Technical specification |
Example EVM Sidechain

Version: 1.0

Supplied by: IOG

Authors:

Dominik Zajkowski
dominik.zajkowski@iohk.io

Kasper Kondzielski
kasper.kondzielski@iohk.io

Lech Głowiak
lech.glowiak@iohk.io

Oleksii Avramenko
oleksii.avramenko@iohk.io

Aurélien Richez
aurelien.richez@iohk.io

Joris Beau
joris.beau@iohk.io

Abstract

This document presents a novel approach to solving the problem of transferring value between two or more blockchains. The document is divided into two parts. The first part explains the concept of a Cardano sidechain, including the rationale for design decisions taken, and also introduces the Cardano sidechain toolkit. The second part shows how all these concepts work together in the form of an example EVM sidechain.

Document Information

Version Control			
Version Num.	**Date**	**Summary of Changes**	**Updated By**
1.0	Jan 2023	First public release	Dominik Zajkowski

0 Introduction

Main goals

This document has these three goals:

1. Thoroughly explain the main chain and sidechain and their interaction patterns.
2. Explain how the example EVM sidechain is one possible instantiation of a Cardano sidechain's design.
3. Introduce the concept of a sidechain toolkit and how the example EVM sidechain is an instance of said toolkit.

Out of scope

Some outstanding concepts are intentionally omitted in this first iteration of a technical specification:

- Tokenomics: no ready-to-go production deployment is proposed or paths forward discussed.
- Using consensus protocols other than OBFT: this technical specification assumes that the sidechain node client leverages OBFT. While it is possible to use other consensus protocols, this technical specification does not cover such cases.
- Checkpointing: this document covers checkpointing very superficially.
- Sidechain upgrade strategy: upgrading a sidechain is not covered by this technical specification.
- Relay actor: the actor capable of enacting change on the main chain.

Overview

Section 1 ('Introduction to Cardano sidechains') explains the concept of a sidechain and its relationship with the main chain.
Section 2 ('Main chain') describes the role of the main chain and how communication patterns look from the main chain's perspective.
Section 3 ('Sidechain') covers the role of the sidechain and what communication patterns look like from the sidechain's perspective.
Section 4 ('Cardano sidechain toolkit') introduces the sidechain toolkit.
Section 5 ('Example EVM sidechain') combines the first four sections and introduces the example EVM sidechain.

Table of Contents

List of figures

List of samples

1 Introduction to Cardano sidechains

What is a sidechain?

This document uses the term sidechain in two different ways. First, it refers to a communication pattern between two chains. In this context, one chain (called the main chain) remains unchanged, while the other chain (called the sidechain) is heavily modified and depends on the main chain to function. This relationship allows data from one chain to be represented on the other.

The second meaning of sidechain in this document refers specifically to the chain that is dependent on the main chain.

To establish a communication channel to the main chain, it must have on-chain facilities that allow it to run on-chain applications capable of verifying certificates of authenticity for facts originating from the sidechain. Cardano uses Plutus scripting to achieve this.

Since the sidechain cannot operate independently, it must follow the main chain and store data in its own on-chain state, as long as the data is considered trustworthy (eg settled) on the main chain. The sidechain must also be able to verify and certify the data it includes in its state. Simultaneously, it needs to be able to operate as a blockchain to:

- Mint blocks
- Accept and process user input
- Achieve consensus

It is also assumed that a group of entities (the block-producing committee) will be responsible for running and maintaining the sidechain.

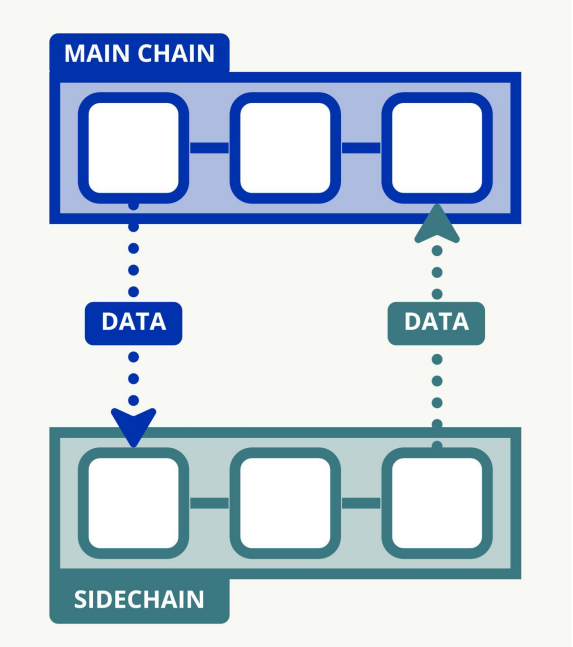

Fig. 1. High-level data flow between main chain and sidechain

The dependency between the main chain and the sidechain is observation. The sidechain can observe trusted (settled) events on the main chain. These events are foreign from the perspective of the sidechain and need to be adapted for inclusion in the sidechain's state. The dependency between the sidechain and the main chain requires enacting change on the main chain. The sidechain produces (certificates) data that is verifiable on the main chain and this data needs to be made available to an on-mainchain application.

The relationship between main chain and sidechain boils down to two streams of data:

- Passive flow: data observed on the main chain that is put into the sidechain's persistence facilities.
- Active flow: data observed on the sidechain and made available on the main chain.

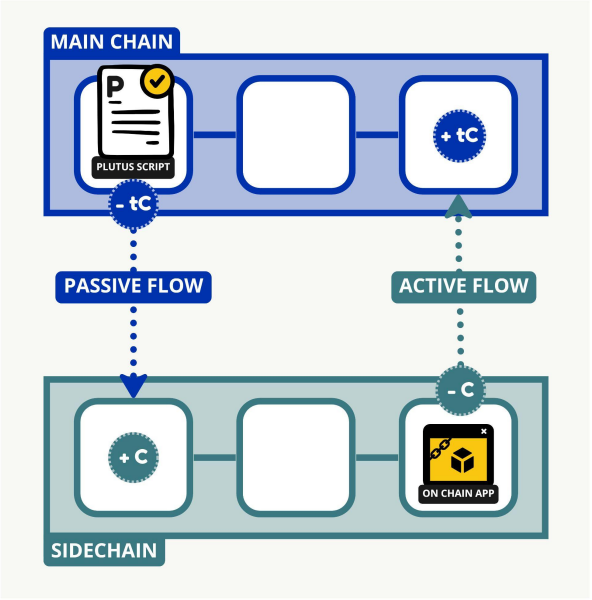

Fig. 2. Example value transfer flow between main chain and sidechain

To illustrate the concepts above, an example use case can be followed.

The passive flow
A user interacts with an on-mainchain application and **marks** tokens tC to be moved to the sidechain. Components supporting the passive flow observe this event and make it known to the sidechain. As part of regular operations, the sidechain observes this event and stores it in its own ledger granting the user the requested tokens C.

The active flow
A user interacts with an on-sidechain application and **marks** tokens C to be moved to the main chain. Components supporting the active flow trigger signature gathering rounds from the entities running the chain. A certificate of authenticity trustable by the main chain is created after gathering sufficient backing. The certificate is made available on the main chain. Next, the user can interact with an on-mainchain application to **claim** their tC tokens.

The example EVM sidechain

A big part of this technical specification is the example Ethereum Virtual Machine (EVM) sidechain. It's beneficial to introduce some basic concepts.

- The example EVM sidechain client: a Scala-based blockchain client capable of running in sidechain mode. Its consensus protocol is Ouroboros-BFT: A Simple Byzantine Fault Tolerant Consensus Protocol (OBFT) [1], capable of executing Solidity contracts (more in Section 5).
- Block producers: OBFT is a permissioned consensus protocol, which means that only a limited and known list of allowed entities can mint blocks.
- Time in the OBFT consensus protocol is divided into slots: this is a chain parameter (**S**) and dictates the size of the window (in seconds) a block producer gets to mint a block.
- Sidechain epoch: to remedy the permissioned nature of OBFT, the concept of sidechain epochs has been introduced. A given committee of block producers is allotted a given

amount of time (an epoch) during which the entities take turns (slots) in minting blocks. When an epoch ends, a new committee is selected and the process repeats.
- Block finality: OBFT satisfies the notion of *k*-consistency, where *k* is the parameter that describes the amount of blocks that need to follow a particular block for it to be considered immutable.

Sidechain definition

The "Proof-of-Stake Sidechains" paper [2] provides a formal rigorous cryptographic definition of a sidechain. It is quite abstract, captures many possible configurations, and identifies common concepts and components that take part in the interaction. For the purposes of this document, the following definition taken from [2] should suffice:

> 'Sidechains, (…) are a way for multiple blockchains to communicate with each other and have one react to events in the other.'

The above definition holds a few key concepts mentioned in the previous subsection. **Participating chains** are at least two chains engaged in data exchange. **Foreign events** are the data items that settle on the observed chain and are of interest to the observing chain. **Communication** stands for a way to move foreign events from one participating chain to another.

The previous subsection also introduced the concept of observation. This is a form of communication that assumes one party being the active participant and the other being unaware of the relationship. It does not necessarily mean that the observer is the active party.

There are two distinct patterns of observing foreign events:
1. Direct observation: a chain is equipped to directly observe foreign events from a source chain by directly following state transitions and chain growth.
2. Certificate-based: an indirect way of observing events on the source chain by posting certificates for these foreign events and verifying these certificates on the target chain without explicit knowledge of the source (observed) chain.

Having defined the basic concepts, it is possible to describe ledger dependencies between the participating chains (named A and B for convenience). One of two models needs to be adapted to handle this relationship:
1. Independent staking: in this model, both chains are independent of each other. The stake ownership of chain A will not affect the block creation probability on chain B.
2. Merged staking: in this model, at least one chain will be affected by stake ownership changes stemming from the other chain. This translates to the stake distribution on chain A having a direct effect on the probability of block creation by a particular entity on chain B.

With the above definitions established, and the high-level requirements from the previous section in place, it is possible to capture the expected characteristics of the main chain in a Cardano sidechain setting:
- A main chain has no special facilities to accommodate the sidechain.
- A main chain is passive in any 'observation' relationship.
- A main chain has facilities to accommodate the 'certificate-based observation' relationship.
- A main chain's stake distribution is not directly affected by the relationship with sidechains.

Bringing these characteristics together, it's possible to refine the above definition:

> A sidechain is a blockchain tied to a main chain via a set of observation patterns that allow data to flow unilaterally or bilaterally. One of the aspects of this relationship is the link to the stake distribution observed on the main chain affecting the behavior of the sidechain.

Note on Hydra Head protocol [3]: One example of a protocol that can be also viewed as a 'sidechain' is the Hydra Head protocol. Like the sidechain proposed in this document, it allows offloading computation from the main chain to the sidechain. However, there are differences in how entities running both protocols are selected. In the case of Hydra Head, it is an arbitrary group capable of running the protocol (it doesn't need to be representative of the mainnet stake). At the same time, the Hydra Head protocol sacrifices liveness for security such that even a single party can't be cheated by all other parties colluding, but any participating party can stall the protocol. Sidechains rely on a particular threshold of assumed honesty that is necessary for both safety and liveness.

Cardano's role in Sidechains Toolkit

This document focuses on a particular use case of a sidechain. The lopsided communication pattern defined in the previous subsection is further refined, allowing for vast variation on the observing chain's part but it assumes that the main chain is Cardano.

Cardano is a large and robust blockchain. It holds a great amount of value and is widely adopted and diverse. One of the aspects of said diversity is a very high degree of decentralization of block creators. This, along with the value held, principled algorithms, and protocols translates into security and trustworthiness. The principled approach has many upsides (rock-solid operation for years, no data lost due to chain operation errors, many hard forks successfully applied, no disruptive hard forks to date), and it also means that innovation and change is purposefully slow but still possible. The value of such a solid structure that is publicly available and utilizable is hard to overstate, and it would be interesting to have an easy to follow path of leveraging such a robust source of trust (**root of trust**) without exerting change in the low-level algorithms and protocols of Cardano. This document proposes 'Cardano sidechains' as one of the possible paths to leverage Cardano as the root of trust.

What is a Cardano sidechain?

Building on top of the previous subsections, it is possible to define a Cardano sidechain: a blockchain that follows Cardano's foreign events, incorporates them into its state, and is capable of producing Cardano-verifiable certificates proving on-sidechain state. This is a very broad definition that will be refined in the following sections. For now it is sufficient to explain why a Cardano sidechain might be useful.

A Cardano sidechain can implement many different consensus algorithms, ledger rules, execution environments, user facing APIs, and much more. Such an abundance of features cannot be covered by any one chain. The goal is to establish a known and easy to follow way of enhancing Cardano and to benefit from Cardano's established security. In other words, a Cardano sidechain is a tool that provides its users with a spectrum of use cases, from experimentation to fully-fledged blockchains while benefiting from a well-established root of trust.

Furthermore, with the flexibility of Cardano sidechains, it is possible to tune the security guarantees and participation expectations to a particular use case, and even to a particular stage of a chain's growth. In principle, this approach is aimed at enabling innovation and growth, removing the need to start with a well-established (well funded) chain just to run experiments, while, at the same time, supporting fully-fledged solutions. Additionally any Cardano sidechain (as defined in this document) is considered 'trustless'.

What does 'trustless' mean?

In this context, *trustless* means that the communication pattern between the two chains does not introduce additional external parties that need to be trusted beyond the Cardano blockchain (there is no additional committee owning a special 'bridge wallet', arbitrage, etc.). The chain's success is solely dependent on Cardano and the entities running the sidechain.

This does not mean that a sidechain automatically inherits any security guarantees of its main chain. The characteristics of a blockchain, including decentralization, security, and scalability, depend on the consensus protocol, committee selection protocols, and the level of integration with the main chain. These factors must be considered when evaluating the security guarantees of a particular sidechain.

Trustless Cardano sidechain

A Cardano sidechain is a communication pattern between two chains, the main chain (Cardano) and a sidechain (some other chain, like the example EVM sidechain). In this relationship, the main chain remains unaware of the sidechain and is only capable of verifying certificates for particular data coming from the sidechain while serving as the root of trust. The sidechain, on the other hand, is fully aware of the main chain, is capable of following its growth and settlement, and can take action on its own ledger based on the events coming from the main chain.

Principles of Cardano sidechain

The problem statement is as follows: 'how to leverage Cardano mainnet security to enable new paradigms, build new communities, run experiments, expand the languages supported, and much more'. In other words, how to leverage Cardano to enable its own community to bootstrap new and expand existing ecosystems, and facilitate purpose-built Web3 solutions. Meeting these objectives leads to the following design principles:

1. Enablement without disruption.
2. Utility at low cost (entry, usage).
3. Permissionless extensibility.
4. Bootstrapped security and community.

Enablement without disruption

The Cardano main chain is unaware of its compatibility with any sidechain. No special changes or allowances are needed, and all the building blocks are in place to run Cardano sidechains. The solution may require some enhancement in the form of CIPs [4, 5] that would improve the sidechain's end-user experience, as well as ease the creation of future chains. In principle, it is possible to create a sidechain without disrupting the Cardano main chain.

Utility at low cost (entry, usage)

The design focuses on keeping the sidechain's operational parameters flexible. A sidechain can be as small as one node governed by nothing more than a single UTXO on Cardano main chain, or as

large as the whole Cardano's SPO group. By leveraging this flexibility, it's possible to accommodate a myriad of use cases while at the same time controlling the complexity and cost of deploying and operating a sidechain. Keeping to this principle, the design aims to ease adoption of sidechains as tools for solving business problems by sharing core functionality and a novel way of solving particular problems.

Permissionless extensibility

Fundamentally, the concept of sidechains needs to enable extensibility of Cardano with new features. Simultaneously, it's paramount that creating and running any sidechain is not stifled by any central authority or group. To that end, it needs to leverage tools widely available to the community and focus on empowering it to create as many and as diverse sidechains as possible.

Bootstrapped security and community

At the core of the sidechain design lies the relationship between main chain and sidechain. The obvious benefit of using a well-established set of design choices is the reduction of friction when setting up a sidechain. One could describe it as a "batteries included solution" that allows quick bootstrapping of a sidechain. A perhaps less obvious benefit is the participation of the Cardano community. For the sidechain to be useful, it needs to be easy for Cardano users to benefit from and contribute to the proliferation of new chains that will become available.

Usefulness of Cardano sidechains

Why is the Cardano sidechain design considered useful? The proposed sidechain interaction pattern is put in front of the community as a recommended way to bring to the ecosystem new features in a secure, transparent, and trustworthy way. By leveraging Cardano security, this design engages a great asset that the Cardano ecosystem has available and makes it easier to start something new. This in turn strengthens the Cardano community as it permissionlessly enables new use cases, has the capacity to usher in new users, scales the ecosystem, and enables the democratization of cryptocurrency technology to the real world use cases that deliver a new global financial operating system.

Key Cardano sidechain building blocks

The design of Cardano sidechains on a very high level can be expressed with the following key concepts:

- Dependency on the main chain
- Ferrying messages implemented by different communication channels (incoming and outgoing)
- Sidechain-specific ledger rules

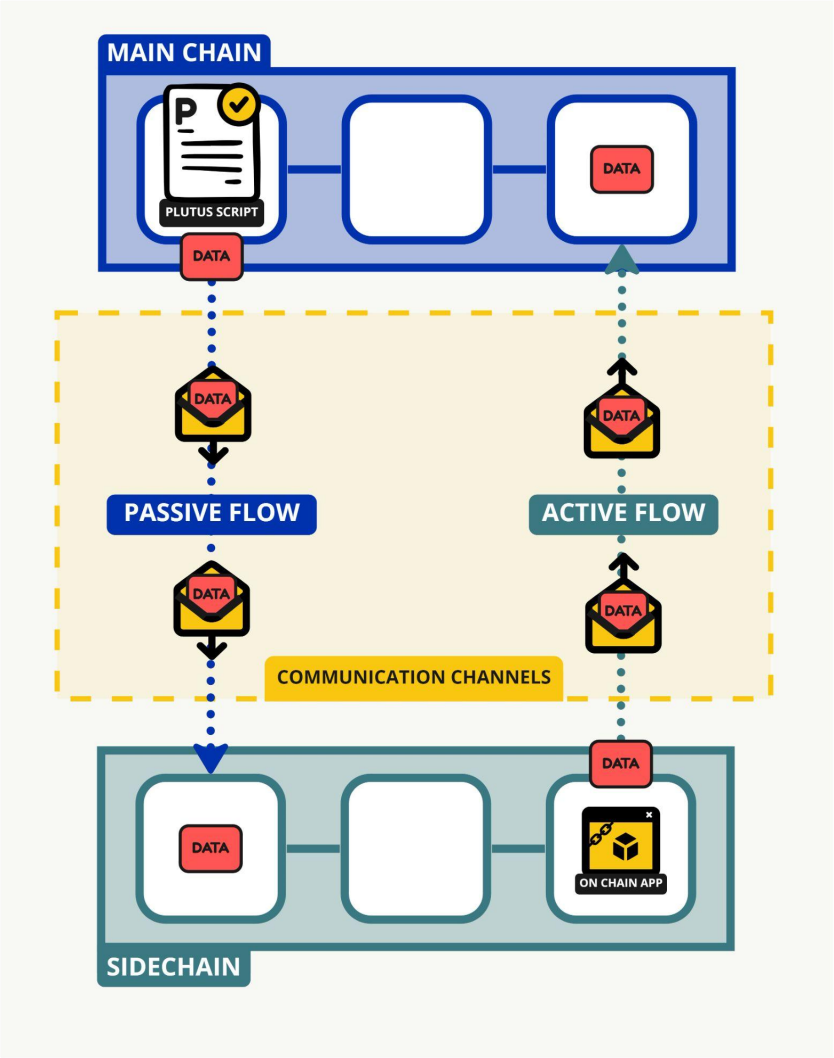

Fig. 3. High-level data flow focusing on directionality

The sidechain communication pattern focuses on enablement without disruption. Every functionality that has been leveraged is widely available and does not require any special adjustments in Cardano.

There are two fundamental flows that enable the sidechain communication pattern:

- ***Passive flow***: handles **incoming data** from the main chain to the sidechain and the main chain acts as the **root of trust** (see What is the 'root of trust'?).
- ***Active flow***: handles **outgoing data** from the sidechain to the main chain and the main chain acts as a trusted settlement layer (see Sidechain Certificate Creator).

Every other interaction pattern covered in this specification boils down to some combination of these two data flows.

Dependency on the main chain

As mentioned above, one of the core assumptions of the whole sidechains proposal is that it is possible to leverage the main chain as the root of trust. This observation is key to keep in mind when designing and building such lopsided solutions: one of the chains relies fully on the other's characteristics. The downside of this is that if the main chain fails, *all* sidechains fail. The upside is that if a sidechain fails, only that one sidechain fails, without affecting the main chain or any other sidechain.

Ferrying messages

Messages moving between the chains can be grouped by the direction in which they flow.

1. On one end, the sidechain needs to observe the main chain and react to discovered events. This **passive flow** is supported by a chain follower and a sidechain module (more on these in Section 2). As long as the main chain is considered the root of trust, any sidechain can build up its own internal mechanics based on trustable data, eg:
    - Staked ada distribution: how much ada delegated to a particular SPO.
    - Main chain epoch nonce: the seed of randomness in a main chain epoch.
    - Attestment of a token transfer: the documented data and value transfer requests from main chain to sidechain.
    - Committee candidates: list of SPOs registered as candidates to participate in a sidechain consensus protocol.

2. On the other end, the sidechain may need to take effect on the main chain (the **active flow**). To support this use case, two components are introduced: Sidechain Certificate Creator and Sidechain Certificate Validator (more on these in Section 3). These components are responsible for the proper creation of certificates attesting that some data exists on the sidechain. These certificates are leveraged to verify data on the main chain. Possible data being moved to main chain to leverage the trusted settlement layer are:
    - Committee rotation: keys of the next sidechain committee signed by the current sidechain committee
    - Transferred tokens: proof of transactions moving value from sidechain to main chain
    - Sidechain checkpoint: a known finalized block height in the sidechain

Sidechain-specific ledger rules

As mentioned above, the sidechain is the part of the communication pattern that accommodates the main chain. In other words, whatever mechanisms need to be put in place to express the desired outcomes for a particular use case, the sidechain will usually be the area where modifications and adjustments are made. These adaptations will be covered in detail in Section 2, 3, and 5).

Example rules that need to be supported:

- Main chain to sidechain token transfer rule: how to enact change on the sidechain when the chain follower reports a token transfer transaction.
- End of main chain epoch rule: what steps to take to prepare for a new main chain epoch (new nonce, new candidates, etc.).

- Block verification rule: what invariants based on data coming from the chain follower need to be true for a block to be verified.

Summary

In this section, Cardano has been established as a candidate for the main chain for a whole family of sidechains. The notion of a sidechain as a communication pattern has been introduced and a high level description of how to interpret foreign events has been given.

2 Main chain

This section focuses on the role of the main chain and the adaptations/components needed for building a Cardano sidechain. Beyond that, it covers the abstract interaction patterns between the main chain and sidechain, and dives deeper into the actual flows needed to support an example EVM sidechain.

What is the 'root of trust'?

The relationship between main chain and sidechain relies on establishing that one of the participating chains serves as the 'root of trust'. On a high level, this translates to an accessible and trustworthy source of data, and any chain can serve such a purpose as long as certain expectations are met:

- Finality guarantees: the main chain presents some observable finality of block settlement, that represents how long it takes for a particular datum to be effectively immutable.
- Ability to represent foreign data: the main chain needs to have on-chain facilities able to represent sidechain-specific data (registrations, certificates, etc.).
- Ability to run custom on-chain applications: similarly to foreign data representation, the main chain needs to have facilities allowing the coding of arbitrary logic supporting sidechains.
- Ability to follow the main chain: the main chain needs to be externally observable, either directly or via third-party tooling.

When selecting Cardano as the 'root of trust' for Cardano sidechains, the following characteristics were also taken into account:

- Proof-of-stake consensus protocol: ties into the commitment to minimizing the environmental impact of proposed solutions.
- Number of block creator pools: the number of block creators affects the chain's security by lowering the probability of collusion between participants.
- Number of active stakers: it is also a factor when considering security of the chain, perhaps in a less straightforward way. Active stakers move their stake to pools that give them the best return on investment (ROI), effectively shaping how the staking pools have to operate and adapt.
- Total value staked: sheer numbers do not guarantee as much as numbers backed by real monetary value. This again has an impact on security and trustworthiness of the chain even if the underlying system does not penalize misbehavior.
- Cost of interacting with the chain: how much it costs to operate a sidechain, and what currency is this cost settled in.
- Availability of additional data: source of randomness, source of time (epoch cadence).

Cardano as the 'root of trust'

The Cardano mainnet has the following characteristics:

- Finality is determined by the *securityParam* [6] (value is 2,160),
- It is capable of representing foreign data via *native tokens* [7],
- It is capable of running on-chain applications via Plutus scripts [8],
- There is a way to follow the chain state via applications like Cardano db-sync [9].

