:1–18, 2016.

[47] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[48] Penumbra. Penumbra: Private, cross-chain dex and privacy protocol. `https://www.penumbra.zone`, 2025. Accessed: 2025-10-31.

[49] IBC Protocol. How the ibc protocol works. `https://ibcprotocol.dev/how-ibc-works`, 2025. Accessed: 2025-11-14.

[50] Cosmos SDK. *Introduction to ABCI, Cosmos SDK v0.53*. Cosmos Network, 2025. Accessed: 2025-11-18.

[51] Panther Team. Threshold cryptography: An overview. `https://blog.pantherprotocol.io/threshold-cryptography-an-overview/`, 2025. Accessed: 2025-11-21.

[52] Penumbra Guide Team. Penumbra guide. `https://guide.penumbra.zone`, 2025. Accessed: 2025-10-27.

[53] Penumbra Protocol Team. Penumbra protocol github repository. `https://github.com/penumbra-zone/penumbra`, 2025. Accessed: 2025-10-27.

[54] Justin Thaler. Proofs, arguments, and zero-knowledge. *Found. Trends Priv. Secur.*, 4(2-4):117–660, 2022.

[55] Nicolas Van Saberhagen. Cryptonote v 2.0. *Cryptonote Whitepaper*, 2013.

[56] Andrew Vella. Penumbra chain: A privacy hub for the interchain. `https://simplystaking.com/penumbra-chain-a-privacy-hub-for-the-interchain`, 2024. Accessed: 2025-10-27.

[57] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[58] zkSecurity. Public report of auditing penumbra's circuits. `https://blog.zksecurity.xyz/posts/penumbra/`, 2023. Accessed: 2025-12-08.