The network is operated by a group of staking pools responsible for creating blocks, and are rewarded for this effort [10]. End users have the ability to stake their ada with any pool of their choosing. This is achieved via liquid staking [11]. The larger the group of unique entities operating the chain, the lower the probability of malicious actors having impact on the blockchain state. At the time of writing (January 2023), Cardano mainnet has:

- Over 3,200 SPOs, with over 2,200 distinct entities running the blockchain.
- Over 3.5 million staking addresses.
- Over 25 billion ada staked,
- A native token *burn* transaction cost of approximately 0.4 ada.
- Posting a *claim* transaction cost of approximately 2.5 ada.
- Easily accessible epoch nonce epoch cadence to support such use cases as a Cardano sidechain.

Cardano is considered the 'root of trust' for Cardano sidechains due to the large group of entities running the chain and the high amount of ada staked by a large number of participants (the Cardano users). This means that if a sidechain observes incoming data that has finalized on the main chain, there is a miniscule probability of this data ever changing. This observation allows the sidechain protocol to be defined in terms of stable data on the main chain (settled, finalized).

Sidechain interactions from main chain perspective

Chains performing a sidechain role can achieve new characteristics by leveraging the trust established on their main chain. The interaction patterns established for the purpose of supporting an example EVM sidechain are as follows:

- Registration & de-registration of Cardano SPOs as block-producing committee candidates
    - A data dependency between main chain and sidechain where the former informs the sidechain which entities are going to run its consensus protocol

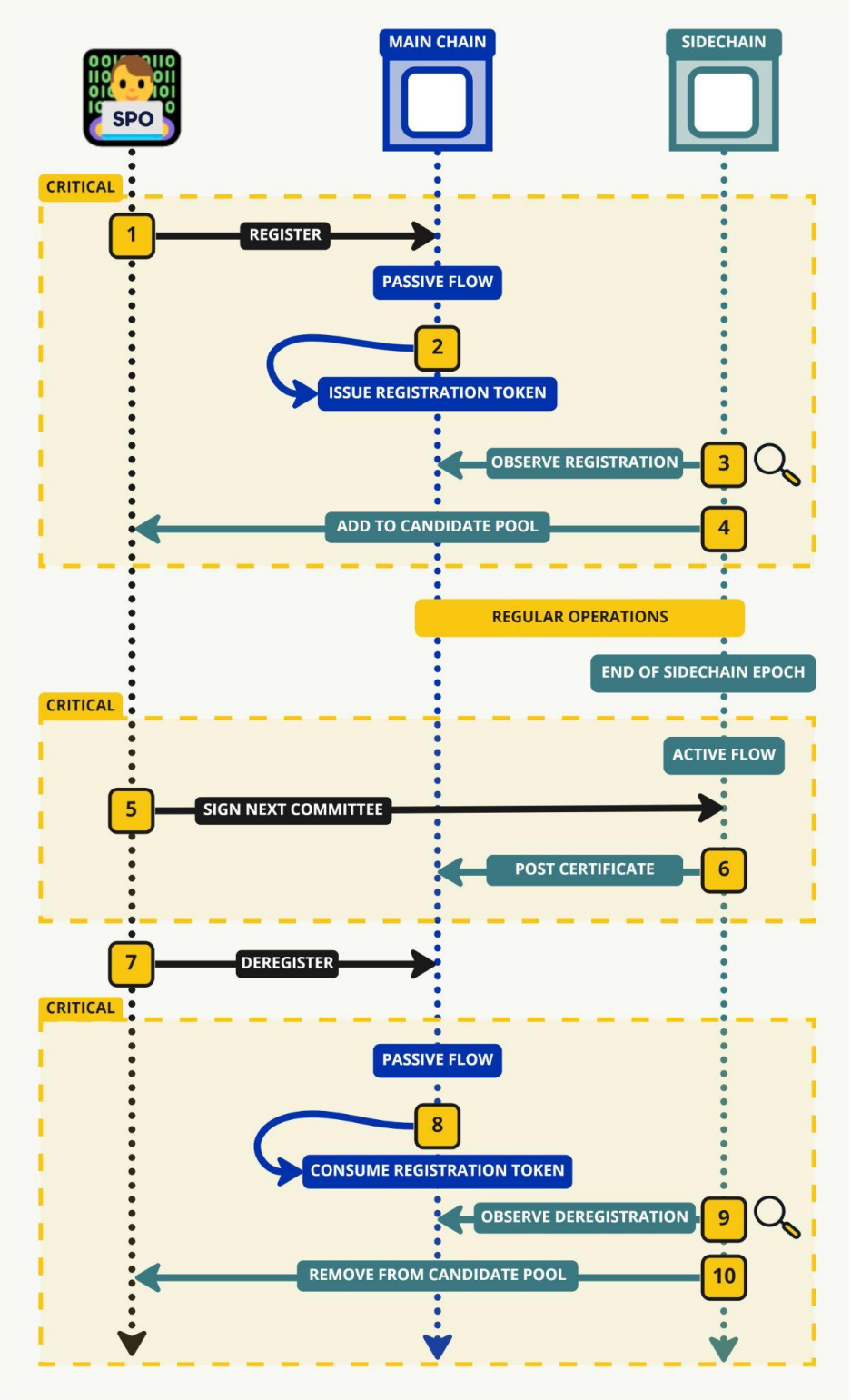

Fig. 4. Registration & de-registration flow sequence diagram

- It can be broken down into passive and active flow components: observing of (de-)registrations and committing certificates of the next committee.

- The ability to register as a sidechain block producer candidate contributes to a sidechain's behavior: it enables block producing committee rotation and facilitates dynamic participation in the sidechain.

- Moving tokens from the main chain to the sidechain
  - A data dependency and interaction pattern between main chain and sidechain where an established flow (actions resulting in data on both chains) results in value being moved from main chain to sidechain in a trusted way.

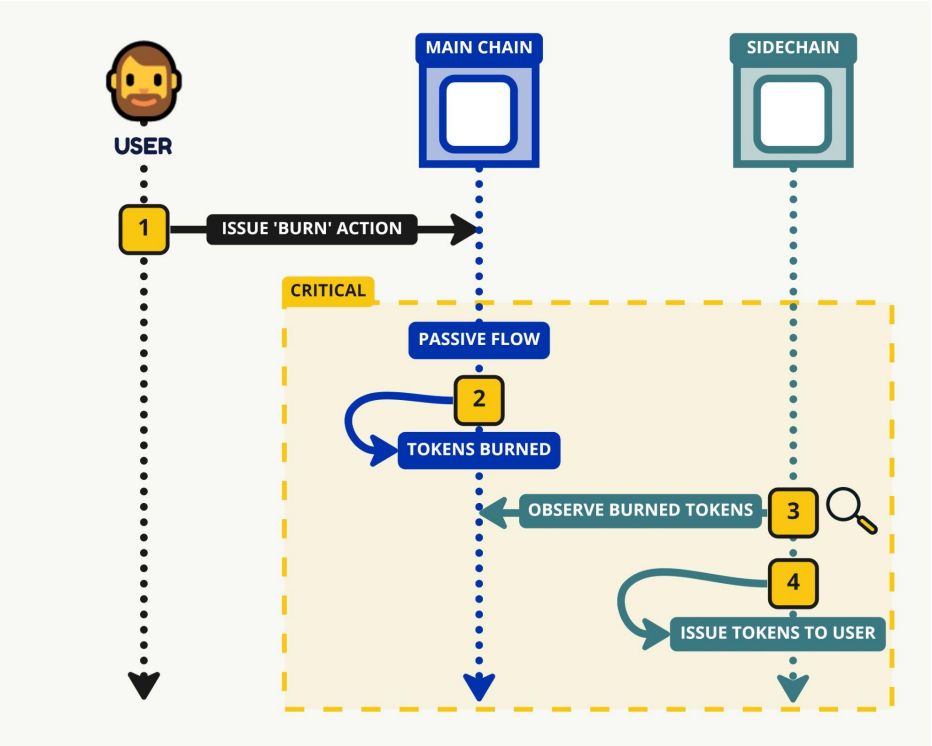

Fig. 5. From main chain to sidechain token move flow sequence diagram

  - It is solely a passive flow: an action on the main chain results in state change on the sidechain.

- Moving tokens from the sidechain to the main chain
  - A data dependency and interaction pattern between sidechain and main chain where an established flow (actions resulting in data on both chains) results in value being moved from sidechain to main chain in a trusted way.

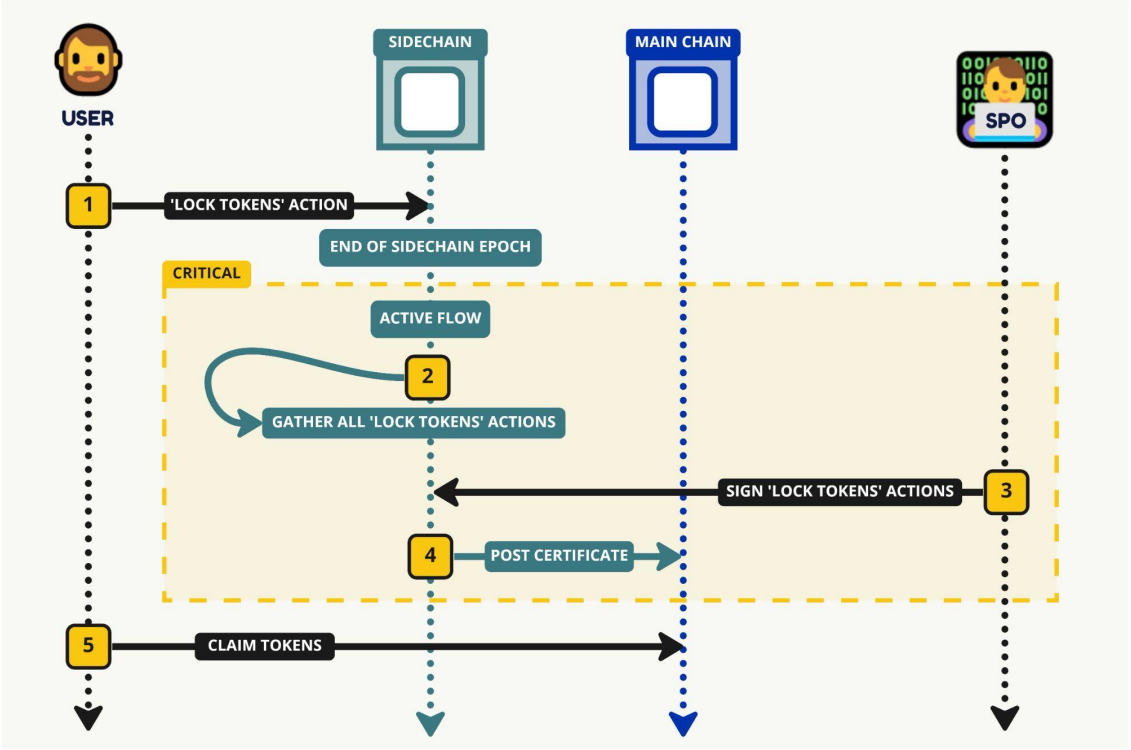

Fig. 6. From sidechain to main chain token move flow sequence diagram

- It is solely an active flow: an action of the sidechain results in data becoming available on the main chain.

The fundamental concern mentioned in Section 1 is focused solely on the direction of the data (to sidechain, passive flow; to main chain, active flow). First, a detailed description of the generalized flows will be given. Subsequent sections will cover flows that enable an example EVM sidechain:

- Registration of Cardano SPOs as block-producing candidates.
- De-registration of Cardano SPOs as block-producing candidates.
- Moving tokens from the main chain to the sidechain.
- Moving tokens form the sidechain to the main chain.

Passive flow from the main chain's perspective

A generalized passive flow describes how to move any piece of data from the main chain to the sidechain. At the core of the passive flow there is a fundamental assumption that a sidechain can process incoming data from the main chain and it is possible to make that data available in a trustworthy way. To enable the flow, a passive reading mechanism is needed. The purpose of this mechanism is to read and transform data available on the main chain and make it available on a sidechain. Its focus is on tracking on-chain data that allows the operation of a particular sidechain (eg, transactions from main chain to sidechain).

It consists of two components:

- Chain follower: a general purpose component capable of reading and indexing main chain events.
- Sidechain module: a dedicated component of a sidechain capable of translating main chain events (reported by the chain follower) into sidechain events.

This mechanism needs to be able to follow the current state and also allow traversal of historical data.

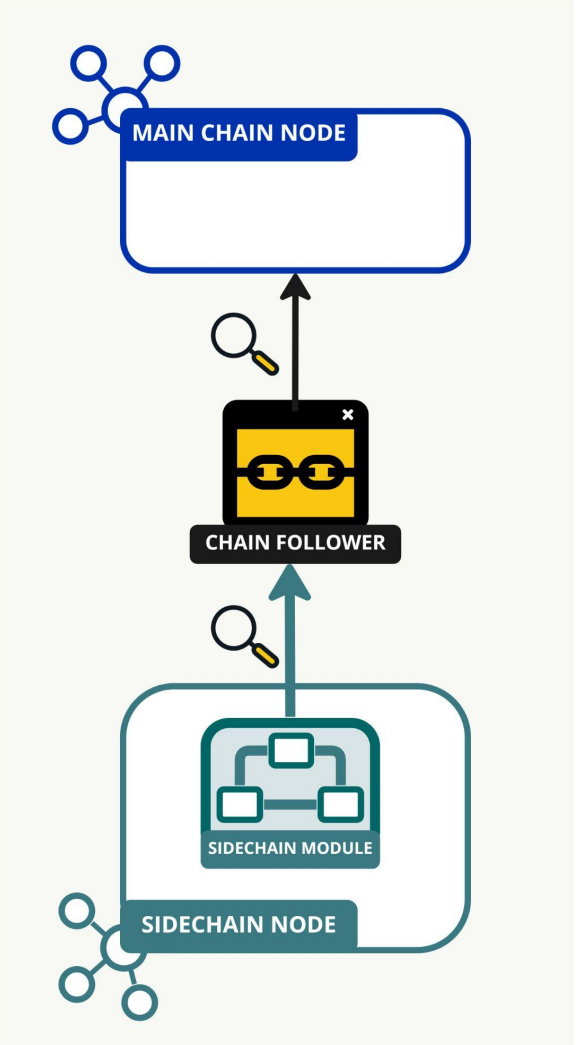

Fig. 7. Components participating in the passive flow

In general, any passive flow requires an interaction point. On the main chain, this is served by purpose-built on-chain scripts that support a particular passive flow. From a low-level on-chain behavior's perspective, these scripts produce UTXOs that are used as markers of particular events and are tracked by the sidechain's chain follower component. The sidechain module can transform these on-mainchain events and take action on the sidechain (more on this is Section 3).

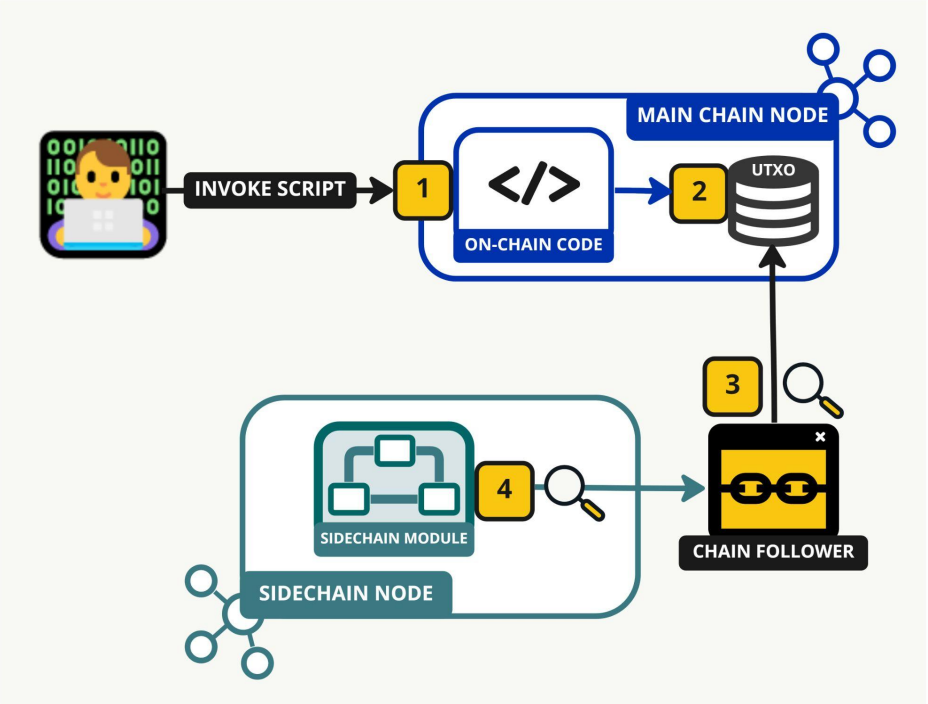

Fig. 8. An interaction diagram of a generalized passive flow

The passive flow from the perspective of the main chain:

1. An actor takes action on the main chain (interacts with some on-chain state via an on-chain script).
2. The interaction results in some new state that is captured as a UTXO on the main chain
3. The chain follower observes this new state and takes note of it.
4. The sidechain module notices the new state (main chain event) indexed by the chain follower and processes it according to the particular passive flow logic. This step is subject to additional limitations that are sidechain-specific. The example EVM sidechain assumes that the data observed by the sidechain module needs to be settled on the main chain before it is introduced to the sidechain.

Active flow from the main chain's perspective

A generalized active flow describes the actions and data dependencies required when moving data from sidechain to main chain. The key data dependency is a certificate built by the sidechain's block-producing committee, which can be independently verified on the main chain. To create this certificate, a protocol needs to be followed: the sidechain committee members need to agree on a particular blockchain state, and must then provide signatures attesting to that shared state and make those signatures available on the main chain in the form of a certificate. This is the high-level description of the certificate-based observation of a chain described in Section 1.

This flow consists of three components:

- Sidechain Certificate Creator: on a very high level, it is a set of rules on how to create a certificate proving that a particular event was settled on the sidechain. In practice, it's an on-sidechain component that implements the rules of certificate creation. It governs the cadence and participation of block-producing committee members.

- Sidechain Certificate Validator: an on-mainchain component that implements the same rules of certificate creation as the Creator, but with the distinct purpose of verifying certificates.
- Relay actor: a component that automates ferrying certificates from the sidechain to the main chain. Its sole purpose is to make certificates created on the sidechain available on the main chain. It does not participate in guaranteeing their authenticity.

To support active flows, a set of on-mainchain scripts (Sidechain Certificate Validator) is required. The generalized flow depends on purpose-built Sidechain Certificate Creators capable of noticing, and providing proof of, on-sidechain events for the main chain to process and verify (this will be discussed in detail in Section 3). The system depends on a relay actor to shuttle these proofs as transactions from the sidechain to the main chain. When the proof is available on the main chain, a set of on-chain scripts can be engaged to verify a particular sidechain event.

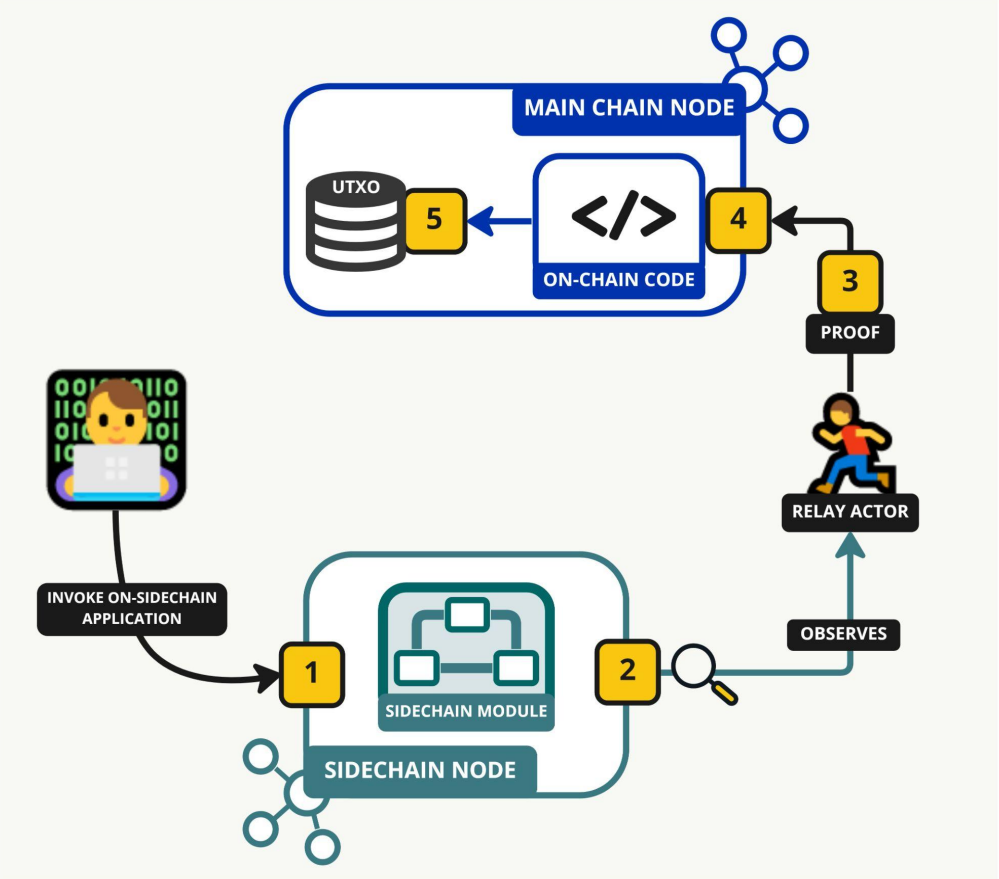

Fig. 9. An interaction diagram of a generalized active flow

The active flow from the perspective of the main chain:

1. The sidechain goes through the protocol of creating a certificate (detailed in Section 3) proving an on-sidechain event **E** (result of a user's interaction with the sidechain with the goal of moving data to the main chain).
2. A relay actor observes the creation of an on-sidechain certificate.
3. A relay actor engages an on-mainchain script and provides a certificate as proof of an on-sidechain event **E.**
4. The on-mainchain script verifies the posted proof and stores it on-chain as a UTXO.
5. This proof can be later referenced or challenged as part of other flows.

Example EVM sidechain flows from the main chain's perspective

Having the two generalized flows (passive and active) defined, it's possible to describe specific flows supporting an example EVM sidechain.

Main chain flow: registration of Cardano SPOs as block-producing candidates

This flow supports an example EVM sidechain in allowing block producing committee members to be opt-in and also supports the committee's rotation.

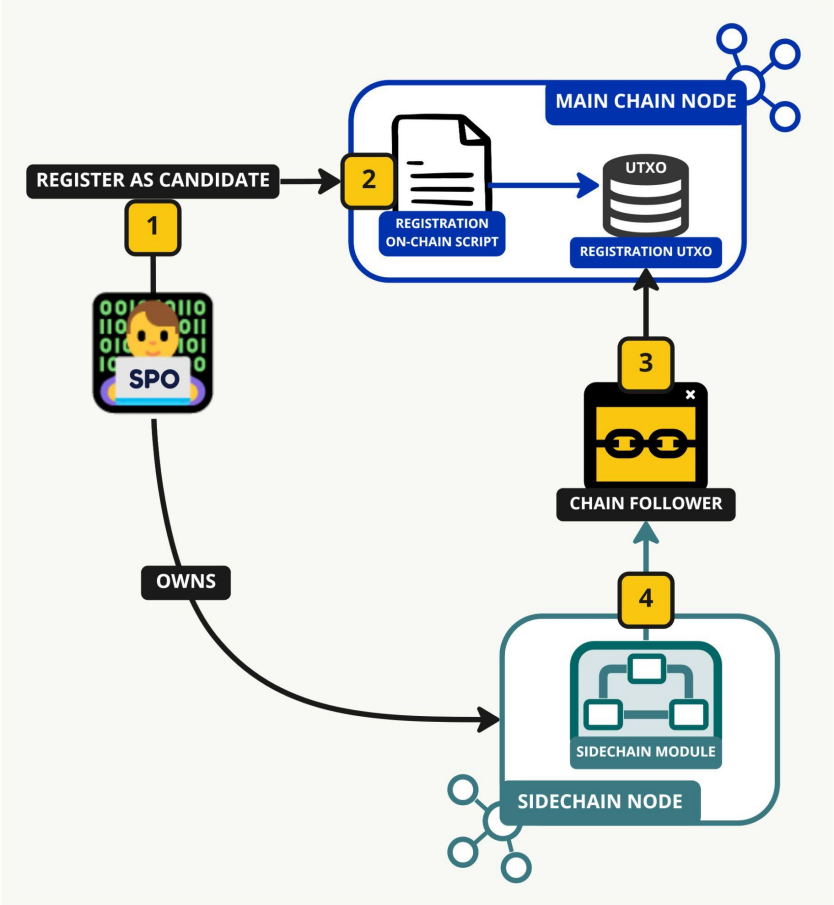

Fig. 10. An interaction diagram of registration flow

The flow of registration of block-producing candidates from the perspective of the main chain is as follows:

1. An SPO prepares messages proving that they are an owner of keys on the main chain and pointing at keys for the sidechain. These messages are inputs for the registration's on-chain script.
2. The SPO invokes the sidechain candidate registration script. If successful, a sidechain registration UTXO is created. This UTXO holds the sidechain's public key of this candidate.
3. The chain follower component discovers the new UTXO created by the registration script. This data becomes part of the main chain state represented in the chain follower.
4. The sidechain module notices that a new registration was recorded by the chain follower. This data is now available to the sidechain and will be taken into consideration when selecting a sidechain block-producing committee, starting from the next Cardano epoch.

Main chain flow: de-registration of Cardano SPOs as block-producing candidates

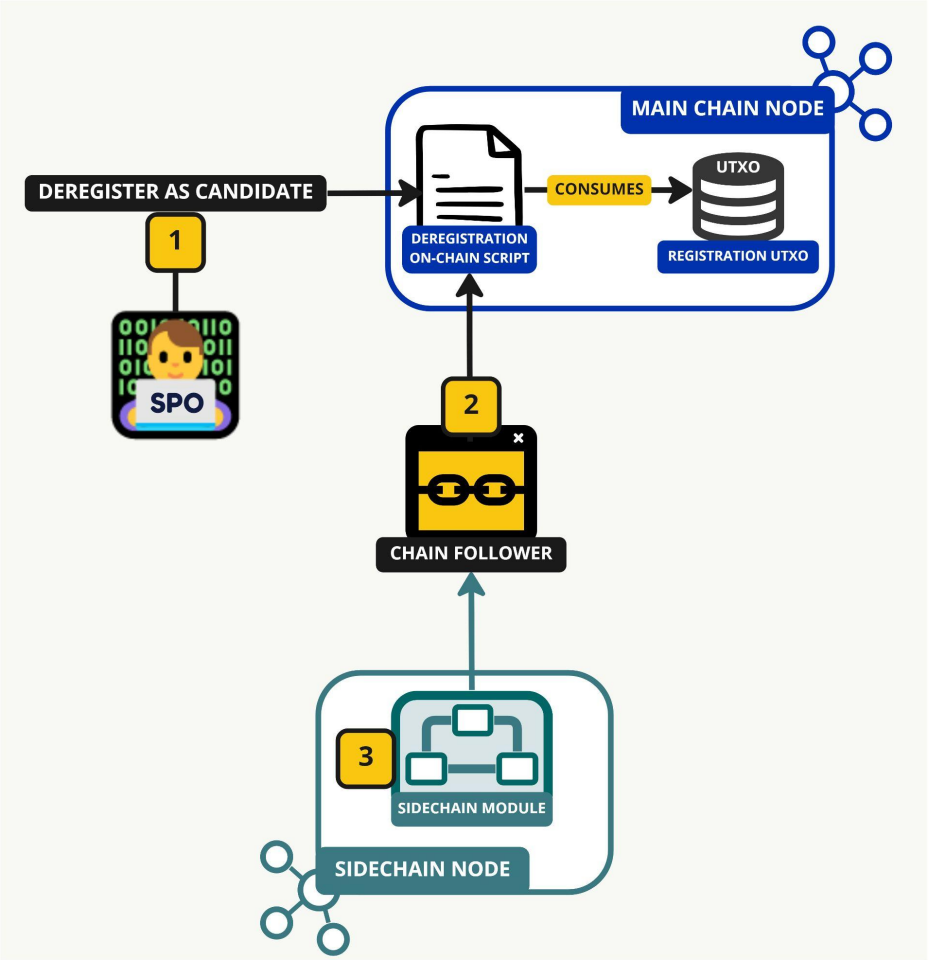

Fig. 11. An interaction diagram of de-registration flow

The flow of de-registration of block-producing candidates from the perspective of main chain is as follows:

1. A registered SPO invokes the sidechain de-registration script with its registration UTXO. If successful, the passed token is consumed and is no longer part of the main chain state.
2. The chain follower component discovers a UTXO being consumed. This event is persisted in the main chain state representation in the chain follower.
3. On the sidechain, the sidechain module notices that the de-registration event (UTXO being no longer available) was recorded by the chain follower. This fact will remove the SPO from the block producing committee selection process..

Main chain flow: moving tokens from the main chain to the sidechain

This flow supports an example EVM sidechain in allowing a user to move tokens from the main chain to the sidechain in a trustworthy manner. As a prerequisite, every sidechain requires a dedicated (sidechain specific) on-mainchain minting policy that governs that sidechain's native token.

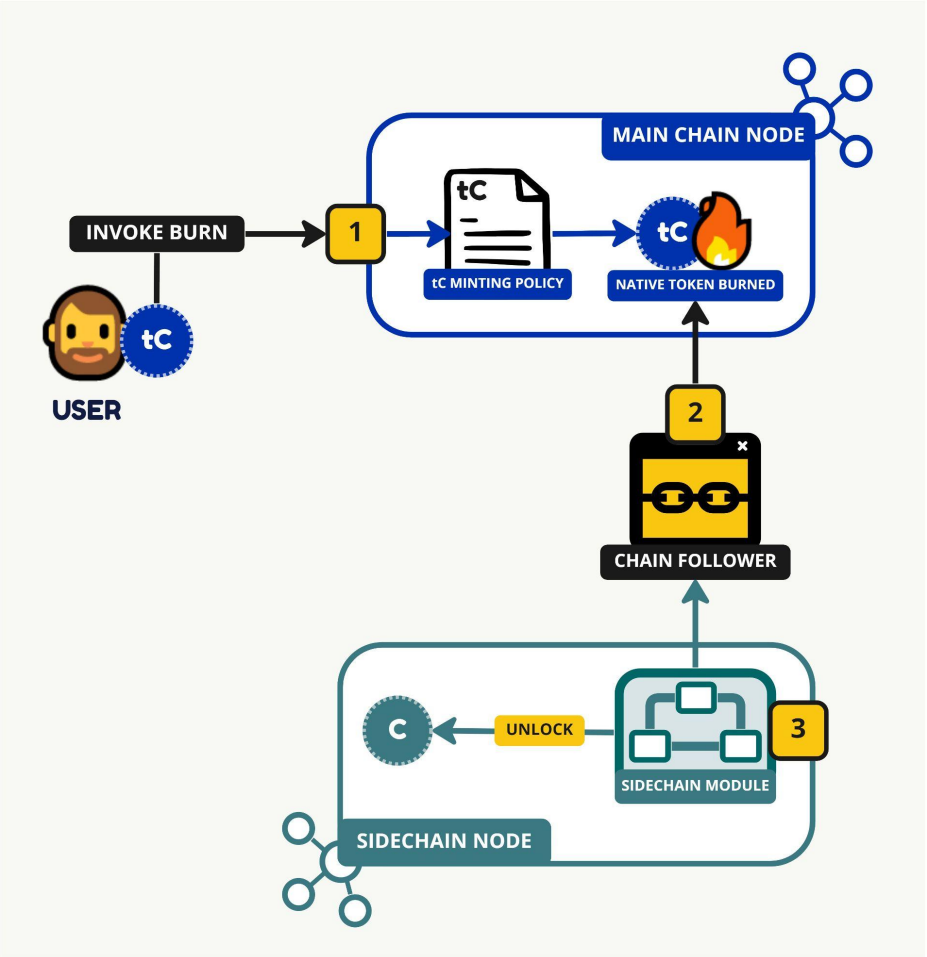

Fig. 12. A interaction diagram of moving tokens from main chain to sidechain

The flow of moving tokens to the sidechain from the perspective of main chain is as follows:

1. A user invokes the *burn* action of the sidechain's native token minting policy, attaching metadata with the intended sidechain address. If successful, a burned native token event is processed by the main chain and the UTXO holding it is consumed. The intended sidechain address is captured on the main chain as part of the burn event.
2. The chain follower component discovers the consumed UTXO and persists this event with metadata in its main chain state representation.
3. On the sidechain, the sidechain module notices that the burn event was recorded by the chain follower. This fact can be taken into account when creating a new block by a block-producing node.

Main chain flow: moving tokens from the sidechain to the main chain

This flow supports an example EVM sidechain in allowing a user to move tokens from the sidechain to the main chain in a trustworthy manner. As a prerequisite, every sidechain requires a dedicated (sidechain specific) on-mainchain minting policy that governs that sidechain's native token.

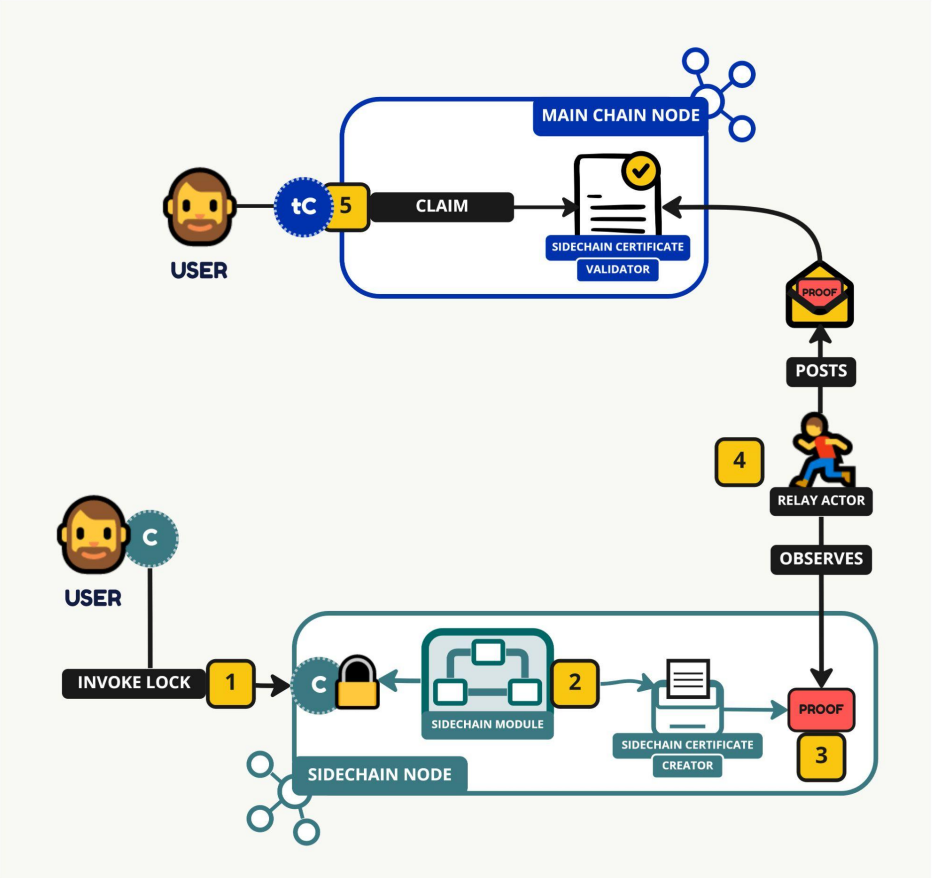

Fig. 13. A interaction diagram of moving tokens from sidechain to main chain

Actions that the main chain is unaware of when moving tokens to the main chain:

1. A user locks an amount of tokens utilizing the sidechain module (this is an oversimplification and is explored in Section 3).
2. At some point, the sidechain module decides to invoke the Sidechain Certificate Creator to build the proof of a locked amount.
3. This proof is signed by multiple committee members and it settles on the sidechain.

The flow of moving tokens to the main chain from the perspective of the main chain is as follows:

4. A relay actor notices this proof and posts it to the main chain's Sidechain Certificate Validator script. If successful, this script will produce a UTXO that holds the proof.
5. A user invokes the claim sidechain token script on the main chain. As long as the user's claim and proof match up, the user is granted a native token B on the main chain.

Summary

This section covered the role of the main chain and why Cardano is a good candidate to fill this role. Additionally, the notion of 'root of trust' has been introduced. Key flows that govern the sidechain from the perspective of the main chain were also introduced and detailed in the context of an example EVM sidechain.

3 Sidechain

This section focuses on the requirements and adjustments needed for the sidechain use case. Similarly to Section 2, it introduces the two abstract flows (passive and active) from the sidechain's perspective and then goes deeper into the flows that support an example EVM sidechain:

- Registration & de-registration of Cardano SPOs as block-producing candidates.
- Moving tokens from the main chain to the sidechain.
- Moving tokens from the sidechain to the main chain.

What is the 'root of trust' according to the sidechain?

From the sidechain's point of view, the main chain is nothing more than a distributed database with eventual consistency (a trusted oracle with settlement time dictated by the security parameter of the main chain). The points of interest are data that manage the sidechain and can be discovered by directly following the main chain. Therefore, the 'root of trust' is a fundamental assumption that the main chain consensus algorithm is sound, and when given adequate time, the data of interest will become immutable.

In the blockchain space, one of the greatest challenges is to get and maintain enough traction for the blockchain to be trustworthy and not susceptible to malicious activity. Having a 'root of trust', and also being able to adjust the reliance on it based on the specific use case, offers an enormous amount of flexibility. Of course, this comes with its own challenges and costs: how does a particular design affect security, transactions per second, cross-chain transactions per second, etc. But the bedrock is established: fundamentally, there is a dependency pattern between a main chain and a sidechain that does not require changes to the main chain, does not introduce a third trusted committee and, at the very least, allows for reading of state from the main chain. This also means that there is a clear lopsided dependency between the two chains: the sidechain cannot operate without being able to track its main chain.

To facilitate potential reuse, the team began by abstracting components and dependencies so that at some point, creating a sidechain could be as simple as writing down custom ledger rules in some high level domain-specific language (DSL) and having all the dependencies already in place.That's the goal, the north star of the Cardano sidechain effort. These are the current areas of focus:

- Chain follower: as mentioned in Sections 1 and 2, this is the read-enabling component.
- Sidechain module: as mentioned in Sections 1 and 2, this represents the sidechain-specific interpretation of the data coming from the chain follower.
- Sidechain Certificate Creator: the component responsible for building and storing on-sidechain proof for a particular on-sidechain event.
- Relay actor: the actor capable of enacting change on the main chain. A high-level description of this actor is provided, though the details of implementation are not covered by this technical specification.

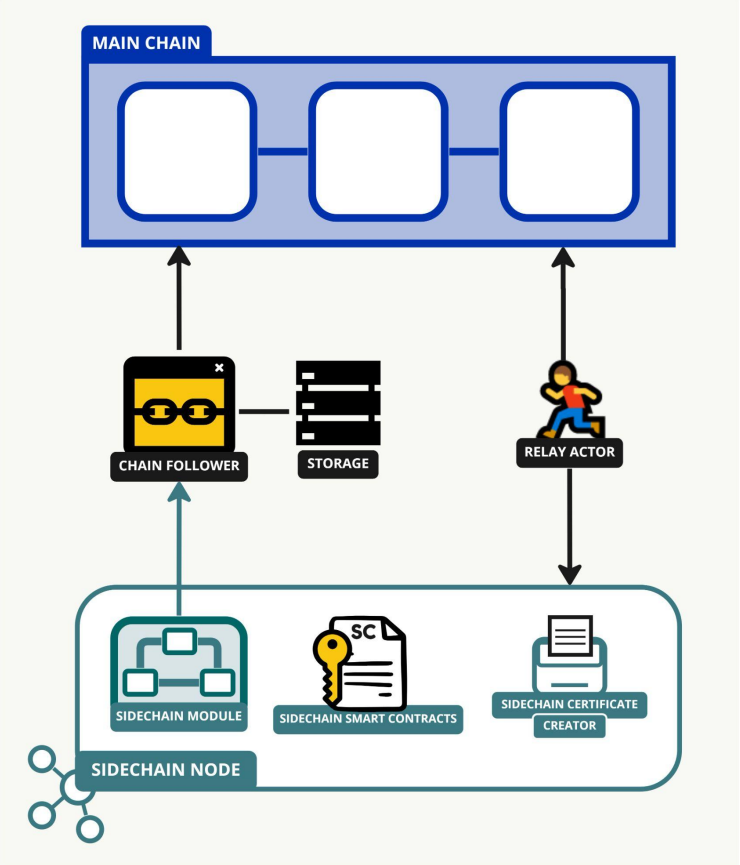

Fig 14. An interaction diagram of a sidechain node

Chain follower

The chain follower's sole duty is to understand main chain events. It can be part of a sidechain node, but it can also be an external dependency. It enables the passive flow of data from the main chain to a sidechain. A chain follower is a general purpose component that does not need sidechain-specific logic. The design assumes that any node that is going to validate sidechain blocks needs access to a synchronized, trusted instance of a chain follower. It is considered a 'read-only' data source.
Example data that can be tracked to support a sidechain:

- Main chain epoch metadata: utilized as the basis for managing sidechain epochs (if applicable).
- Stake pool delegation distribution: utilized as the weight of SPOs participating in running the sidechain consensus protocol (if applicable).
- SPO registrations: utilized to identify eligible participants of the sidechain consensus protocol (if applicable).
- Native token distribution: utilized to identify the weight and participation of eligible participants in the sidechain consensus protocol (if applicable).

Sidechain module

The sidechain module abstracts the low-level data gathered by the chain follower. The module filters, transforms, and serves data to the sidechain node's components. It is also an adequate

place to implement sidechain-specific ledger rules. Additionally, it is the responsibility of the sidechain module to handle “time to finality”. The sidechain module observes data coming from the main chain as soon as it’s available in the chain follower. This means that the data might not be finalized on the main chain yet. It is up to the sidechain module logic to potentially compensate for this fact. If the sidechain assumptions require that it operates on finalized data, the sidechain module needs to be able to handle the finalization period and serve adequate data to the downstream components when applicable (finalized). The sidechain module can either be part of a sidechain node or an external dependency.

Example data and transformations handled by the sidechain module:

- Calculate the next committee members: if a sidechain requires committee member rotation, the sidechain module would abstract the details of candidate discovery away from a module handling consensus.
- Notify end of epoch: if a sidechain has rules triggered at the end of epoch, the sidechain module needs to be able to track and notify consensus and ledger modules in the node.
- Gather transactions that need to be handled on the sidechain: if the sidechain supports incoming transactions from the main chain, the sidechain module needs to adapt them and make them available to the ledger module.
- Serve sidechain’s configuration: if the sidechain has any persisting configuration on the main chain, the sidechain module needs to be able to provide an up-to-date version to modules relying on it.

Sidechain Certificate Creator

The Certificate Creators’ responsibilities are:

- To accrue data on the sidechain that needs to be ferried to the main chain (hoard).
- To inform the governing committee (block producers) of the sidechain that it is time to certify the hoarded data (prove).
- To make the proof available for the ferrying mechanism (rely).

The above functions can be implemented in many ways. An example implementation is described in Section 5. The piece that needs to be well described and common between the sidechain and on-mainchain scripts is the proving step. In other words, the Sidechain Certificate Creator needs to be aligned with the Sidechain Certificate Validator living on the main chain). The proposed solution to this challenge is ad hoc threshold multi signatures (ATMS), which is described in detail in the “Proof-of-Stake Sidechains” paper [2]. In the following section, it is assumed that both the Creator and Validator are implementing the same version of an ATMS scheme. The paper covers the reasoning behind establishing trust between main chain and sidechain, but the short version of it is that as long as the committee running the sidechain is considered honest, the certificates produced by two thirds of it can be trusted. Therefore, taking action on the main chain based on proof provided by an honest majority is acceptable.

Note on trivial-ATMS: a trivial-ATMS or append-scheme or plain ATMS is a simple signature scheme implemented by concatenation of keys and signatures. For more details see [2].

Sidechain interactions from the sidechain’s perspective

In Section 2, the point of view of the main chain was covered in detail. The same flows need to be detailed from the perspective of the sidechain. First, the two generalized (passive and active) flows

will be covered. A short introduction to OBFT and its impact is provided, and deeper discussion of flows supporting the example EVM sidechain follows:

- Observing main chain data changes, in particular registrations of block-producing candidates on the main chain
- Rotating block-producing committee members
- Moving tokens from main chain to sidechain
- Moving tokens from sidechain to main chain

Passive flow from the sidechain's perspective

The passive flow from the perspective of the sidechain revolves around two core notions:

- Taking note of data on the main chain that affects the sidechain's operations but does not need to be persisted to the sidechain, see 'Observing main chain data changes' passive flow.
- Taking note of data on the main chain that needs to be included in the sidechain's ledger state, see 'Moving tokens from main chain to sidechain' passive flow.

Both of these data streams are handled by the chain follower (covered in Section 2) and sidechain module, They are based on observing the main chain and both result in changes to the sidechain. Furthermore, the flow that takes effect in the sidechain's state is a superset of the flow that only tracks main chain data. Taking advantage of this, the 'Observing main chain data changes' flow is explained first. The second one builds on the same ability to read data from the main chain and extends the flow with the ability to make data available in the sidechain ledger.

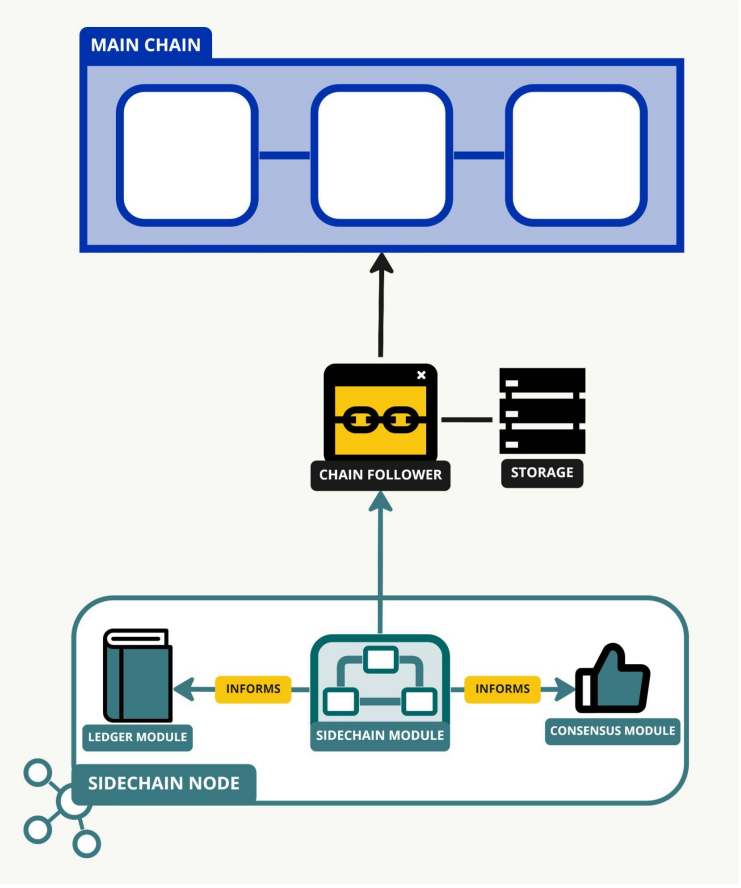

Fig. 15. An interaction diagram for the passive flow of the sidechain module

A generalized passive flow depends on the sidechain module discovering data of interest. The sidechain module needs to be able to guarantee 'at least once' delivery and needs to allow access to historical and current data points.

Active flow from the sidechain's perspective

The active flow has a singular purpose: allow the sidechain to enact change on the main chain without expecting any sidechain-specific adjustments or low level modifications in the main chain. At the core of the active flow, there are two mechanisms that need to be aligned:

- Sidechain Certificate Creator (on the sidechain): component responsible for hoarding and proving on-sidechain data.
- Sidechain Certificate Validator (on the main chain, on-chain code): component responsible for verifying data on the main chain.

The Creator is tasked with building a verifiable proof based on data available on the sidechain, while the Validator is responsible for verifying the posted proof with no detailed knowledge of the specifics of the sidechain, based on on-mainchain data alone.

In general, the proof-based active flow consists of the following steps:

- Accumulation of data for certifying (hoarding): whatever data of interest is supposed to be moved from the sidechain to the main chain, it needs to be gathered in a way that allows inspection and further processing.
- Gathering of signatures phase: the hoarded data needs to go through a process of signing. All block-producing members of the sidechain participate in this step and produce data that can be trusted.
- Creation of certificate: when enough of the signatures (related to ATMS) are available, any independent observer can build a certificate that can be posted to the main chain.
- Verification of certificate on main chain: a certificate needs to be verified on the main chain against the expected public keys of the current committee.

This flow can be used to certify any piece of data that needs to be moved to the main chain in a trustworthy manner.

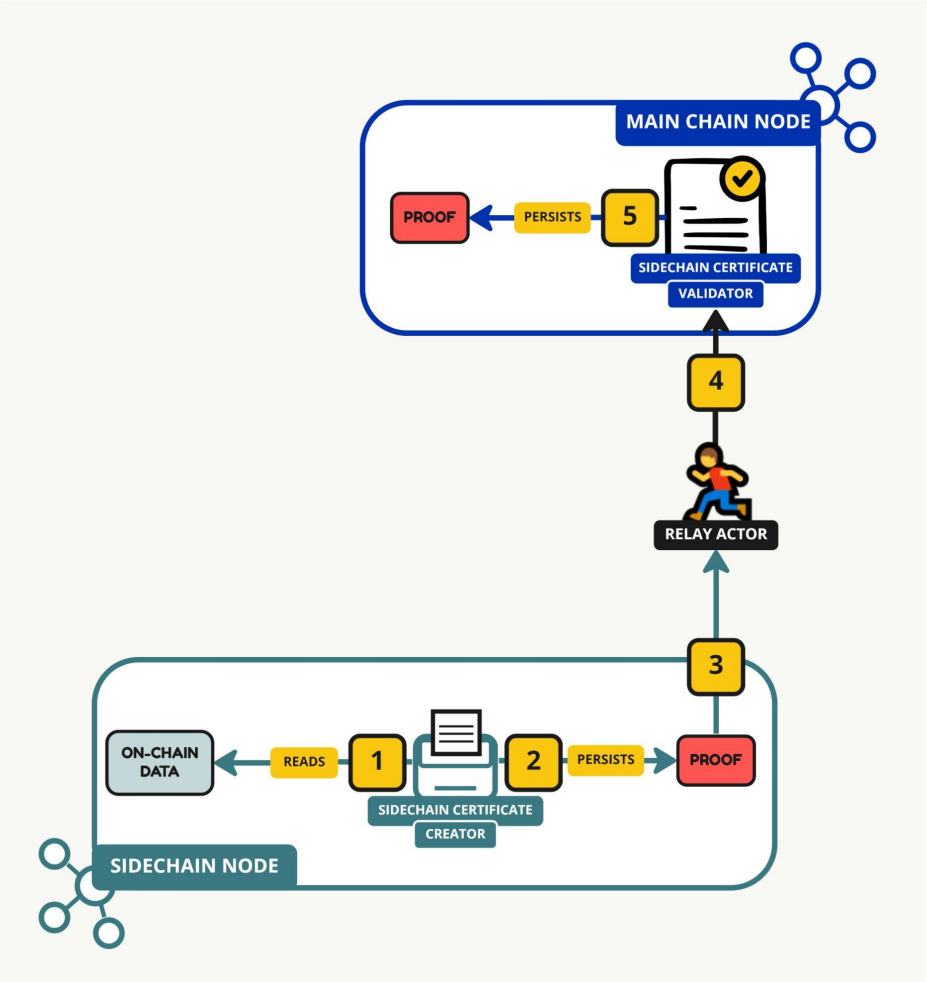

Fig. 16. A generalized interaction diagram of the active flow

1. The Creator discovers that a certain condition has been met and a particular set of data needs to be certified.
2. A new fact (proof) will be put into the ledger by each block-creating participant of the consensus protocol.
3. When enough participants (n out of m) create the required proof, it can be acted on by the relay actor in the form of a certificate (in case of trivial-ATMS, this means an ordered concatenation of signatures).
4. The relay actor invokes the Validator script, passing the crafted certificate to it.
5. If successful, the proof becomes persisted on the main chain.

Overview of key OBFT characteristics

A major part of making the mentioned flows work is an assumption that the sidechain can reach consensus in a reliable and predictable manner. Ouroboros BFT [1] has been selected as the first consensus protocol to run the example EVM sidechain, as it serves the reliability and predictability with the following features:

- Permissioned: the committee is of a finite size.
- Simple: the protocol has one phase and is easy to implement.

- Extensible: it was possible to add a committee rotation mechanism without disturbing the core mechanics of the protocol.
- Guaranteed finality: a particular ledger state becomes immutable after a set amount of blocks.

Building on these protocol features, all the flows covered in Sections 2 and 3 supporting an example EVM sidechain can be implemented. A detailed description of OBFT in the example EVM sidechain is covered in Section 5. This of course does not mean that OBFT is the only sidechain-friendly consensus protocol in existence. The key takeaway is the importance of guaranteeing finality and rotation of committee members as a basis for the passive and active flows.

Example EVM sidechain flows from the sidechain's perspective

Having the generalized flows established it's possible to go into details on flows supporting the example EVM sidechain.

Sidechain flow: observing main chain data changes

The first type of passive flow is the sidechain's operational parameters dependency on data coming in from the main chain. Events that have been identified in this flow are:

- Block-producing committee candidate list change
- Sidechain configuration change

In each of these cases, relevant modules need to be made aware of the change and adjust operations. Such changes will result in observable impact on the sidechain, but since such data is sourced from the main chain, there is no need to put it in the sidechain's ledger. This does not mean that there is no value in persisting it. For example, making the execution layer of a sidechain aware of main chain epochs can be useful, but it's not necessary.

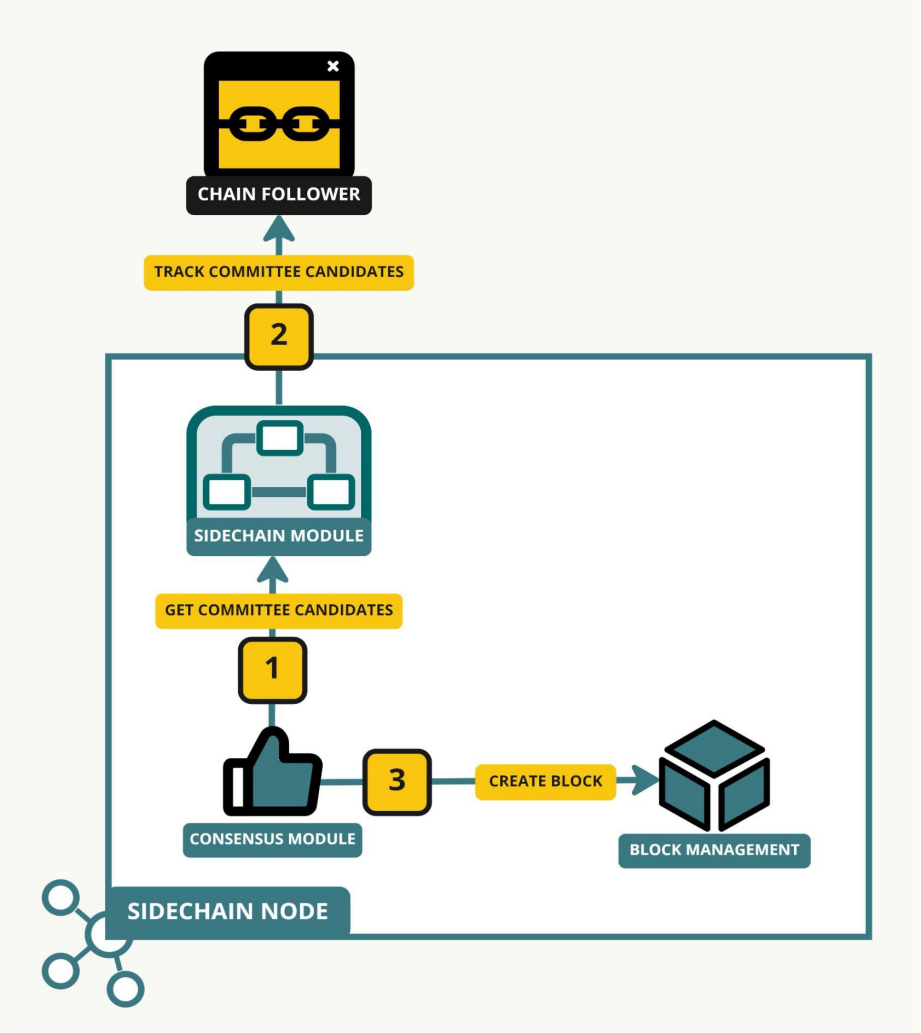

Fig. 17. An interaction diagram for inclusion of operational parameters of the sidechain

In the case of the example EVM sidechain, one of the flows dependent on the main chain data driving sidechain behavior without making its way to the sidechain's ledger is the process of block producer committee rotation. When the appropriate time is observed (end of sidechain epoch), a new committee is deterministically selected from the committee candidate pool. This pool is obtained by requesting data from the sidechain module (and in turn from the chain follower, which sources data from the main chain). A change to the block-producing committee will be observable by inspecting the blocks created, but this data does not need to be made available on the ledger. It follows that at least a sidechain-aware consensus module is needed to accommodate this flow. Since such information affects the consensus protocol, the protocol itself needs to be aware of the impact of main chain events. In the case of the example EVM sidechain, this presents as an adapted version of the OBFT consensus protocol (more on this in Section 5). Verification of validity of such events is built into the consensus protocol. Either a quorum agrees on the state observed on the main chain or it does not.

Sidechain flow: moving tokens from main chain to sidechain

A passive ledger modification flow follows the same steps as the Sidechain flow: observing main chain data changes flow up to the point where the data needs to be made available on the

sidechain's ledger. To allow this, a sidechain-aware ledger needs to be crafted so that data coming from the main chain can be put into the blockchain. The main chain event identified as ledger modifying is the Main chain flow: moving tokens from main chain to sidechain.

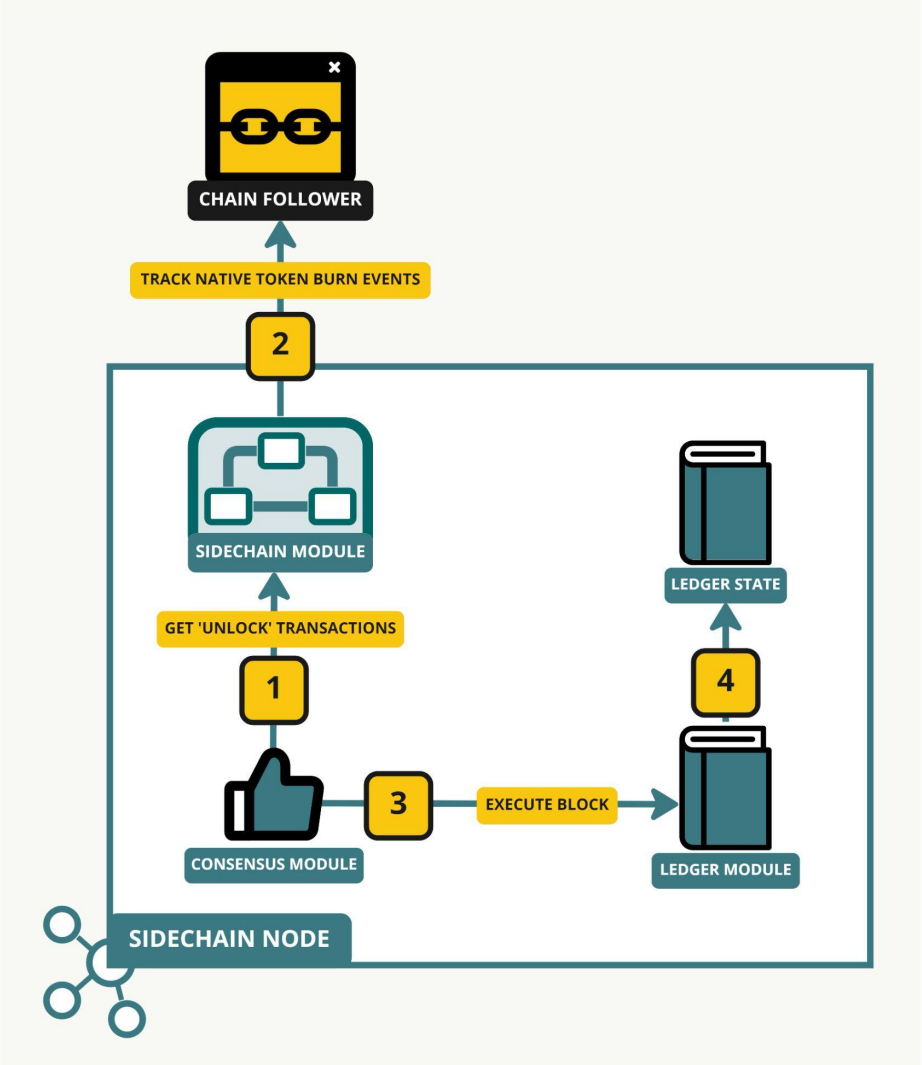

Fig. 18. A dependency diagram of introducing ledger modifications based on data from the main chain

In the case of the example EVM sidechain, the event coming from the main chain is a native token burn action.

1. The consensus algorithm identifies an appropriate time (block creation).
2. It requests a list of incoming token transactions from the main chain and includes them in the crafted block.
3. The block is executed.
4. New ledger state is produced.

This is achieved by introducing a special type of transaction that is allowed to unlock cross-chain funds on the sidechain. These transactions affect the state of the ledger. Some accounts will be awarded with tokens stemming from the main chain.

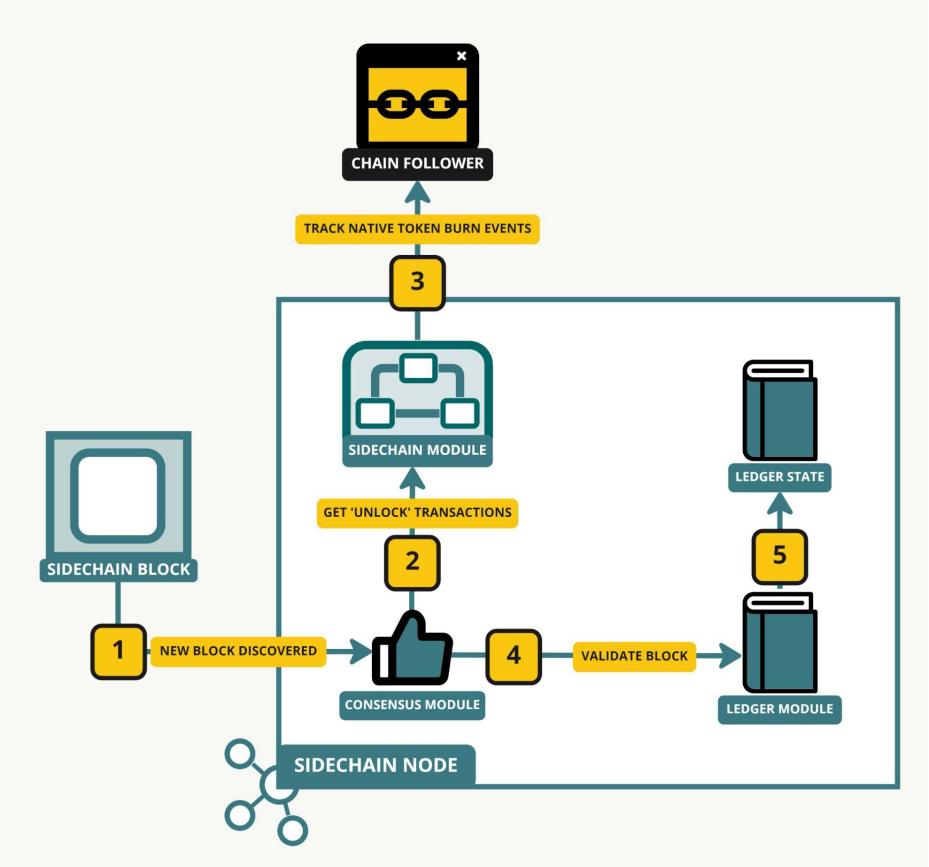

Fig. 19. Dependency diagram of validating a ledger modification based on data from the main chain

Such blocks will be broadcast to the whole chain. Part of the regular operations of a sidechain node include the verification and inclusion of blocks incoming from other block-producing committee members. To be able to verify the new ledger state, a validating entity needs to be able to observe the same incoming main chain events. Only in the case of successful validation of these transactions can a block be included in the blockchain.

To enable both the block creation and verification flows, two modules need to be modified accordingly:

- The consensus module needs to be aware of transactions that are considered special.
- It needs to be able to verify such transactions when validating blocks.
- The ledger module needs to be able to honor such transactions and include them in its state.

In general, the requirement is that no other entity but the block creator is allowed to modify the ledger based on events coming from the main chain. This requirement needs to be enforced as part of validation rules of the chain and needs to be true also when the committee rotates.

Sidechain flow: moving tokens from sidechain to main chain

To understand the flow, a few details about the example EVM sidechain need to be explained:

- Lock: a user-performed action on the sidechain to move tokens from the sidechain to the main chain. It is implemented via a Solidity smart contract and custom remote procedure

call (RPC) methods. The difference compared to the *burn* action on the main chain is that the sidechain is the owner of all sidechain tokens, even the ones that are temporarily moved to the main chain. The main chain, on the other hand, only represents the moved tokens.

- Settled signature: a piece of finalized data stored on the sidechain (a known amount of blocks have been produced on top of the block containing this data).

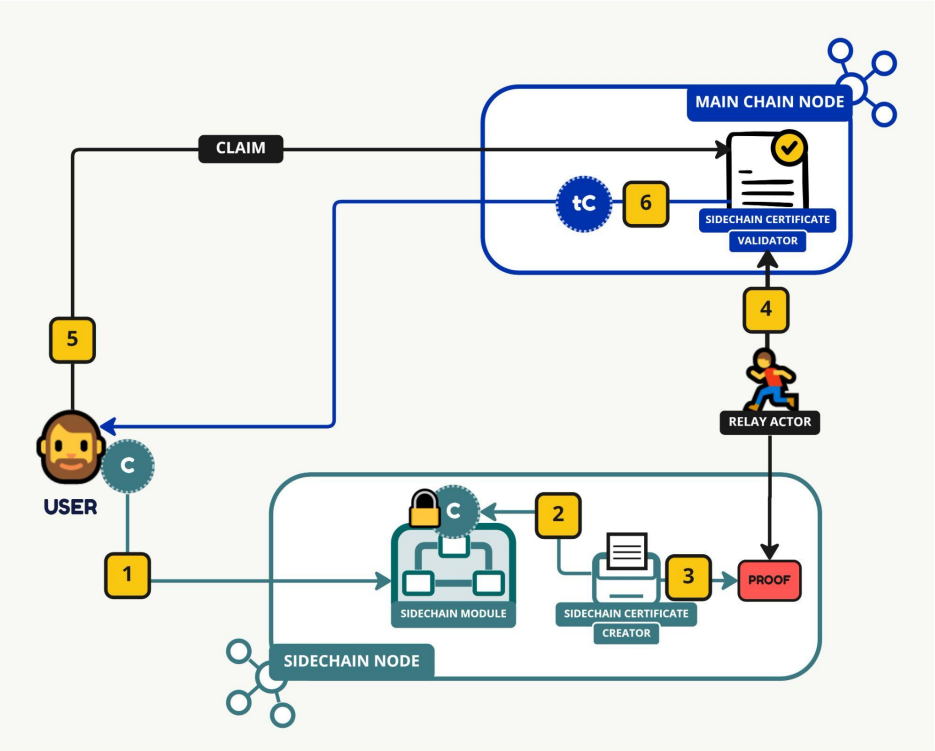

Fig. 20. A sidechain to main chain token transfer interaction diagram

1. The user initiates the flow by invoking on-sidechain mechanisms to *lock* some C tokens.
2. At the end of the sidechain epoch, the Sidechain Certificate Creator starts the process of creating the proof of sidechain to main chain transactions (user issued *lock* transactions). As a result, each committee member persists its own signature to the ledger over the discovered user-initiated *lock* actions.
3. After some time, there are enough signatures posted on the sidechain for the 'n out of m' condition (related to ATMS) to be met and the relay actor can craft a certificate out of these signatures. This step can require sidechain data finalization as a prerequisite for the relay actor to take action.
4. The relay actor invokes the Sidechain Certificate Validator's on-chain script with the certificate of the user-initiated *lock* actions. The Validator checks the posted certificate, and, if successful, persists it on the main chain.
5. At this point, a user calls a *claim* script against the persisted certificate providing proof of ownership of a particular sidechain transaction.
6. If successful, the user will be issued a native token tC representing the C tokens locked on the sidechain.

Sidechain flow: rotating block-producing committee

To understand the flow, one detail about the example EVM sidechain needs to be explained: the rotating block-producing committee creates a linked chain of UTXOs on the main chain going back to the inception of the sidechain. In other words, the management of a particular example EVM sidechain depends on the sidechain block producers certifying the next epoch's committee (more on this in Section 5). Additionally, this flow depends on the ability to observe the registrations captured on the main chain.

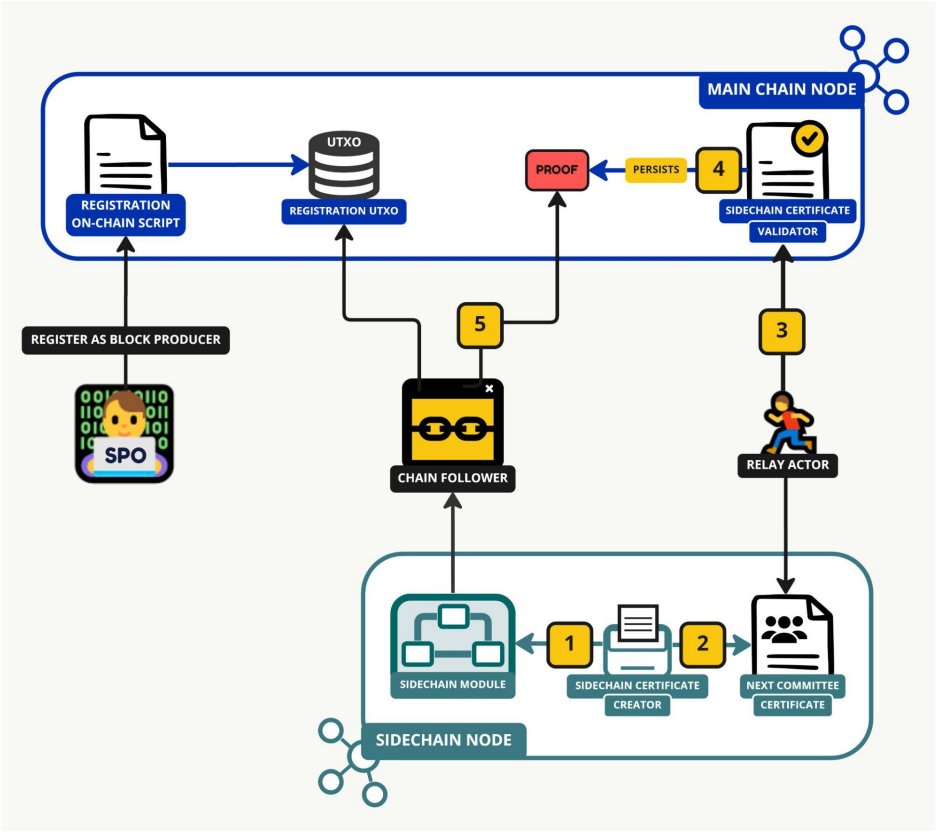

Fig. 21. The full registration flow diagram

- At the end of a sidechain epoch, each committee member calculates the participants of the next committee (based on the on-mainchain registrations) and persists a signature of this calculation to the sidechain ledger.
- After some time, there are enough signatures posted on the sidechain for the 'n out of m' (related to ATMS) condition to be met and the relay actor can craft a certificate of these signatures.
- The relay actor invokes the Validator's on-chain script with the certificate of the next sidechain committee.
- The Validator checks the posted certificate and, if successful, persists it on the main chain.
- The persisted certificate becomes available via the chain follower. This isn't leveraged in the example EVM sidechain, but allows for tighter coupling between main chain certificate availability and sidechain operations if needed.

Summary

This section focuses on the sidechain part of the Cardano sidechain communication pattern. First, the sidechain's view on the 'root of trust' was covered. Key components of the sidechain were introduced, and described and key flows from the perspective of the sidechain were detailed.

4 Sidechain toolkit

Section 1 introduced the idea of a sidechain. Sections 2 and 3 described data dependencies and flows between the sidechain and the main chain that support an example EVM sidechain. With the core ideas and data flows established, it is possible to extrapolate a more generalized approach to building Cardano sidechains: the sidechain toolkit.

What is the Cardano sidechain toolkit?

The toolkit as presented is a study of feasibility and a set of recommendations on how to extend Cardano with new features (consensus protocols, execution environments, custom ledger rules, etc). The goal for this first iteration of the toolkit is to lay the groundwork for what a Cardano sidechain is and where to go next to facilitate and support a family of Cardano sidechains. The example EVM sidechain provides the community with a running sidechain and opens opportunities for further work in the toolkit space. The toolkit is released early in its development cycle to involve the community, SPOs, partners, projects, developers, researchers and other interested parties in discovering and refining Cardano sidechains' functionality, scope and potential use cases.

The team adopted a practical approach with one explicit goal in mind: provide the first iteration of the sidechain toolkit to the community as early as possible. This meant building on top of existing solutions and adapting to new use cases. The team attached an existing client to a Cardano testnet and built out the passive and active flows. The client chosen for modification is a version of an Ethereum client (Mantis [12]) but with the OBFT [1] consensus protocol and with sidechain mode baked into all necessary modules. This means that the client supports Ethereum JSON RPC methods, MetaMask, Remix IDE, EVM Solidity contracts, etc. This also means that a proof-of-work (PoW) consensus protocol is replaced by OBFT, which is a permissioned consensus protocol. More on all this in Section 5.

What is in the sidechain toolkit?

The first iteration of the toolkit consists of:

- Main chain Plutus scripts that address main flows of the example EVM sidechain:
    - Block producer candidate registration Plutus script.
    - Block producer candidate de-registration Plutus script
    - Move tokens to sidechain Plutus script
    - Sidechain Certificate Validator Plutus script
    - Minting policies to support the native token representing the sidechain token
- Chain follower capable of following the on-mainchain events that govern the sidechain
    - The required capabilities are covered by Cardano db-sync
- This technical specification:
    - Serves as an introduction and a guide to Cardano sidechains.

    - Besides that, Section 5 covers in detail how the example EVM sidechain is built. In particular:
        - How to integrate sidechain mode in the consensus protocol.
        - How to make the client dependent on configuration coming from the main chain.
        - How to hoard transactions going out to the main chain.
        - How to accrue committee signatures for outgoing data.
        - How to verify sidechain blocks holding main chain burn events.
        - How to integrate sidechain mode into an Ethereum-like ledger.

The first iteration of the toolkit supports a very particular type of sidechain, which has the following features:

- Observes the main chain for token distribution (ie, ada). The example EVM sidechain implements a set of policies that dictate which parties can become block-producing nodes. There are three criteria:
    - The block producer candidate pool needs to be at least of size **n.**
    - Each block producer needs to have at least **w** ada staked.
    - The sum of ada staked in the sidechain needs to be at least **p.**
    - **n, w,** and **p** are defined for each instance of the example EVM sidechain separately.
- Allows only SPOs as candidates to participate in the block-producing committee of the sidechain. A large part of leveraging the security of Cardano is fulfilled by leveraging the trustworthiness of SPOs that run Cardano. These are the only entities allowed to be block producers of a sidechain in the first toolkit iteration.
- Acknowledges native token burning transactions on the main chain as events for issuing tokens on the sidechain: this design choice leverages an existing on-mainchain mechanic, the ability to create a Cardano 'BurnAct' on the main chain and interpret it on the sidechain. This is a dependency on an implementation detail that in theory could change.
- Supports multiple committee rotations in one main chain epoch: since the action to move tokens from a sidechain to the main chain is tied to the cadence of committee rotations, it is important to have these rotations happen more frequently than on the main chain.
- Depends on a running chain follower instance to be available: this dependency simplifies the design of a particular sidechain module but introduces an additional component. Currently, this component is Cardano db-sync.
- Handles up to *k* blocks of network unavailability: OBFT introduces the notion of finality. After *k* blocks are appended to a chain (where *k* is a sidechain parameter), a particular block becomes final (immutable). This means that the sidechain network is capable of weathering network outages of up to *k* blocks per block producer. This translates to a maximum of *k * slot_length * committee_size* seconds of network outage before a permanent split may happen.
- Creates an append-scheme (trivial-ATMS) style certificate of sidechain events (proof): to expedite the work, a simple ATMS scheme was chosen. The scheme affects the maximum size of the sidechain committee and how the Sidechain Certificate Validator operates. It enables moving data from the sidechain to the main chain.
- No modifications to the main chain were needed: this expedites work and proves that it is possible to build up a sidechain with no modifications to the main chain.

What the sidechain toolkit is not

The first iteration of the Cardano sidechain toolkit, as presented in this document, does not solve:

- Automatic sidechain creation: while the ambitions of the team reach as far as fully automated sidechain creation, provisioning and deployment, this is not yet possible.
- Block creator rewards: the choice to focus on the toolkit meant that some aspects of an operational chain became less important. The tokenomics and rewards are of course important, but can be solved on top of a working sidechain. Hence, they are not covered by this version of the specification.
- Long Cardano mainnet finality: the current implementation of the example EVM sidechain assumes that data will finalize on the main chain before being moved to the sidechain. A sidechain has to declare a maximally allowed failure probability and derive the amount of blocks that need to be observed on the main chain for a particular block to be considered final. For the example EVM sidechain this probability is the same as the for the main chain and is used to calculate the *securityParam* (see Cardano as the 'root of trust'). In the case of the Cardano mainnet, this means that a particular block needs to be followed by 2,160 blocks, which translates to roughly 12 hours.
- Sidechain client SDK: the example EVM sidechain is implemented via a custom client, which has been heavily modified to accommodate the use case. The team would like to provide the Cardano community with a flexible and customizable development kit that would enable any number of custom sidechains. Today's focus is to enable the example EVM sidechain and iterate on it.
- Inter-blockchain communication pattern: a big part of a large family of sidechains is the concern of moving data between sidechains without involving the main chain. This has not yet been solved for Cardano sidechains.
- On-demand transactions to the main chain: as defined for the first iteration of the toolkit, the token transfer from the sidechain to the main chain is tightly coupled with the sidechain epoch cadence. A potential solution to how the cadence is set up is to allow 'on-demand' sidechain-to-main chain transactions. While this also is the team's ambition, the two mechanisms are currently tied together.
- Instant finality: early in the development cycle, a choice was made to move forward with OBFT. It was a great learning experience and it allowed the project and the toolkit to move forward rapidly. One of the features it does not support is instant finality. It would be interesting to find a suitable consensus protocol that can support a sidechain mode and also guarantee instant finality in the presence of a large number of nodes.
- Cost of operating a sidechain: no calculations were made to estimate the cost of running all sidechain components.

Can the sidechain toolkit v1 be reused?

The Cardano sidechain project had one clear goal: to show that it is possible to have a mutually beneficial dependency between two chains. Along the way, components and patterns emerged that could be abstracted, reused, and leveraged for new sidechains.

The biggest reusable piece of the current sidechain toolkit is the Plutus scripts set governing the sidechain. This is by far the most innovative piece enabling the flows described in Sections 2 and 3.

The second one is the way the communication pattern is shaped between the chain follower (using Cardano db-sync) and the sidechain module. While the module is fit for purpose for the example EVM sidechain, the patterns of reading and following the state on the main chain can serve as a running blueprint to follow in other clients.

The third reusable piece is the interaction pattern between the sidechain and the main chain. The way in which data can be hoarded, signed, certified, and moved to the main chain is specific to the implementation of the example EVM sidechain. Note that the pattern can be replicated in other clients.

The fourth claim to reusability is this document. It gathers all the knowledge, decisions, and challenges that were overcome to bring this first Cardano sidechain to fruition.

Lastly, the example EVM sidechain is a great practical application of the ideas laid out in this document for anyone interested in getting their own sidechain running. Even if no code is reused, the communication patterns, dependencies, limitations, and trade-offs are there for anyone to learn from and apply.

Summary

This section focused on introducing the notion of a sidechain toolkit and how the current iteration of Cardano sidechains ties into the toolkit work and the example EVM sidechain.

5 Example EVM sidechain

This section is a deep dive into the example EVM sidechain. It covers the example EVM sidechain setup, main chain components overview, interaction with the chain follower, the sidechain module workflow, and changes in the client to accommodate the sidechain mode.

What is the example EVM sidechain?

The example EVM sidechain is the sidechain toolkit applied to a blockchain client. It consists of a set of Plutus on-mainchain scripts, a chain follower, and a sidechain client capable of running in sidechain mode.

The main chain

The main chain component of the example EVM sidechain serves as the root of trust for the whole system.

It consists of:

- a set of sidechain parameters that govern the behavior of the example EVM sidechain.
- a set of minting policies that govern the example EVM sidechain token.
- a set of Plutus scripts that implement the rules of *burn*, *claim*, and *register* actions.

Additionally, as a companion to the core components, the following set of tools is required:

- sc-evm-cli: a command line application capable of preparing registration messages. This tool is required by the registration flow, among others.
- sidechain-cli: a client-facing set of off-chain scripts that simplifies the UX when interacting with Plutus on-chain scripts, built using the cardano-transaction-lib (CTL) [13]

The following sections provide an overview of the core actions along with examples of ‘sc-evm-cli’ and ‘sidechain-cli’ usage.

Chain follower in the example EVM sidechain

The chain follower component in the example EVM sidechain is realized by Cardano db-sync [8]. It has been chosen for its capability to index all the data that the example EVM sidechain depends on. A detailed description of data dependencies is provided in the following sections.

Sidechain client

The example EVM sidechain client is a blockchain node capable of following a main chain. For the purpose of the example sidechain, an Ethereum-compatible client has been chosen. It is capable of running Solidity smart contracts, it supports Ethereum RPC methods, and has an OBFT consensus protocol that supports sidechain mode. The following sections describe the client's internals, the sidechain module, Sidechain Certificate Creator, and dependency on the chain follower.

Interactions between the main chain and the sidechain

The actions that result in communication between the participating chains have been described and laid out in Sections 2 and 3 of this document. For the purposes of the example EVM sidechain, a detailed description of each action is provided in the following sections, focusing on particular aspects of the interactions. A detailed description of the data residing on the main chain is covered in What data is handled on the Cardano main chain?. Integration of the sidechain client with the chain follower is covered in How is the chain follower integrated with the EVM sidechain client?. Sidechain operation goes into the details of a sidechain's inner workings.

The mentioned interactions can be categorized into 4 types:

- Initialization of the sidechain on the main chain.
- Registration and rotation of block-producing committee members.
- Token transfer to the sidechain.
- Token transfer to the main chain.

Before analyzing each of these flows in detail, it is important to outline the high-level processes and components participating in each of them.

Initialization of the sidechain on the main chain

To start an EVM sidechain, its creator must first perform an initialization action using the 'init' command from the *sidechain-cli* tooling. This action will be monitored by the chain follower and sidechain module through an on-mainchain non-fungible token (NFT). This NFT holds the first block-producing committee hash and is pointed to by a one shot token called `CommitteeHashPolicy`.

Sidechain operation, Sidechain startup conditions, and List of block-producing candidates provide more information on how this process works.

Registration and rotation of block-producing committee members

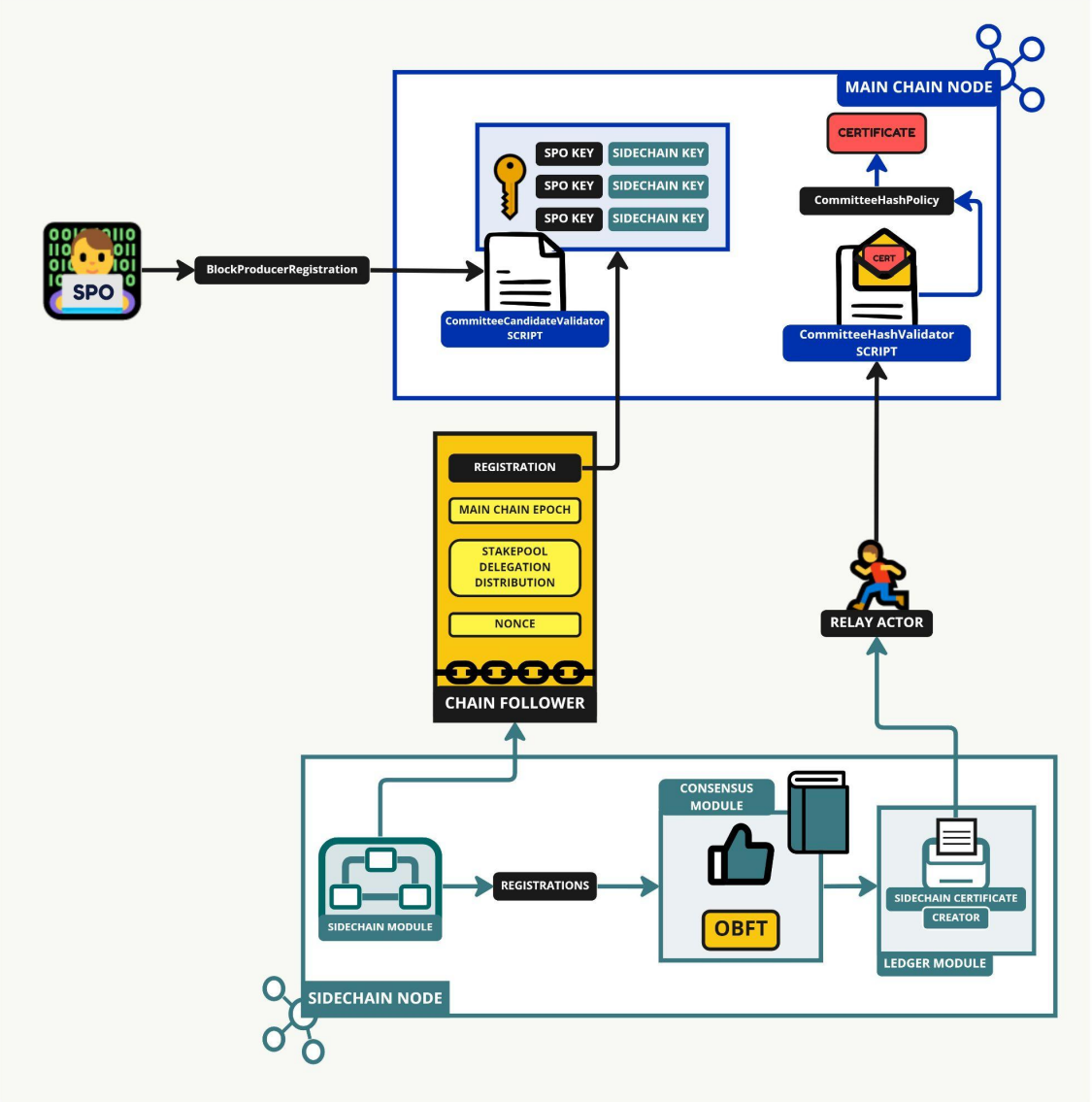

Fig. 22. Registration of SPOs as committee members

During regular operation of an EVM sidechain, the block-producing committee can change. This process is enabled by two related mechanisms: candidates registration and registration monitoring on the sidechain. To register, an SPO posts a `BlockProducerRegistration` datum to the `CommitteeCandidateValidator` script. The script stores this information as a UTXO on the main chain. By monitoring the UTXOs produced by this script, the chain follower can discover all candidates. This list enables committee rotation on the sidechain. By fetching this information from the main chain, any EVM sidechain client can determine and verify the current and past block-producing committees.

In addition to rotating the committee, there is also a handover process. The current block-producing committee uses the Sidechain Certificate Creator to produce a certificate for the next committee. This certificate is transferred to the main chain by a relay actor and posted to the

`CommitteeHashValidator` script. This script is responsible for updating the reference represented by the `CommitteeHashPolicy` token.

These mechanisms are described in more detail in Registration, List of block-producing candidates, Committee handover, and Detailed description of transferring tokens to the main chain.

Token transfer to the sidechain

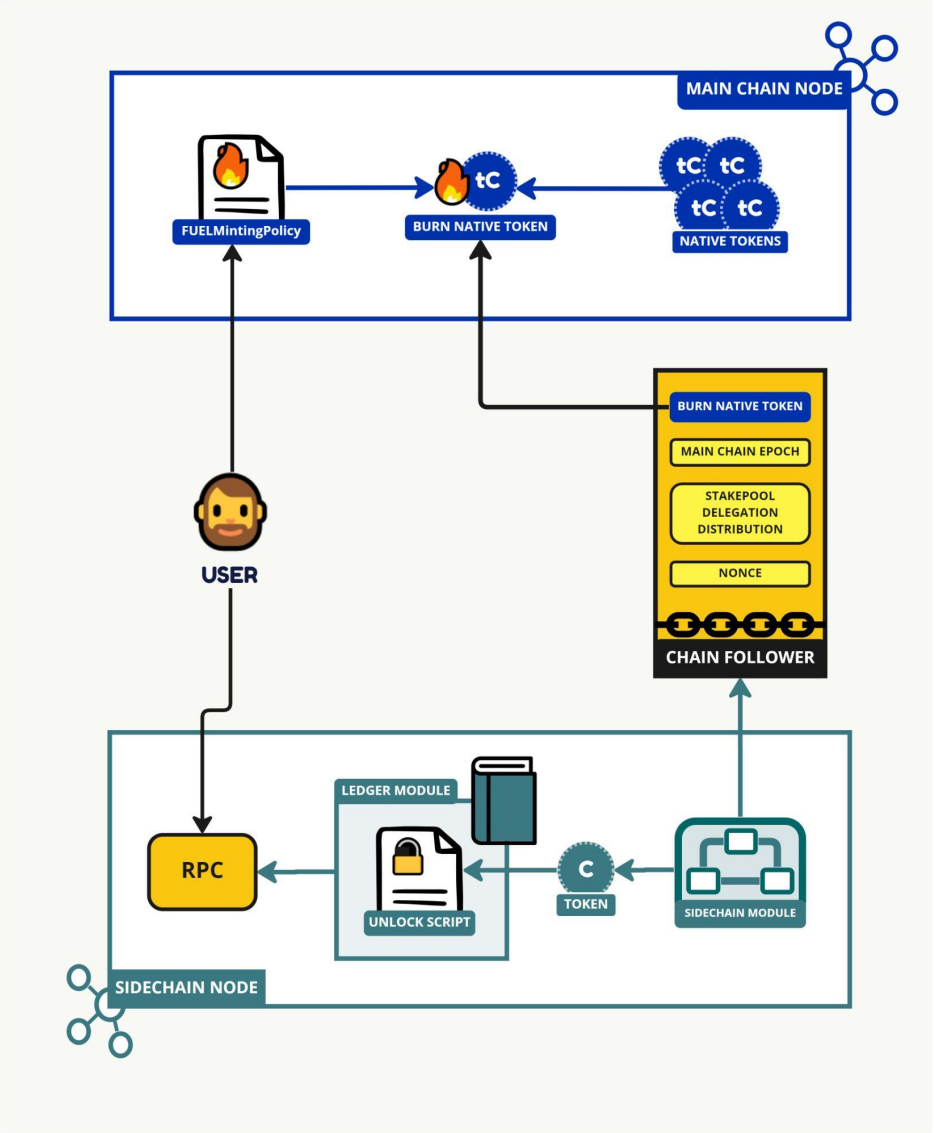

Fig. 23. Token transfer to sidechain

A user can transfer tokens from the main chain to the sidechain by interacting with the sidechain's native token minting policy (called the `FUELMintingPolicy`). This action creates a `BurnAct` on the main chain, which includes the target account address on the sidechain. The chain follower monitors this transaction. The sidechain module transforms it to a sidechain transaction and passes it to the *'unlock'* function of the sidechain bridge smart contract. This function transfers the requested funds to the requested account.

Transferring tokens to the sidechain, Incoming transactions, Sidechain-specific ledger rules, and Detailed description of transferring tokens to sidechain cover this flow in more detail.

Token transfer to the main chain

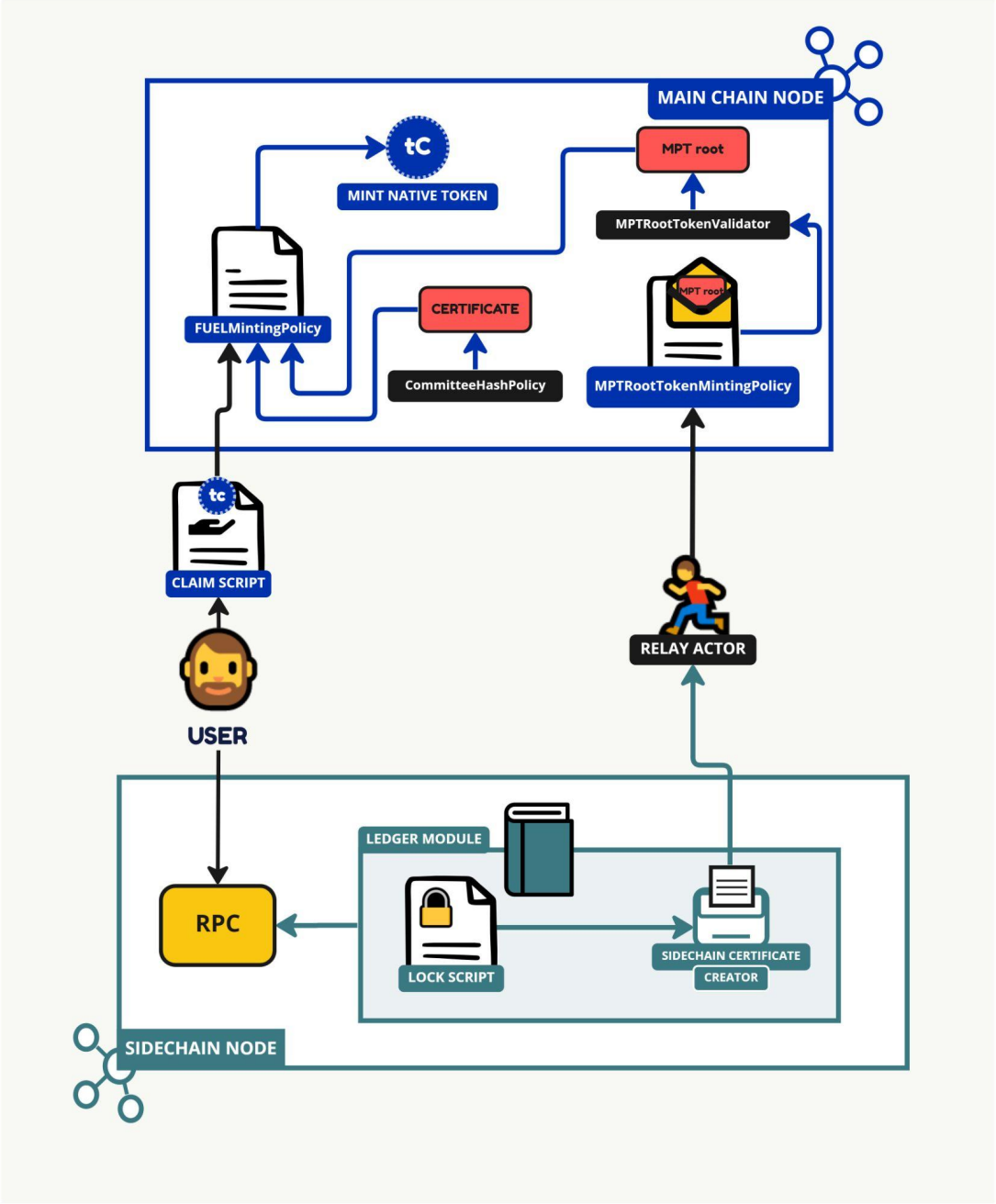

Fig. 24. Token transfer from sidechain

A user can transfer tokens from the sidechain to the main chain by sending a transaction using the `eth_sendTransaction` method and invoking the *lock* function of the sidechain bridge smart contract. The current block-producing committee observes such transactions and signs them. A relay actor picks up these signatures, creates an *outgoing transactions certificate* holding them, and pushes this certificate to the main chain's `MPTRootTokenMintingPolicy`.The policy verifies the data against the expected sidechain committee members and stores a new `MPT root`

representing the sidechain to the main chain transactions. The user then runs a *sidechain-cli claim* script that interacts with the sidechain's native token minting policy (`FUELMintingPolicy`) and claims the requested sidechain tokens on the main chain.

More details on this flow can be found in Transferring tokens from the sidechain, Transactions batch Merkle root upload, and Detailed description of transferring tokens to the main chain.

What data is handled on the Cardano main chain?

This section assumes that a user has some level of familiarity with Plutus [14] and the extended UTXO (EUTXO) model. In particular, it is useful to understand:

- **Datum**: a piece of data attached to a transaction output, which is typically used to carry state.
- **Redeemer**: this is a piece of data attached to the spending input. It is typically used to provide an input to the script from the spender.

The main chain needs to support the following flows to enable the example EVM sidechain:

- Registration
- Token transfer to the sidechain
- Token transfer to the main chain

This section focuses on the data handled, persisted, and settled on the main chain to enable the example EVM sidechain.

Data supporting the (de-)registration flow

To become a block producer on the sidechain, an SPO has to register on Cardano. Each registration is realized by a single UTXO belonging to a specific Plutus contract called CommitteeCandidateValidator. This UTXO identifies the registered entity, the chosen sidechain, and the key to be used to identify the entity on the sidechain. Note that this process is not replayable (ie, no one can reuse this information to register again with the same payload). To register, an SPO must use the following data:

Registration message

The SPO must sign a registration message containing the following fields. The ordering matters:

- sidechainParams (object): an object containing the sidechain params (see description below).
- sidechainPubKey (byte string): the public key from the Elliptic Curve Digital Signature Algorithm (ECDSA) key pair that the block producer will use to sign blocks on the sidechain.
- inputUtxo (object): a UTXO identifier that must be consumed by the transaction that creates the registration. This UTXO is arbitrarily chosen by the SPO and does not need to have any specific property. It acts as a nonce in the signed message to prevent a replay attack. Without it, someone could reuse the signature to register the block producer even after the producer was deregistered.

The **sidechain params** is an object that contains information identifying the sidechain. All these parameters are well-known sidechain properties shared before the sidechain creation. The ordering matters:

- chainId (integer): EVM sidechain ID.

- genesisHash (bytestring): hash of the genesis block for the sidechain.
- genesisUtxo (object): UTXO consumed when creating the initial committee NFT; this field ensures that only one NFT can be minted.
- thresholdNumerator (integer): associated with thresholdDenominator, gives the ratio of signature that must be valid to consider a sidechain certificate valid.
- thresholdDenominator (integer): associated with thresholdNumerator, gives the ratio of signature that must be valid to consider sidechain certificate valid.

UTXOs are defined as objects with two fields – txHash and utxoIndex.
The message is serialized in Concise Binary Object Representation (CBOR), a standard Plutus data format. The resulting bytestring is signed twice: first with the cold secret key, and then with the sidechain private key. These signatures will end up in the registration UTXO datum.

Here is an example of a signed message with the following values:

sidechainParams	chainId	101
	genesisHash	9241f550ab67b74928f0fa33bdf61092f9864cb97783e8254132cadeb6989534
	genesisUtxo	7247647315d327d4e56fd3fe62d45be1dc3a76a647c910e0048cca8b97c8df3e#0
	thresholdNumerator	2
	thresholdDenominator	3
sidechainPubKey	0392d7b94bc6a11c335a043ee1ff326b6eacee6230d3685861cd62bce350a172e0	
inputUtxo	7247647315d327d4e56fd3fe62d45be1dc3a76a647c910e0048cca8b97c8df3e#0	
signed message	d8799fd8799f186558209241f550ab67b74928f0fa33bdf61092f9864cb97783e8254132cadeb6989534 d8799fd8799f58207247647315d327d4e56fd3fe62d45be1dc3a76a647c910e0048cca8b97c8df3eff00 ff0203ff58210392d7b94bc6a11c335a043ee1ff326b6eacee6230d3685861cd62bce350a172e0d8799f d8799f58207247647315d327d4e56fd3fe62d45be1dc3a76a647c910e0048cca8b97c8df3eff00ffff	

Code 1. Signed message and data used to obtain it.

Registration UTXO datum

The registration UTXO must contain a datum with the following fields. The ordering matters:

- mainchainPubKey (bytestring): the SPO's cold verification public key, which is used to identify the SPO.
- sidechainPubKey (bytestring): the public key from the ECDSA key pair that the block producer will use to sign blocks on the sidechain.
- mainchainSignature (bytestring): the signature for the registration message using the SPO cold key.
- sidechainSignature (bytestring): the signature for the registration message using the sidechain ECDSA block producer key.
- consumedInput (object): a UTXO identifier that must be consumed by the transaction that creates the registration. It acts as a nonce in the signed message to prevent a replay attack.
- ownerPublicKey (bytestring): a payment key nominating the owner of the registration. See Deregistration for more details.

Note about registration validation

It is important to understand that on Cardano it is not possible to prevent someone from creating a UTXO belonging to a contract. The Plutus contract only ensures that a transaction spending one of its UTXOs follows whatever rules are defined by the contract. As a consequence, it is the responsibility of the sidechain client to ensure that a registration is valid as anyone can create a UTXO belonging to CommitteeCandidateValidator.

Deregistration

To deregister, an SPO must consume the registration UTXO so that it is no longer unspent. CommitteeCandidateValidator enforces that the transaction consuming the UTXO is signed by the account identified by ownerPublicKey.

Transferring tokens to the sidechain

Sidechain native token transfer from the main chain to the sidechain is done by burning the tokens on the main chain. On Cardano, a mint action contains a 'mint redeemer', which is a piece of data similar to the datum passed as an argument to the minting policy script [15]. The mint redeemer must contain only one field:

- `recipient`: a bytestring representing the recipient's address on the sidechain

When the sidechain block producers see that a burn action becomes stable on the main chain, they can add a corresponding transaction on the sidechain to transfer the funds to the sidechain recipient mentioned in the mint redeemer.

Note about the conversion rate: on an EVM chain, the token is much more divisible than on Cardano because it is on a 32-byte integer. As a consequence, it's not possible to apply a 1:1 ratio when transferring tokens between Cardano and other chains. A constant conversion rate is applied for tokens on the main chain and on the sidechain (parameter TOKEN_CONVERSION_RATE in the Solidity bridge contract on the sidechain). The current value is 10^9, meaning that one sidechain token on Cardano is equal to 10^9 tokens on the sidechain. The bridge contract enforces that transfers from the sidechain always lock a multiple of that value to avoid rounding errors.

Transferring tokens from the sidechain

Uploading a transaction proof to the main chain

In contrast to the token transfers from the main chain to the sidechain that happen independently from one another, the transfer to the main chain happens in a batch every sidechain epoch. Because the main chain does not follow the sidechain, it cannot automatically handle this transfer. Instead, there must be a trusted source for the transactions. To avoid the introduction of an additional entity that could possibly become the target of attacks, a decision was made to trust the majority of the sidechain committee. If the majority of the committee is not trustworthy, the whole sidechain is broken. The committee will craft a certificate proving the fact of sending tokens and present it to the main chain. The process of crafting the certificate requires a two-thirds majority of the committee to reach consensus on at least as many signatures. Since it is more expensive compared to simply signing a transaction, token transfers to the main chain are grouped into batches and sent once per sidechain epoch. This is done by uploading a hash created by the block-producing committee (who have the right to sign a batch of transactions) onto the main chain.

Note on the how a sidechain epoch is organized: an example EVM sidechain epoch lasts for *12*k* slots (where *k* is the stability parameter from OBFT). Users can send their transactions by calling a *lock* function on the bridge's Solidity contract. During the first *8*k* of a sidechain epoch (*regular phase)*, users perform `lock(amount, recipient)` transactions on the bridge contract, which stores them in the current transactions batch. During the next *2*k* slots (*closed batch phase*), the

lock transactions are added to the next epoch's batch. The current epoch batch is signed by the current committee in the last *2*k* slots (*handover phase*).

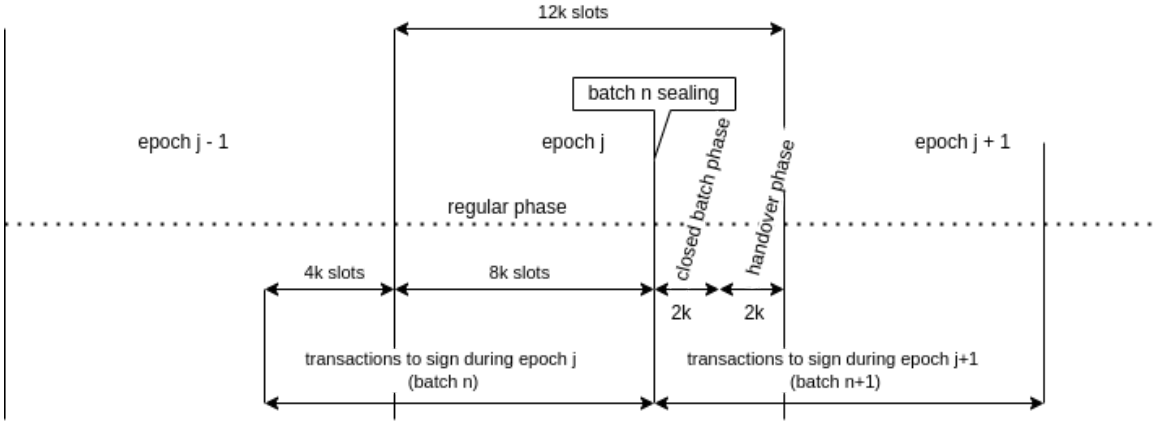

Fig. 25. Epoch phases timeline

At the end of an epoch, each block producer should produce a maximum of two ECDSA signatures:

- a signature for the next committee that will become block producers during the next sidechain epoch.
- a signature for the transactions from the sidechain to the main chain that occurred in the current epoch batch, if any.

If there were no transactions to the main chain, this signature will be omitted. After an epoch is done, a relay actor (does not need to be a sidechain block producer) can fetch these ECDSA signatures from an RPC method *sidechain_getEpochSignatures* exposed by an EVM sidechain node.

Note: the upload process is designed to be extended so that more than one batch can be sent per sidechain epoch.

The relay actor creates a transaction on Cardano calling a specific contract, `MPTRootTokenValidator` (for more details see Main chain Plutus Scripts). The transaction creates an NFT and the mint redeemer contains the following fields:

- merkleRoot (bytestring): Merkle root for the transaction batch.
- previousMerkleRoot (bytestring): the latest Merkle root before this one. This parameter allows the creation of a chain of trusted roots and ensures that a committee cannot add new Merkle roots after the fact.
- signatures (array of bytestrings): a list of signatures, with the same order as their corresponding public keys.
- committeePubKeys (array of bytestrings): a list of public keys of the committee. This list should have the same hash as the one from the current committee.

Each item in the Merkle tree is encoded with standard Plutus data format in CBOR with the following fields:

- index: 32-bit unsigned integer, used to provide uniqueness among transactions within the tree.
- amount: 256-bit unsigned integer that represents the amount of tokens being sent out of the sidechain.
- recipient: arbitrary length bytestring that represents decoded bech32 Cardano address.
- previousMerkleRoot: the latest Merkle root before the current transaction batch.

Claiming of sidechain tokens on the main chain

Users who have previously locked funds in the sidechain bridge contract can use the RPC method `sidechain_getOutgoingTransactions` to find its tx index and `sidechain_getOutgoingTxMerkleProof` to obtain a Merkle proof for it. The proof will allow the minting policy script on Cardano to check that the transaction is indeed part of the Merkle tree whose root was uploaded earlier by the relay.

Users can then claim their token by creating a minting transaction on Cardano. The transaction will be allowed by the minting policy script if:

- The mint redeemer contains the Merkle proof.
- It uses the previously uploaded Merkle root UTXO as a reference input [16].
- The proof matches the Merkle root.
- The transaction was not already claimed (checked via a distributed set [17]).

If successful:

- It updates the distributed set to add the transaction to prevent double spending.
- It creates a UTXO for the recipient with the expected amount.

Note on distributed set: a transaction cannot be claimed twice thanks to a distributed set on Cardano. When claiming, one has to prove that the transaction is not included in the current set. A short explanation of that set is that:

- Transactions are saved in the set as their hash.
- A set is represented as a sorted linked list of transaction hashes. This means that a node in the list will hold the hash of the transaction and the hash of the next element.

This way, one can prove that a transaction T is not in the set by providing a node N where `N.value < hash(T)` and `hash(T) < N.next` because as the list is sorted, if it is was there, then an element `N.next <= hash(T)` would be true.

Example of distributed set usage: Given a hash that is a single letter from "a" to "z".
In this example, an empty set is `{value: "", next: "zz"}` (all values "a" to "z" can be proven to be absent from this node).

To prove that “c” is not in the distributed set, it is sufficient to show that `{value: "", next: "zz"}` is an element. Then, the list is modified to have `[ {value: "", next: "c"}, {value: "c", next: "zz"} ]`.

On Cardano, each list item is stored as a UTXO. The UTXO contains a token whose name is *blake2b_256* hash and the datum is:

datum	`newtype DsDatum = DsDatum { dsNext :: BuiltinByteString }`

Code 2. Shape of the UTXO datum.

When claiming, the contract `DsInsertValidator` ensures that the UTXO that proved the absence of the transaction in the set is consumed and that UTXOs with the new values are created.

Main chain Plutus scripts

The following main chain scripts, policies, and validators are used to manage the sidechain on the main chain:

- `FUELMintingPolicy`: minting policy validating the mint or burn actions of sidechain native tokens on the main chain.
- `MPTRootTokenMintingPolicy`: minting policy for storing sidechain to main chain transaction bundles' Merkle roots.
- `MPTRootTokenValidator`: script address for storing `MPTRootToken`.
- `CommitteeCandidateValidator`: script address for storing registrations of block-producing committee candidates.
- `CommitteeHashValidator`: script address for the committee members' hash.
- `CommitteeHashPolicy`: one-shot token pointing to the current valid committee hash.
- `DsConfValidator`: validator holding a distributed set configuration.
- `DsConfPolicy`: one shot token identifying the UTXO holding a distributed set configuration.
- `DsInsertValidator`: validator handling distributed set entry.
- `DsKeyPolicy`: tokens identifying a distributed set's entries.

All of these policies/validators are parameterized by the **sidechain params** outlined in What data is handled on the Cardano main chain?. This means that all the minting policy and validator script hashes uniquely identify each sidechain.

FUELMintingPolicy

Sidechain tokens are represented on the main chain using the native tokens functionality. `FUELMintingPolicy` is a minting policy validating the mint or burn of sidechain native tokens on the main chain. It is used for transferring tokens from the sidechain (Transferring tokens from the sidechain) and to the sidechain (Transferring tokens to the sidechain). See figure 23.

MPTRootTokenMintingPolicy

Merkle roots serving as a proof of transactions happening on the sidechain have to be persisted on the main chain in a form that allows referencing them from other main chain scripts. This mechanism is implemented as non-fungible native tokens, where each token represents a different and unique Merkle root of all transactions that happened during a sidechain epoch. `MPTRootTokenMintingPolicy` is a policy that governs these tokens by performing the following checks during the minting process:

- verifies that the hash of *committeePublicKeys* matches the hash saved on chain.
- verifies that all the provided signatures are valid.
- verifies that *size(signatures) > 2/3*size(committeePubKeys)*.
- verifies that the list of public keys does not contain duplicates.
- verifies that if *previousMerkleRoot* is specified, the UTXO with the given roothash is referenced in the transaction as a reference input.

If all checks pass, a new non-fungible token is minted and the resulting UTXO is locked using the `MPTRootTokenValidator` script.

MPTRootTokenValidator

`MPTRootTokenValidator` verifies that the UTXOs containing an `MPTRootToken` cannot be unlocked from the script address. See figure 24.

CommitteeCandidateValidator

`CommitteeCandidateValidator` is a script for handling registration (Registration of Cardano SPOs as block-producing candidates) and de-registration (De-registration of Cardano SPOs as block-producing candidates) of Cardano SPOs as block-producing candidates.

CommitteeHashValidator

As mentioned in Registration and rotation of block-producing committee members, each committee is represented as a datum under the UTXO that holds the committee' NFT. During a committee rotation, when a new committee is submitted, the validator script performs the following actions:

1. Verifies that the hash of `committeePublicKeys` matches the hash saved on-chain.
2. Verifies that all provided signatures are valid.
3. Verifies that `size(signatures) > 2/3 * size(committeePubKeys).`
4. Verifies the one-shot token `CommitteeHashPolicy` of the UTXO holding the old verification key at the script address.
5. Consumes the above mentioned UTXO.
6. Verifies that *sidechain epoch of the new committee hash > sidechain epoch of the consumed committee hash UTXO.*
7. Outputs a new UTXO with the updated committee hash containing the committee NFT to the same script address.
8. Verifies that the reference to the last Merkle root is referenced in the transaction.

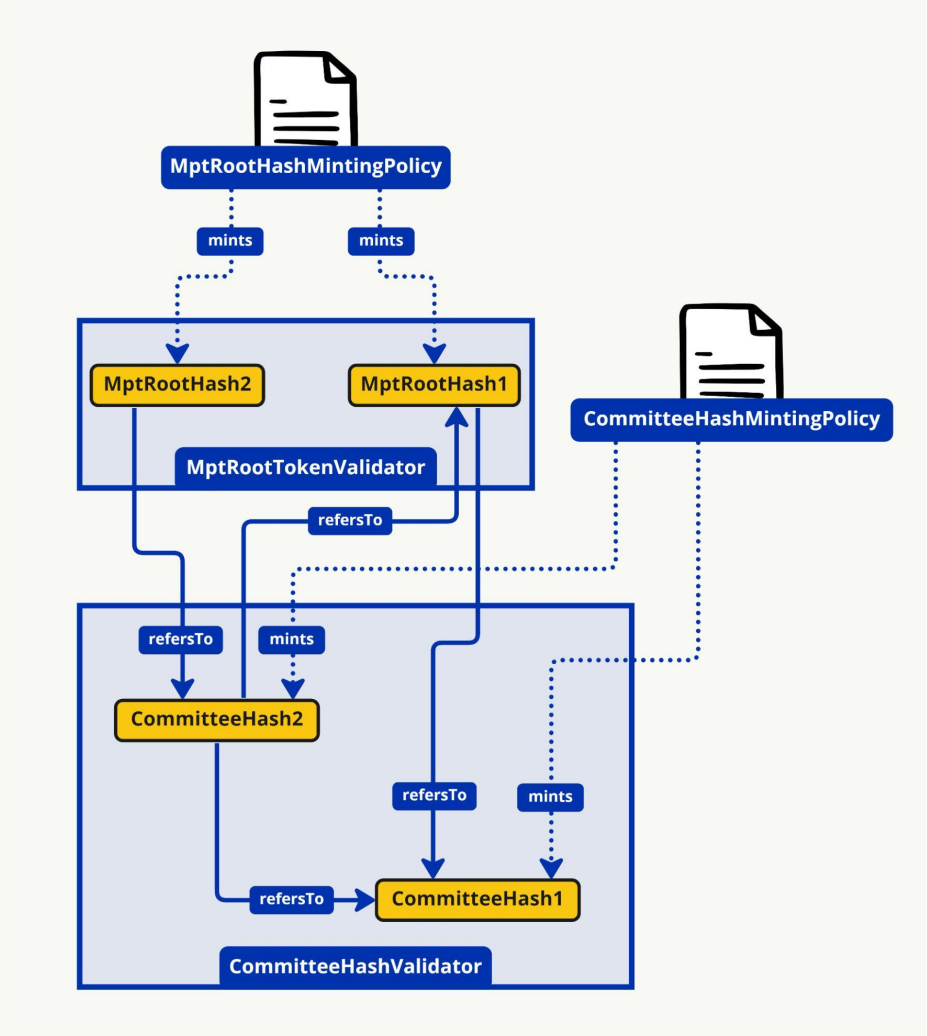

Fig. 26. Active flow artifacts representation on main chain

CommitteeHashPolicy

This is a native token policy that governs the `committeeNft` token. The sidechain committee is stored as a datum on the main chain containing a hash of all public keys of the committee members. That datum belongs to the UTXO that holds the `committeeNft` token. Rotation of the committee is an action during which the current UTXO under the `CommitteeHashValidator` script is consumed, and a new one is created. The ownership of the token is required to make it possible to distinguish this particular UTXO from any other UTXO that can live under that script address. Every sidechain will have exactly one such token that will be minted by the sidechain owner during the sidechain initialization.

DsConfValidator

`DsConfValidator` is the validator that holds the configuration of the distributed set [18] as a datum. It's not parameterized. This validator always returns false and cannot be spent.

DsConfPolicy

`DsConfPolicy` is a one-shot token to uniquely identify the UTXO holding the distributed set configuration. It must be parametrized by a distinct UTXO.

This minting policy verifies the following:

- It spends the distinct UTXO, and exactly one of this token is minted to ensure that this is a one shot-token.
- This token is paid to DsConfValidator.

DsInsertValidator

`DsInsertValidator` is the validator that does the verification of the distributed set insertion operation. It must be parameterized by the currency symbol of `DsConfPolicy` that uniquely identifies a `DsConfValidator`.

DsKeyPolicy

The keys of the distributed set are stored on-chain using the token name of the minting policy `DsKeyPolicy`. It must be parameterized by the validator hash of `DsInsertValidator` and `DsConfPolicy`.

`DsKeyPolicy` uses a redeemer (as defined in What data is handled on the Cardano main chain?), and verifies if it is spending exactly one `DsInsertValidator` with exactly one `DsKeyPolicy` token.

How is the chain follower integrated with the EVM sidechain client?

The chain follower needs to be able to:

- follow example EVM sidechain specific on-mainchain data.
- serve this data to the sidechain client.

The implementation selected for the example EVM sidechain is the Cardano db-sync. The integration with the sidechain client is done using PostgreSQL database queries. The following sections cover in detail how this data is retrieved and handled.

Which data is required to run a sidechain?

The basic data that runs the sidechain:

- Main chain epoch.
- Main chain epoch nonce.
- UTXOs produced by a script.
- UTXOs produced by a minting policy.
- Registration UTXO.

The EVM sidechain client has two main use cases as a sidechain.

- The first use case is validating blocks on the sidechain. Data supporting this use case are:
    - The list of registrations.

    - Stake delegation distribution on Cardano.
    - The Cardano epoch nonce
- The second use case is exchanging tokens between the main chain and the sidechain. This use case is enabled by discovering:
    - When a fuel token burn happens.
    - When the committee is initialized and updated on Cardano.
    - When a Merkle root is uploaded to Cardano.

To get the data, the sidechain uses a trusted chain follower, which makes it easier to query the state of Cardano.

Which chain follower?

Currently, the sidechain client uses Cardano db-sync as its chain follower. Cardano db-sync fetches Cardano's on-chain state and stores it on a PostgreSQL database, which means that any application can use SQL for data querying. Note that the chain follower database state is the only view of Cardano available to the sidechain client, so it needs to be trustworthy. As a consequence, it is a requirement to run a sidechain client against a trusted db-sync node to ensure that the data has not been tampered with.

The fact that the example EVM sidechain client uses Cardano db-sync is hidden behind a few interfaces that return only useful data from Cardano, namely:

- Main chain epoch nonce.
- A list of partially validated candidates. The chain follower layer ensures that the registrations are well formed but the full validation is the responsibility of the sidechain module.
- A list of incoming transactions.
- The committee NFT.
- The Merkle root NFTs

One important aspect of the relationship between sidechain and main chain is that the sidechain does not only need to get the current state of Cardano, but also needs to be able to validate historical blocks and transactions to answer questions like:

- What was the candidates list during a given arbitrary Cardano epoch?
- Was a transaction from the main chain to the sidechain finalized when a particular block was emitted?

This document assumes the usage of Cardano db-sync 13.0.5.

List of block-producing candidates

The most frequent information the sidechain needs to get is the list of block-producing candidates. Registrations are represented as UTXOs on a specific address with a specific datum.

The list of candidates needs three main data points to be available:

1. Which block represents the state that will be the reference for the registrations.
2. The open registrations at that block.
3. The main chain stake distribution for the registered SPOs for the epoch.

Finding the reference block

To ensure that the transaction containing the registration is stable at the beginning of a given epoch, there is a registration slot in the previous epoch used as a reference point to look at the main chain state. This reference point is used as the source of the block-producing candidate list for the next epoch. In other words, a registration or de-registration happening after this reference point will not be considered for the next epoch.

The chosen slot depends on two parameters from the Cardano main chain [9]:

- `securityParam`: the number of blocks that must be produced on top of another block for it to be considered stable. For reference, this is 2,160 blocks on Cardano mainnet.
- `activeSlotsCoeff`: the probability of having a block produced in a Cardano slot. For reference, this is 0.05 on Cardano mainnet.

The slot is chosen to be `2*<securityParam>/<activeSlotsCoeff>` slots before the first slot of the epoch.

Then the sidechain module finds the latest block (the reference block), which was appended to the chain during or before the reference slot (ie, the highest block with the slot lower or equal to the reference slot) with the following query:

reference block query	
	`SELECT` `  block.block_no,` `  block.hash,` `  block.epoch_no,` `  block.slot_no,` `  block.time` `FROM` `  block` `WHERE block.slot_no <= $slot` `ORDER BY block.slot_no DESC` `LIMIT 1`
parameters	slot: reference slot

Code 3. Reference block query

Finding the registration

Finding the registrations means that the sidechain client needs a query to find UTXOs belonging to an address of the registration script at the reference block (see Finding the reference block).

query	
	`SELECT` ` origin_tx.hash as origin_tx_hash,` ` tx_out.index as utxo_index,` ` origin_block.block_no as tx_block_no,` ` origin_block.slot_no as tx_slot_no,` ` origin_block.epoch_no as tx_epoch_no,` ` origin_tx.block_index as tx_block_index,` ` tx_out.address,` ` datum.value as datum,` ` consuming_tx.hash as consuming_tx_hash,` ` consuming_block.epoch_no as consuming_tx_epoch,` ` consuming_block.slot_no as consuming_tx_slot` `FROM tx_out` `INNER JOIN tx origin_tx ON tx_out.tx_id = origin_tx.id`

	INNER JOIN block origin_block ON origin_tx.block_id = origin_block.id LEFT JOIN datum ON tx_out.data_hash = datum.hash LEFT JOIN tx_in consuming_tx_in ON tx_out.tx_id = consuming_tx_in.tx_out_id AND tx_out.index = consuming_tx_in.tx_out_index LEFT JOIN tx consuming_tx ON consuming_tx_in.tx_in_id = consuming_tx.id LEFT JOIN block consuming_block ON consuming_tx.block_id = consuming_block.id WHERE tx_out.address = $address AND origin_block.block_no <= $unspentAtBlock AND (consuming_tx_in.id IS NULL OR consuming_block.block_no > $unspentAtBlock)	
parameters	unspentAtBlock	block used as a reference point to get the UTXOs. It does not matter if the UTXOs are consumed after this block
	address	wallet address to which the UTXOs belong

Code 4. Registrations query

This query fetches the following information about UTXOs:

- transaction hash and index for the UTXO.
- block information about the UTXO (block number, slot, and epoch).
- datum (if any).
- information about a consuming transaction (if any) (transaction hash, block number, and slot).

The query also filters out UTXOs that were either created after the given block or spent before that block. This way the sidechain module gets the state of the chain at the block passed as parameter '`unspentAtBlock`'.

Finding the stake delegation distribution

The sidechain client gets the stake delegation distribution with the following query:

query	SELECT ph.hash_raw, SUM(es.amount) FROM epoch_stake es INNER JOIN pool_hash ph ON es.pool_id = ph.id WHERE es.epoch_no = $epoch GROUP BY ph.hash_raw	
parameters	epoch	epoch for which the stake delegation distribution is requested

Code 5. Stake delegation distribution query

The sidechain client can then validate discovered block-producing candidates by parsing and validating the datum and matching each candidate with its stake. Note that due to a limitation in db-sync, there is no way to know the stake of an SPO at the beginning of a given epoch as it is computed in a non-transactional way so it is not trivial to determine when the computation is done. For this reason, the sidechain uses the stake delegation distribution from the previous epoch instead of the current one.

Finding the Cardano epoch nonce

A final query the sidechain needs for the committee computation is the nonce for a Cardano epoch.

query	SELECT nonce FROM epoch_param WHERE epoch_no = ${epoch.number}	
parameters	epoch	epoch for which the nonce is requested

Code 6. Epoch nonce query

The discovered value of nonce is used as the seed when running a deterministic block-producing committee selection algorithm. One limitation of db-sync is that it only adds an epoch in the database once it's started, but the sidechain needs to know the nonce in advance if we want to sign the certificate for the next epoch. As a consequence, the sidechain uses the nonce from the previous main chain epoch.

Incoming transactions

In the case of the example EVM sidechain, transactions that move value to the sidechain are main chain burn actions. The following query is used to discover the required data:

query	SELECT rd.value AS redeemer, mtm.quantity AS value, tx.hash as tx_hash, b.block_no as block_no FROM ma_tx_mint mtm INNER JOIN multi_asset ma ON ma.id = mtm.ident AND ma.policy = ${fuelAsset.policy} AND ma.name = ${fuelAsset.name} INNER JOIN redeemer r ON r.tx_id = mtm.tx_id INNER JOIN redeemer_data rd on rd.id = r.redeemer_data_id INNER JOIN tx ON tx.id = mtm.tx_id INNER JOIN block b ON tx.block_id = b.id WHERE b.slot_no >= $fromSlot AND b.block_no <= $blockUpperBound AND mtm.quantity < 0 AND r.purpose = 'mint' ORDER BY (b.block_no, tx.block_index)	
parameters	fuelAsset.policy	the policy ID for the sidechain native token name (this is a hash of the script that validates mint operations)
	fuelAsset.name	the asset name as a bytestring
	fromSlot	lower bound for the slot
	blockUpperBound	maximum block number to consider

Code 7. Incoming transactions query

The sidechain client enforces that the transactions happen in the same order on Cardano and on the sidechain. This means that the above query fetches the list of burn actions sorted by the block number and transaction index, starting at the latest processed transaction. The sidechain queries burn actions (ie, negative mint actions) for the sidechain native tokens and fetches the redeemer, which contains the requested token recipient. It also filters out transactions that are not yet stabilized on Cardano.

Committee handover

During a committee handover, the next committee is selected and the output is hashed and sent to Cardano to a contract called `CommitteeHashValidator`. The current committee is identified via a committee NFT. The UTXO owning the NFT will contain the hash of the latest committee uploaded to Cardano. This is not used to perform the handover itself, but as a way to observe what is happening on the main chain. On a low level, the sidechain searches for UTXOs containing a specific NFT using the following query:

query	SELECT origin_tx.hash AS origin_tx_hash, tx_out.index AS utxo_index, origin_block.epoch_no AS tx_epoch_no, origin_block.block_no AS tx_block_no,

	`` origin_block.slot_no AS tx_slot_no, origin_tx.block_index AS tx_block_index, datum.value AS datum FROM ma_tx_out INNER JOIN multi_asset ON ma_tx_out.ident = multi_asset.id INNER JOIN tx_out ON ma_tx_out.tx_out_id = tx_out.id INNER JOIN tx origin_tx ON tx_out.tx_id = origin_tx.id INNER JOIN block origin_block ON origin_tx.block_id = origin_block.id LEFT JOIN datum ON tx_out.data_hash = datum.hash LEFT JOIN tx_in consuming_tx_in ON tx_out.tx_id = consuming_tx_in.tx_out_id AND tx_out.index = consuming_tx_in.tx_out_index WHERE multi_asset."policy" = ${nft.policy} AND name = ${nft.name} AND consuming_tx_in IS NULL LIMIT 1 ```	
parameters	nft.policy	the policy ID for the committee NFT
	nft.name	the asset name for the committee NFT (note that at the time of this writing this is empty)

Code 8. Committee handover query

The sidechain module queries for a UTXO that contains a token corresponding to the committee NFT policy and an asset name. This process assumes that the result is really an NFT. In other words, it does not check if several UTXOs match the query, since only one unspent output should exist with that token. It fetches information about the transaction and the datum.

Transactions batch Merkle root upload

The sidechain does not need to track uploaded Merkle roots for regular operations since this part is handled by the Plutus scripts governing the sidechain. Making it available in the sidechain client streamlines user experience by making the necessary data for next epochs upload readily available. Each Merkle root upload creates its own NFT (different name but with the same policy ID) so it is enough to query for the latest mint action for a given policy ID.

query	``` SELECT origin_tx.hash, ma."name", origin_block.block_no, origin_tx.block_index, rd.value FROM ma_tx_mint mtm INNER JOIN multi_asset ma ON ma.id = mtm.ident INNER JOIN tx origin_tx ON mtm.tx_id = origin_tx.id INNER JOIN block origin_block ON origin_tx.block_id = origin_block.id LEFT JOIN redeemer r ON r.tx_id = mtm.tx_id LEFT JOIN redeemer_data rd ON rd.id = r.redeemer_data_id WHERE ma."policy" = $policyId ORDER BY (origin_block.block_no, origin_tx.block_index) DESC LIMIT 1 ```	
parameters	policyId	the policy ID for the Merkle root NFTs

Code 9. Transactions batch Merkle root upload query

The sidechain module fetches the mint redeemer for the transaction, which contains some information about the Merkle root and the signatures.

What is the example EVM sidechain client?

The example EVM sidechain client is a blockchain client capable of executing Solidity contracts, is compatible with Ethereum RPC methods specification, and follows an OBFT consensus protocol adapted to run in sidechain mode.

The client

The example EVM sidechain client is a Java Virtual Machine (JVM)-based blockchain client written in Scala 2.13. It is based on the work done for the Mantis client [12]. As such, it complies with the Ethereum RPC methods specification [19]. As of this writing, it supports only the sidechain mode, in other words it is only capable of following a main chain. Beyond that, for the testnet period, the client will not have a peer-to-peer (P2P) layer enabled.

Adaptations for OBFT

The example EVM sidechain client has a custom implementation of the OBFT consensus protocol. OBFT is an Ouroboros proof-of-stake-inspired blockchain protocol, which replaced the Ethash [20] proof-of-work-based (PoW) consensus protocol in the client. Minor adjustments to internal operations were needed, namely:

- Difficulty is set to a constant value both from the perspective of the EVM and block handling.
- The ommer set is always empty.
- BASEFEE op-code [21, 22] returns a constant as this London feature is not leveraged in the sidechain yet.

The RLPx protocol is based on version eth/66 with some minor adjustments:

- New messages introduced: `GetStableHeaders`, `StableHeaders`, `GetFullBlocks`, `FullBlocks`.
- Status message contains the stability parameter and hash of configuration.

Adaptations for sidechain mode

To enable sidechain mode in the example EVM sidechain client, the following adaptations were made:

- Custom RPC methods in the '`sidechain_`' namespace: these methods allow introspection into the state of the sidechain and the main chain.
- Adapted OBFT consensus protocol and module: adaptations cover the notion of a sidechain epoch and rotation of the block-producing committee.
- Adapted ledger rules: adaptations cover the rules governing inclusion of main chain to sidechain transactions and mechanisms covering the end-of-sidechain-epoch ceremonies.

Architecture

The client serves two independent interaction paths:

- RPLx protocol for inter-client communication (such as block exchange, transaction gossip, catch-up, etc.)
- RPC methods for user-facing APIs (posting transactions, checking receipts, fetching execution logs, etc.)

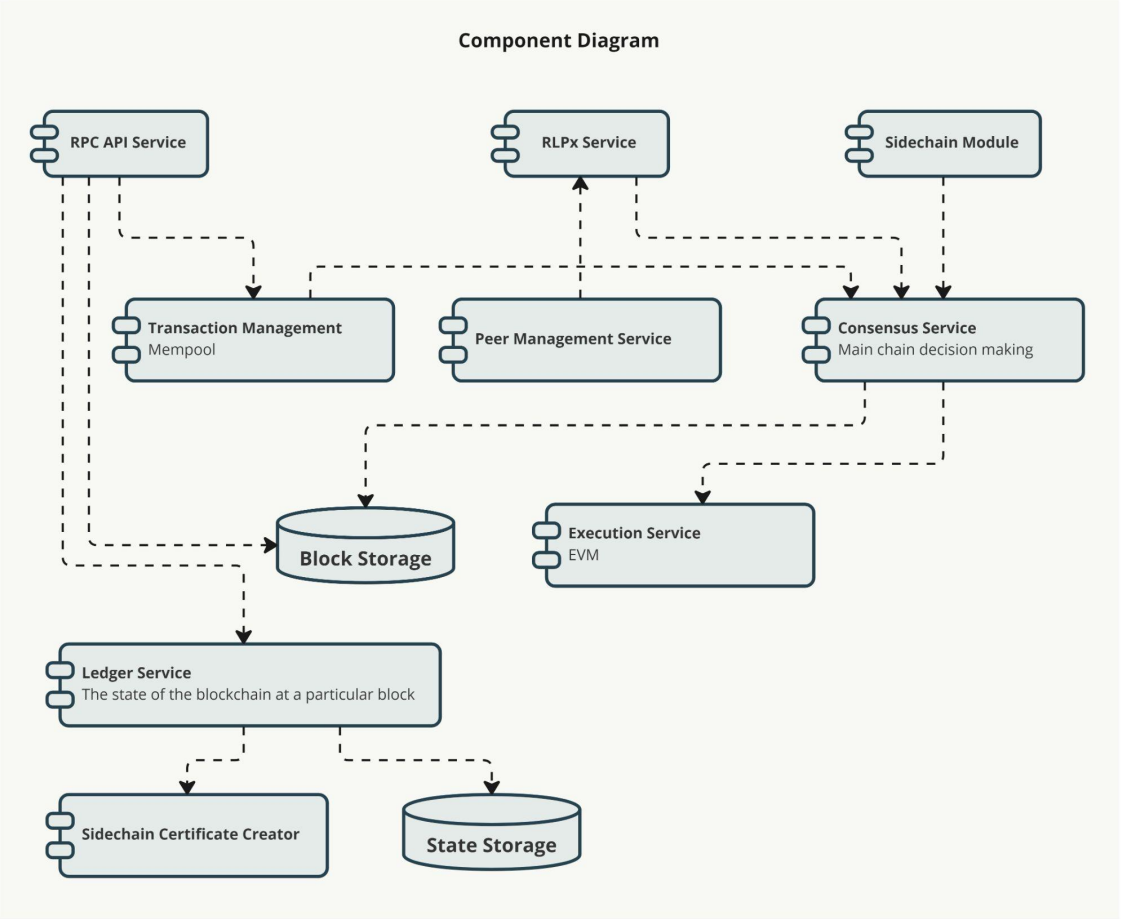

Fig 27. Example EVM sidechain client architecture diagram

On a very high level, the client is organized into three streams that drive the progress of the blockchain:

- Slot ticker: whenever it is time to mint a block (and as long as the client is set up to produce blocks), the slot ticker triggers the consensus module to perform the necessary steps to create a new block.
- Epoch ticker: whenever a sidechain epoch draws to a close, the epoch ticker triggers the necessary steps on each block-producing client.
- Incoming data: whenever new data is served by peers, incoming data triggers an internal response: validation, processing, and potentially storage.

Sidechain module

The sidechain module contains the logic needed in the sidechain client to support the interactions described in Sidechain interactions. In addition to that, due to the difference in how tokens are represented on the sidechain and on the main chain, the sidechain module is responsible for performing token amount conversion so that both chains can represent an equally broad range of values (refer to Transferring tokens to the sidechain, Note about the conversion rate for more details).

We can further divide the sidechain module into the following areas:

- Consensus customization.
- Customized ledger rules set.

- A persistence layer that allows storing sidechain specific data.
- Start-up conditions for the sidechain.
- Sidechain-specific JSON RPC methods.

Many of those areas need data from the main chain to operate. Because of that, the sidechain module has a strict dependency on a Cardano follower module.

To explain those components' functionality, some additional terminology is needed.

Transactions between the sidechain and the main chain are called *cross-chain* transactions:

- Outgoing transactions initiate on the sidechain and result in moving value to the *main chain*
- Incoming transactions initiate on the main chain and result in moving value to the *sidechain*. See Transferring tokens to the sidechain.

Sidechain epoch

The EVM sidechain with OBFT consensus protocol has a stability parameter *k* (more on this in Consensus protocol - OBFT).

Sidechain epoch requirements are:

- Each sidechain epoch spans *12*k* slots.
- Sidechain epoch length is equal to *(main chain epoch length) / N*, for some natural *N*.
- There exists a main chain epoch that starts at the same time as the first sidechain epoch.

A sidechain epoch is divided into three phases:

1. *Regular* phase, which spans the first *8*k* slots. Outgoing transactions from this phase will be signed in the *handover* phase of this epoch.
2. *Closed transactions batches* phase, which spans the next *2*k* slots. Transactions from this phase will be signed in the *handover* phase of the next epoch.
3. *Handover* phase, which spans the last *2*k* slots. Block producers add transactions with signatures for committee handover and outgoing transactions batches. New transactions from this phase will be signed in the next epoch.

Sidechain-aware consensus

One of the requirements for the passive bridge was to allow for committee rotation among the candidates that registered on the main chain using the 'registration' Plutus smart contract. The sidechain module defines a component that implements committee discovery and its rotation over time in a deterministic way, using the main chain as a source of entropy (nonce).

The formula for calculating a committee for a particular sidechain could be sampling with replacement or any other standard technique eg "GearBox: Optimal-size Shard Committees" [36]. This calculation is verifiable and favors including candidates with larger stake delegations.

Sidechain-specific ledger rules

Below is the list of ledger rules that were either modified or added to support sidechain related operations:

1. The sidechain storage tracks several parameters: slot number, phase of the epoch (see Uploading a transaction proof to the main chain) and epoch number. This change is not reflected on the transaction level but it is a direct change in the sidechain state (similarly to how rewards in Ethereum are implemented). To keep them in sync, each new block has to make necessary changes to the Merkle Patricia Trie [23]. It is considered invalid otherwise.

2. During the handover phase, when producing a block, each block producer has to include a transaction in that new block that targets a specific address, including:
    a. Signature for the next committee.
    b. Signature for outgoing transactions for a given epoch, if any.

    Outgoing transactions are stored in the sidechain storage. Not including either of these (or including a signature for outgoing transactions where there should be none) will result in the block being considered invalid by the rest of the network.
3. Incoming transactions that reflect transfers coming from the main chain can happen at an arbitrary time. In other words, there is no rule enforcing inclusion of pending incoming transactions in the ledger beyond the rule that the EVM sidechain preserves and enforces the order of main chain transactions. On the other hand, validators need to cross-check every incoming transaction included in the block against their state of the chain follower. On the EVM layer, such transactions must result in the emission of a particular EVM event with correct values. Any deviation in those values will result in the block being considered invalid by the rest of the network.

Sidechain storage

Many of the flows that the sidechain supports could be implemented without a designated persistence layer, making use of the fact that blockchain is a stateful component. However, from the perspective of simplifying implementation and making it more discoverable for the end users, the team decided to implement sidechain-specific storage. The storage is implemented as a regular smart contract compatible with Ethereum smart contracts specification and can be inspected using regular EVM tooling (eg Remix IDE). In other words, it does not use any non-standard EVM opcodes, as that would prevent it from running on different implementations of the EVM. The contract is called `BridgeContract` and it is included in the genesis block of the example EVM sidechain.

The BridgeContract is responsible for:

- Keeping funds locked when transferring tokens to the main chain.
- Storing batches of outgoing transactions for the purpose of a Merkle root creation.
- Keeping track of the Merkle roots chain.
- Storing signatures for the purpose of sidechain certificate creation.
- Unlocking funds when transferring tokens to sidechain.
- Performing conversion between main chain and sidechain token amounts.
- Keeping track of the latest handled incoming transaction.

Sidechain startup conditions

The sidechain module is also responsible for providing custom logic that oversees the sidechain initialization process. Below is a breakdown of these rules into two categories.

Quantity and quality restrictions regarding sidechain candidates

The first requirement for the sidechain to be started is to have a sufficient amount of valid SPOs registered for participation. A ‘sufficient amount’ is defined by a configuration parameter (`sc-evm.sidechain.committee.min-threshold`) of a sidechain. It can be set to an arbitrary number that is higher or equal to the configured size of the committee *S*.

The qualification of SPOs for a particular sidechain can be implemented in many different ways. For now, there is a single rule that requires SPOs to have at least the configured minimal amount of ada delegated to them (`sc-evm.sidechain.min-candidate-stake`).

Last but not least, there is the `sc-evm.sidechain.committee.min-stake-threshold` configuration property that defines the minimum sum of the stake delegated to all candidates for the sidechain to be started.

These properties allow the sidechain creator to set some expectations regarding the amount and quality of SPOs required for the sidechain to transition into an operational mode.

Supporting conditions

At the end of each sidechain epoch, the committee needs to create a valid sidechain certificate that will be put on the main chain. Such a certificate needs a certain threshold of signatures (configured during the sidechain creation by its creator) in order to be considered valid by the main chain smart contract. Because of that, the sidechain cannot start at the end of a sidechain epoch, as that would not give the algorithm enough time to collect the required amount of signatures and so no certificate could be posted on the main chain. A sidechain that cannot post any proofs on the main chain is not very useful. This rule ensures that a sidechain won't start during the handover phase of the epoch, effectively preventing the unwanted scenario from playing out.

Sidechain-specific JSON-RPC methods

The following additional JSON-RPC methods are supported for the purpose of implementing passive and active flows as well as making the user experience better:

- `sidechain_getCurrentCandidates`: returns most up-to-date candidates.
- `sidechain_getCommittee`: returns committee for a given sidechain epoch.
- `sidechain_getCandidates`: returns a list of registered candidates for the sidechain for a given epoch.
- `sidechain_getEpochSignatures`: returns signatures for a given sidechain epoch that can be used to create a sidechain certificate.
- `sidechain_getStatus`: returns current synchronization status for both sidechain and the chain follower.
- `sidechain_getOutgoingTransactions`: returns a list of outgoing cross-chain transactions for a given sidechain epoch heading towards the main chain.
- `sidechain_getPendingTransactions`: returns a list of incoming cross-chain transactions coming from the main chain that await finalization or are already finalized and awaiting processing.
- `sidechain_getParams`: returns sidechain-specific operational parameters.
- `sidechain_getOutgoingTxMerkleProof`: returns Merkle proof of the transaction that happened on the sidechain for the purpose of claiming tokens on the main chain.
- `sidechain_getSignaturesToUpload`: a helper endpoint used as a part of automated submission of the sidechain certificate onto the main chain.

For a more detailed description of these endpoints and rest of the API, please read the API documentation [24, 35].

Consensus protocol - OBFT

OBFT [1] is a consensus protocol implemented in the example EVM sidechain. As such, it has the responsibility to eventually build agreement over the network. OBFT is a deterministic protocol designed to be simple while guaranteeing consistency and liveness. It is resilient to Byzantine faults up to one third of participants, and it accommodates temporary loss of synchrony (network split or delays up to the stability parameter *k*). Additionally, when the chain is observing a period of

synchrony, a maximum of one child per parent block is produced, resulting in efficient usage of resources.

Overview of the protocol

This section explains how OBFT defines the behavior of its participants without modifications to accommodate sidechain mode. Dedicated comments and Sidechain module provide details on the adaptations of OBFT in order to work as a sidechain.

A brief overview of the interactions
A common behavior for all participants is to build a local version of the chain based on the blocks received via the network. Some participants may also act as block producers. Upon the creation of a new block, the issuer updates their local chain with the new block and then broadcasts the block to other participants. Upon the reception of a block, a participant evaluates whether the block is valid and contributes to extending the local chain. The local chain is always a valid chain. It is valid when all the blocks are legitimate and all their transactions have been executed successfully.

A brief overview of subsections

- Participation describes when a participant is a legitimate block producer.
- Selection of truth describes how a participant selects one chain from all the alternatives received from the network.
- Stability describes the portion of a local chain that is indefinitely settled.
- Mempool refers to a set of transactions to be incorporated into the blockchain.
- Bootstrapping explains how a participant connects or reconnects to the network (ie, how a participant builds their local chain based on the local chain of their peers).

Participation

The question of defining whether a participant legitimately issued a block is central in the resolution of consensus in a blockchain. OBFT simplifies the question of participating in the chain by defining ahead of time when a participant has the right to propose a new block.

To establish deterministic participation, the protocol limits its participants to an upfront known set of entities called a block-producing committee. These block producers are in turn responsible to extend the chain. The blockchain is treated as a sequence of intervals called slots. A slot is an opportunity to extend the chain with a single block. Each slot is allocated to exactly one block producer. A block producer becomes a slot leader and is the only rightful issuer of a block for a given slot number. As a result, the duration of a slot (`sc-evm.pos.slot-duration` parameter) defines the pace of growth of the chain because there is at most one block per slot. The division of slots ensures the right to issue a block being equally distributed among the block producers.

In comparison with a PoW protocol, the role of block producers is very similar to *miners*. In both cases, they propose new blocks. Since in OBFT each block producer is responsible for issuing blocks at a determined slot, the participants do not compete to have their block included in the local chains of their peers. As a consequence, both creating and validating a block is cost efficient, fast, and simple. Creating a block means signing a block, which is achieved with the private key of the block producer assigned to a given slot. Validation of the right to participate is simply verifying that the block was issued by a slot leader and that it is properly signed. In fact, the work carried out by a block producer is not significantly bigger than for any other node following the chain without issuing blocks. This means that the resources and the energy needed to run such a protocol are greatly reduced compared to a PoW approach.

Note: The adaption of OBFT to sidechain mode adds the notion of epochs and the rotation of block producers. An epoch is the period during which the set of block producers is fixed. The block producers are elected for the duration of an epoch via candidatures observed on the main chain. During the epoch, the protocol and the allocation of slots are applied as described before, considering the elected set of block producers.

Selection of truth

Ideally, the participation *in turns* results spontaneously in a unique chain because block producers are not competing for slots and thus the blocks extend the chain one by one. Therefore, upon reception of a block, if the block is properly issued and coherent with the previous blockchain state, the local chain is updated. However, in the event of faults, network issues, or malicious behavior, forks may appear proposing alternative versions of the chain. A deterministic selection of a unique source of truth must be established. Upon reception of new blocks, only the valid forks are considered. That is to say that all their blocks are rightfully issued and the sequence of transactions held by the chain is coherent. In the next step, the local chain is replaced by the longest chain amongst all known valid forks, assuming that the fork is longer than the local chain. The logic behind selecting the longest fork translates into the amount of participation – the longer the chain, the more participation it allows. Hence, a higher *density* as there is a higher number of blocks in a given number of slots. As a consequence, a higher *density* is achieved when participants collaborate to build a common chain because of *in turn* participation.

Stability

Another key aspect of the protocol is the stability parameter *k* (`sc-evm.pos.stability-parameter-k`). Considering a valid chain, any block deeper than *k* blocks from the latest block is considered as stable. That is to say that the *k* newest blocks of the chain are transient and are subject to be discarded for a valid alternative while the blocks older than *k* blocks are immutable. The result is that forks cannot be longer than *k* blocks. Stability is a major guarantee that simplifies many use cases. For instance, the latest stable block is a convenient target for a participant that is joining the network.

It is important to note that OBFT only applies to the range of *k* transient blocks. Beyond this margin, the guarantees of the protocol no longer apply. Consequently, OBFT cannot recover block producers that were unable to follow block production from a majority of the participants for a sufficient amount of time. The chain itself degrades from such unresponsive parties due to the validity of OBFT relying on a minimum of two thirds of active and non-byzantine participants. The resilience to network split and delays stems from the time required by a subset of the block producers to produce an alternative version longer than *k* blocks.

Mempool

Each block producer maintains a pool of transactions pending to be added to a block. Transactions are kept in the pool for *u* (`sc-evm.pos.transaction-ttl-rounds-u`) slots. Once included in a block, transactions are removed from the pool. New transactions from other participants are added to the pool and re-broadcast to other peers. Transactions directly received by RPC methods are added to the pool and broadcast. Only transactions compatible with the local chain are considered.

Bootstrapping

A participant can join a running chain by requesting blocks for a time to find a *dense witness*. A *dense witness* is a subsequent chain of blocks of length equal to the number of block producers

that has at least two thirds of its slots filled. If provided with a recent *dense witness*, the newcomer can catch up and start following the chain from the latest block of the *dense witness*.

Details of implementation

Equality of chains

OBFT states that if a new valid chain is not strictly longer than the local chain, the local chain is kept. It also assumes that blocks are received one by one. However, implementation details make it possible that two or more valid and strictly longer chains than the local one are handled at the same time, therefore the first one is selected according to an increasing alphanumeric sort of hash of the last block. Eventually, upcoming blocks extend one of the alternatives until stabilization.

Slot events

A slot event sets the boundary between two slots. In the current implementation, slot events are time-based, as opposed to logical time where slot events would be propagated as state transitions via the network. The genesis block corresponds to slot number 0, hence slot events can easily be locally computed given the timestamp of genesis, the slot duration, and synchronized clocks (UTC time zone).

Bootstrapping

Bootstrapping is the process of connecting a client to the network. The newcomer needs to catch up with the latest state of the blockchain. First, a network stable target is defined out of the stable blocks of discovered peers via direct requests. Then, the missing data between the local stable (typically genesis) and the network stable target block is requested.

Due to the decentralized nature of the network, finding a target is not straightforward. The idea is to find a recent *dense witness*. A *dense witness* is a valid part of the chain with density of blocks higher than two thirds over *k* blocks. It is guaranteed by OBFT that such a part of the chain can be trusted as a target due to high participation of block producers. To compute the density, the client requests the current stable block from its peers and an ancestor *k* blocks from its parent. Out of these, the latest block of the most recent valid *dense witness* is selected as a target. Then the missing data is requested from peers and the local chain is populated. In the end, if the network's stable block has progressed, the procedure loops from the newly updated local stable block. Retries happen if an error occurs or if the local and target do not form a valid chain. The application stops retrying and crashes when an attempt on all valid candidates has been carried out.

There are two approaches to retrieving the missing data: *full-sync* or *fast-sync*. *Full-sync* requests all the blocks of the chain (via `GetFullBlocks` messages) and executes all transactions of the chain. *Fast-sync* requests all the blocks of the chain along with their receipts. Additionally, the entire Merkle Patricia Trie of the targeted latest stable block is downloaded (via `GetNodeData` messages). After *fast-sync* completes, it hands control over to *full-sync* mode to catch up potential progress during the synchronization. *Fast-sync* performs faster in most cases, because it retrieves data from the network instead of executing transactions, which is time consuming.

Initialization

Initialization refers to the creation of a new chain on a new network from scratch. The chain starts with a genesis block that contains pre-allocated accounts and a timestamp. The public keys of block producers are given in the `sc-evm.pos.standalone.validators` parameter.

The pitfall during initialization is that there is no dense chain to follow. Because of this, block producers tend to create their own forks and they may not find consensus before they reach *k* blocks and stabilize on a local unrecoverable fork. Some mechanisms are in place to compensate for this possibility. First of all, the client waits until there is at least one other peer connected. When the peer is connected, the client tries to bootstrap as explained in the previous section. If there is no chain to follow, the client tries to bootstrap with a relaxed density constraint: it considers chains with lower density than two thirds, but higher than a configurable threshold (`sc-evm.sync.enable-low-density-sync-fallback` and `sc-evm.sync.network-stable-header-resolver-low-density-threshold`). If there is still no chain to follow, but some chains with density lower than the configurable threshold are observed, the client stops as it considers the network forked too deeply. Finally, if the local stable is at genesis, and all connected peers have not stabilized a first block, the client starts issuing blocks from genesis.

These steps prioritize following chains rather than starting new ones, and give some time for network discovery before starting from scratch.

EVM

Example EVM sidechain client

The sidechain client contains an implementation of the EVM adhering to the Ethereum Yellow Paper [25]. It maintains feature parity up to the London hard fork [26], including EIP-1559 [27], EIP-3541 [28], EIP-3529 [29] and EIP-3198 [21] but excluding EIP-3554 [30] as the sidechain client uses the OBFT consensus protocol and does not have a notion of "difficulty".

Execution tracing

The sidechain client EVM supports execution tracing via the `debug_traceTransaction` [31] endpoint, which allows users to replay transactions as they were executed on the network.

Two types of traces are supported:
1. Basic [32]: generates raw EVM opcode traces by a given transaction hash.
2. JavaScript-based tracing [33]: the output trace is defined by a user-defined JavaScript expression.

Sidechain operation

Based on the descriptions of main chain data types, chain follower utilization, and sidechain client integration with the sidechain mode, it is possible to provide a detailed overview of the active and passive sidechain flows. The following sections reiterate and lay out the notions of generalized flows (both passive and active), and then provide detailed description of their application to the example EVM sidechain:

- Committee registration, rotation, and upload.
- Outgoing transactions: sidechain to main chain token transfers.
- Incoming transactions: main chain to sidechain token transfers.

Terms

- *Lock transaction*: sidechain transaction; a user locks their tokens in the sidechain bridge Solidity contract to claim the matching number of tokens on the main chain when an active flow is completed.
- *sc-evm-cli:* a command line application that facilitates the obtainment of registration signatures and lock transactions preparation
- *Plutus scripts*: used to denote a codebase of on-chain (Plutus) and off-chain (Cardano Transactions Library (CTL)) scripts used by end-users and chain administration to communicate with the main chain. CTL scripts (invoked with *ctl-main* in the examples) require Ogmios, ogmios-datum-cache, and ctl-server nodes to perform operations.
- *Relay actor*: an automated tool that relays sidechain epoch certificates to the main chain.
- *Sidechain bridge contract*: it is part of the Sidechain Certificate Creator, a Solidity contract that allows building ATMS signatures of committee handovers and outgoing transaction batches. It has to be included in the sidechain genesis file and be part of the genesis hash, which is part of sidechain parameters (explained in Registration).
- *Stability parameter k*: defined in Consensus protocol - OBFT.

The passive flow in the example EVM sidechain

Sidechain events occur on the main chain when users interact with a sidechain-specific set of Plutus scripts. This interaction results in submitting transactions to the main chain and some UTXOs being created on the main chain. For the general purpose passive flow, an EVM sidechain node uses a chain follower component to find out about the recent main chain events specific to the sidechain. A chain follower is queried for events (transactions) that happened on the main chain since the last event that the sidechain recorded in its ledger. When new events of interest are discovered on the main chain, a node translates them into EVM sidechain-specific transactions and attempts to add them to the next block it produces. When a block containing such transactions is added to the blockchain, the flow is completed.

When a node receives a sidechain block that contains transactions related to main chain operations of this sidechain, it validates them. It queries its chain follower for the discovered main chain transactions, and validates incoming blocks only if the chain follower returns matching entities.

As part of the operational characteristics of the sidechain passive flow, a settlement period is enforced by sidechain nodes. The length of this period depends on the main chain security parameter and slot length. During this period, a chain follower has already discovered incoming transactions, but the node will not yet add them to a sidechain block. Such transactions are categorized as ‘pending’.

The active flow in the example EVM sidechain

In the general active flow, users call a specific method (the *lock method*) of the sidechain’s Solidity bridge contract. The contract then adds entries to its outgoing transaction batches, which start at the beginning of the closed transaction batch phase of the previous epoch. One batch accumulates transactions for *12*k* slots. Each of the captured transactions will be signed by honest committee members during the *handover* phase of an epoch. It is possible to create an ATMS certificate as long as enough parties participate in this process. The EVM sidechain node exposes the signed

data via an RPC method, and a relay actor posts the certificate to main chain Plutus scripts. These scripts validate and then record sidechain data in the main chain.

During the *handover* phase, each node is supposed to add a specific *handover transaction* to the block it produces. A *handover transaction* contains the handover signature and, if there are any transactions stored in the current batch, the outgoing transactions' signature. The outgoing transactions' signature is obtained by signing a hash of a message made of the previous outgoing transactions' Merkle root, sidechain parameters (see Registration message), and the outgoing transactions' Merkle root. The *handover* signature is obtained by signing a hash of a message composed of the latest available transactions' batch Merkle root (the latest can be the current epoch), sidechain parameters, and sorted public keys of the next committee.

When computing the next committee participants, all nodes should use a main chain nonce, sidechain epoch number, and a set of registered block-producing candidates (described in What data is handled on the Cardano main chain?).

When nodes perform anything specific to the sidechain's operations involving the bridge Solidity contract, in the sidechain ledger there is an EVM logging mechanism used to record the appropriate events. The observed outcome of the *handover* transaction is a list of said EVM log events. The presence of these events is verified during block validation.

When a node receives a block from a *handover* phase slot, it:

1. **Executes** the transactions held in the block and obtains EVM message logs.
2. **Verifies** if these logs contain the event of signing the current transaction's batch by the block-producing node or if the logs are empty if the batch had no transactions.
3. **Checks** if a committee handover signed event was emitted. This event contains the block producer's public key, the handover signature, and the epoch number.

With this two-step mechanism in place, when each committee member has produced a valid block, it's possible to create a trivial-ATMS-style certificate that contains enough public keys and related signatures. These are sorted lexicographically and then hashed together.

The EVM sidechain node serves epoch signatures for every known sidechain epoch since sidechain initialization. The data served consists of the next committee public keys together with a set of handover signatures and, if a given epoch contains any, the outgoing transactions' Merkle root with outgoing transactions' batch signatures.

The served data enables a relay actor to update the main chain with events that happened in the sidechain. It is vital that some party updates the main chain to let users complete the active flow. A relay actor should periodically query a node for signatures that are yet to be added to the main chain, and use these Plutus scripts:

- `committee-hash` for updating committee members.
- `save-root` for updating transactions' Merkle root.

This makes the committee's UTXO available on the main chain. A user who performed a *lock* operation on the sidechain can query an EVM sidechain node's RPC method to obtain a Merkle proof of their transaction (covered in Transferring tokens from the sidechain, claiming). Then they can use this Merkle proof against the *claim* Plutus script that validates a given proof and performs a main chain transaction that mints *EVM sidechain native tokens* to a main chain recipient address.

Detailed description of observing registrations and committee rotation

Sidechain start

To form a working sidechain, enough eligible block producer candidates' registrations have to be recorded on the main chain. For each registration, a candidate should have a chain follower instance and an EVM sidechain node running (see Sidechain startup conditions). A node does not produce blocks before starting conditions are met. When they are, a node can produce blocks and determine the current committee. During the *handover* phase of a sidechain epoch, a node can compute the next committee and an *outgoing transaction* batch, and sign both. Without meeting these conditions, the sidechain cannot operate.

SPO registration

Each SPO is expected to run a cardano-node, a chain follower, and an EVM sidechain node. To register as a committee member candidate, an SPO needs to use their cold signing key (EDDSA) and a private key for the EVM sidechain (ECDSA). A candidate prepares signatures of sidechain parameters they wish to join alongside a UTXO that will be consumed for this registration to prevent a replay attack using both keys.

The *sc-evm-cli* tool can be used to generate such signatures:

command	`sc-evm-cli generate-signature` `--genesis-hash d8063cc6e907f497360ca50238af5c2e2a95a8869a2ce74ab3e75fe6c9dcabd0` `--sidechain-id 80` `--genesis-utxo 8aa190938f68a1bfc41e76117f24d9b5689a739fe824c2035c0b7a761496d12d#1` `--threshold-numerator 2` `--threshold-denominator 3` `--registration-utxo 846a166f6e1f9f597b2f9b866d44e01f296489f723cbb490a4d5d1ebd988b217#0` `--spo-signing-key <cold-signing-key-bytes-hex-string>` `--sidechain-signing-key <sidechain-private-key-bytes-hex-string>`
result	`{` `  "spoPublicKey": "6903d409fadc11d387085a59b53770c3d6fb4c482d54b713a89a430c6987d962",` `  "spoSignature": <bytes of the signature>,` `  "sidechainPublicKey":` `  "023e5054a7c0d33805a845e389794c82cbf1a01f5e16b4f14ff87911bef506f1ed",` `  "sidechainSignature": <bytes of the signature>` `}`

Code 10. Generate signature command

When the signatures are ready, a Plutus *register* script is used to post them on the main chain:

command	`ctl-main register` `--spo-public-key 6903d409fadc11d387085a59b53770c3d6fb4c482d54b713a89a430c6987d962` `--sidechain-public-key 023e5054a7c0d33805a845e389794c82cbf1a01f5e16b4f14ff87911bef506f1ed` `--spo-signature <bytes of the signature>` `--sidechain-signature <bytes of the signature>` `--registration-utxo 846a166f6e1f9f597b2f9b866d44e01f296489f723cbb490a4d5d1ebd988b217#0` `--payment-signing-key-file <path to the key owning the registration-utxo>` `--genesis-committee-hash-utxo` `929d26f89cd3fd7f9d78bc6706d8142e12838e4371069aa5c2752da1afd8b37c#0` `--sidechain-id 80` `--sidechain-genesis-hash f6fa0f49e862cd4f863bb6385c4f045f5bb63a4f1cf96967a4b40cd97e569bd5` `--threshold-numerator 2`

	`--threshold-denominator 3`

Code 11. Register command

Depending on the timing, a registration will either be processed by the sidechain module during the next main chain epoch, as long as the registration happened before the last `((2 * main_chain security parameter) / active slots coefficient)` slots of the current epoch or in the main chain epoch after that.

A note on invoking ctl-main

For each command, *ctl-main* requires a set of sidechain parameters (see Registration message):

- `genesis-committee-hash-utxo`
- `sidechain-genesis-hash`
- `sidechain-id`
- `threshold-numerator`
- `threshold-denominator.`

These parameters are covered in What data is handled on the Cardano main chain?. In the following examples, they will be referred to as *<sidechain parameters>*.

Also, please note that all ECDSA public keys that are parameters to the bridge Plutus scripts are given in a compressed form.

Committee calculation

The pseudo-random committee selection algorithm (see Sidechain-aware consensus) uses the following parameters as inputs:

- *stake delegation distribution* of each candidate.
- *main chain epoch nonce* together with *sidechain epoch number* as randomness seed

and outputs a subset of *input candidates* to be used as the next block-producing committee. When calculating the committee for a particular sidechain epoch, a main chain nonce and the stake delegation distribution for all candidates are needed. Ideally, the sidechain would use the newest stakepool delegation distribution (SDD) and nonce available (latest stable epoch). Due to a limitation of the currently used chain follower, both SDD and the epoch nonce are not readily available right at the beginning of a new main chain epoch. To be able to proceed, the sidechain needs to depend on available data, therefore it uses the SDD and nonce from the epoch directly preceding the 'latest stable epoch'.

Main chain committee awareness

To be able to process messages coming from the sidechain, the main chain has to store committee members' data. This allows the bridge Plutus scripts to validate the signatures of messages posted to them. For any sidechain, the initial committee has to be submitted to the main chain using *init* Plutus script by providing a set of (sidechain) public keys of the initial committee, an epoch number previous to the first epoch that the sidechain was operating in, and sidechain parameters. This operation consumes *genesis-committee-utxo* and does not involve signatures, so it requires a signing-key parameter that owns this UTXO (usually this will be the entity that is starting the sidechain). This operation happens only once.

command	`ctl-main -- init` `--payment-signing-key-file /path/to/chain-genesis-committee-utxo-owner.skey` `--committee-pub-key 02e4ae11784911a2e8e6a8a173b600c7bed83fec80dd5b8fbb89b7106fe1945143` `--committee-pub-key 039e0a4b7489317d47cc1c2140f2f2b01f0445b0eb0adb331e6101686f8b243a6d`

```
--committee-pub-key 03c24db9b1c50dc53d239a04e64077c294eca130a982914f7bb139ef666e289b67
--sidechain-epoch 3489
<sidechain parameters>
```

Code 12. Fist committee init command

The EVM sidechain node serves RPC methods that allow a relay actor to update the main chain with *committee handover* messages:

- `sidechain_getSignaturesToUpload()`: returns an array of objects, containing:
  - `epoch`: epoch number pending committee handover (if the *committee-hash* script wasn't invoked on the main chain)
  - `rootHashes`: an array that contains hex-encoded bytes of Merkle roots pending insertion (if the *save-root* script wasn't invoked on the main chain).
- `sidechain_getEpochSignatures(sidechain_epoch)`: returns all details required to perform *save-root* and *committee-hash*.

A relay actor uses these two methods to run *committee-hash* scripts that should be used to hand over a committee from the current epoch to the next one.

Input parameters are:

- Sidechain public keys of the next committee.
- Pairs of current committee public keys and signatures.
- The epoch number and previous Merkle root.

The on-chain part of the *committee-hash* script validates that there are at least `(numerator*committee_size)/denominator` valid signatures (for the message made of sidechain parameters, sidechain epoch, next committee public keys, and previous Merkle root) supplied and whether or not they belong to the current committee. If yes, then it will store the next committee keys and epoch on the main chain until the next handover signatures in *save-root* and *committee-hash* messages are validated against the new committee public keys.

```
command ctl-main committee-hash
        --payment-signing-key-file /secrets/relay.skey
        --committee-pub-key-and-signature <public key 1>:<signature 1>
        --committee-pub-key-and-signature <public key 2>:<signature 2>
        --committee-pub-key-and-signature <public key 3>:<signature 3>
        --new-committee-pub-key <next committee public key 1>
        --new-committee-pub-key <next committee public key 2>
        --new-committee-pub-key <next committee public key 3>
        --previous-merkle-root <hex string of previous Merkle root>
        <sidechain parameters>
```

Code 13. Committee hash upload command

Signatures from `sidechain_getEpochSignatures`, and previously submitted to the sidechain by nodes, are obtained for each epoch in a way that the outgoing transaction batch is signed first, and then its Merkle root is included as part of the committee handover message. This enforces a particular order of invoking on-mainchain scripts:

- `Save-root` message for a given epoch.
- `Committee-hash` message. If there were no outgoing transactions in any of the previous sidechain epochs, the previous Merkle root will not be present in parameters.

Sidechain committee rotation vs main chain committee awareness

EVM sidechain nodes use only registration data and their own configuration parameters to compute a committee for each sidechain epoch. No other interaction is required with the main chain. This means they can progress with the epoch cadence at their own pace. However, the main chain part has to be made aware of the committee for each sidechain epoch starting with the *init* committee, to allow interactions like *save-root* and *claim*.

Detailed description of transferring tokens to the main chain

To transfer tokens to the main chain, a user submits a signed transaction that calls *lock(recipient)* Solidity method, where:

- `recipient` is a main chain address that will receive tokens.
- `transaction` *receiving address* is an address of bridge contract *0x696f686b2e6d616d626100000000000000000000*.
- `transaction value` is the expected value to be transferred. It has to be a positive multiple of 10^9 (because of the conversion rate, refer to Detailed description of transferring tokens to sidechain).

Lock validates the transaction amount, records the transaction in the transaction batch, in the smart contract storage, and emits a *tokens locked* EVM event (which contains transaction sender, value, recipient, and index details within the batch).

If the current epoch phase is *regular*, the transaction is appended to the current transaction batch, otherwise into the next epoch's transaction batch.

Signing epoch transaction batch

In the *handover* phase, block-producing nodes construct *sign* transactions and *committee handover* transactions:

- Call Solidity `getEpochTransactions(current_epoch)` to get the transaction batch.
- Call Solidity `getPreviousMerkleRoot(current_epoch)` to get the previously uploaded Merkle root.
- Create new transactions Merkle root from the fetched batch, previous Merkle root, and sidechain parameters
- Use the new Merkle root to prepare new committee signatures or use the previous Merkle root if there were no outgoing transactions.
- Call Solidity *sign* to create a transaction that stores a new Merkle root and its signature. Besides that, the signature for a new committee is stored. This process emits '*handover signed*' and '*outgoing transactions signed*' EVM logs.

Merkle root construction: to create an epoch transactions' Merkle tree, a new leaf is created for each transaction. Leaf data is obtained by taking CBOR representation of datum encoding of the transaction index (within the batch), value, recipient, and the previous Merkle root [34].

In the example EVM sidechain, a block-producing node is required to include handover-specific transactions whenever it has the opportunity to create a block. This requirement is part of an incoming block validation and will result in blocks being rejected if not honored.

Updating the main chain and claiming tokens

The ability to upload handover-specific data to the main chain, which allows users to *claim* their tokens, hinges on the following RPC methods served by the EVM sidechain node [35]:

- `sidechain_getOutgoingTransactions(sidechain_epoch)`: returns an object with a *transactions* array, in which there is an object containing details about a singular outgoing transaction: *recipient, value,* and *tx_index*. This method can be used to inspect a user's *transaction id*.
- `sidechain_getOutgoingTxMerkleProof(sidechain_epoch, tx_index)`: returns an object with a *proof* field that holds hex-encoded Merkle proof bytes of a given transaction. The data encodes both the proof and the data of the transaction. Note: The data for this endpoint is available only *after* the handover mechanism is done for a particular epoch.
- `sidechain_getSignaturesToUpload()`: returns an array of objects. Each of them contains *epoch* numbers with pending committee handover messages (*committee-hash* wasn't posted), and a *rootHashes* array that contains hex-encoded bytes of Merkle roots pending insertion (*save-root* wasn't posted).
- `sidechain_getEpochSignatures(sidechain_epoch)`: returns all details required to perform *save-root* and *committee-hash*.

At the beginning of a sidechain epoch, a relay actor uses `sidechain_getSignaturesToUpload` and `sidechain_getEpochSignatures` methods to gather required data and run *save-root* and then *committee-hash* Plutus scripts.

command	`ctl-main save-root` `--payment-signing-key-file /secrets/relay.skey` `--merkle-root 26ab4cf08c534d7c5de43d941eb44db0594dcd5e648154113b96211b9fb67b1e` `--previous-merkle-root 4bf487bad864561eb753222e5f60828c12c695684946ffca3e05eef427dd64d9` `--committee-pub-key-and-signature` `023c04939acf0e63bcc97188b6acebd3ce16eed9682dae13721eb5fd8f453d6636:f9e32af2c8e88e270bb74d` `cbf084d2e4077cf54faadd8d7c07166b13485f4bce32625349e04a8d7293d3c91516133e2c8efbafdb0c19fc6` `c76fc6367a9bbc58d` `--committee-pub-key-and-signature` `023e5054a7c0d33805a845e389794c82cbf1a01f5e16b4f14ff87911bef506f1ed:cdf98234c89ead2f317cf9` `db22e193e6c9dc535cd6e97c5b9cf351e59ea6bd7c6b11ec2a0a477aafc4dd85d793526d3de9ecd9ce3149969` `eb289c80ff4f123ee` `--committee-pub-key-and-signature` `03e3a4992391a50aa018c6308f028749475107ce442ad7107cfd20b2d3586cf995:ce82d859a109bc220ac20e` `07120815e71249f546247233afff222d04434e4eee367fbb7a6c94c48dcede3385aaa2221c20f4e5a7b5b7c22` `d8d8bd7680a49bfbd'` `<sidechain parameters>`
result	`{` `  'endpoint': 'CommitteeHash',` `  'transactionId': '7b285de3ee26bee51325f142b5442dc553979a68fea3cdc5f75a7d44ec2c4d9f'` `}`

Code 14. Save root command

The *save-root* script validates signatures with the current committee public keys and saves the Merkle root on the main chain.

From the moment this data is stored on the main chain, users can request their tokens with the *claim* script. On-chain code validates the given Merkle proof against stored Merkle roots and, if

successful, grants the user their requested sidechain tokens. One Merkle proof can be used at most once.

command	ctl-main claim --payment-signing-key-file /path/to/recipient-signing-key --combined-proof d8799fd8799f001a0001d4c0581d606e9d4f6a3f900f7b510b27f29d8e79c550686ce093946b23b3d1828ed87 99f58206ed10e8180f6e3ecbb19c060ac2dbf51eccc33569b91e2047a58be9a9e84ffd2ffff80ff <sidechain parameters>

Code 15. Claim command

Sidechain to main chain token transfer flow is complete.

Detailed description of transferring tokens to sidechain

The *mint* Plutus script is used to mark a specified number of EVM sidechain tokens to be transferred from the main chain to the sidechain. The mechanism uses the *burn* action available in minting policies. The script posts a burn transaction for the sidechain native token to the main chain, which results in adding the UTXO (for the Plutus *mint* script address) that stores the recipient address in the redeemer data of the transaction. The script validates that the amount is positive and that there are enough EVM sidechain tokens available to the key signing the transactions.

command	ctl-main burn --payment-signing-key-file <path to user signing key> --amount 251 --recipient aabbeeccddaabbeeccddaabbeeccddaabbeeccdd <sidechain parameters>

Code 16. Burn command

The EVM sidechain node is configured with minting native token policy id and asset name. When a node is about to produce a new block, it queries the Solidity bridge contract for the last processed mint transaction and then queries the chain follower component for mint transactions newer than the last processed one. These transactions need to be considered finalized on the main chain. The node issues a transaction calling the Solidity bridge contract's *unlock* function, passing the main chain transaction ID, recipient, and the amount discovered in the redeemer data of the burn transaction coming from the main chain. The Solidity bridge contract stores the transaction ID as the new last processed mint transaction id and transfers tokens to the recipient address. The Solidity bridge contract also emits an `IncomingTransactionHandledEvent` EVM log event to allow block validation.

Because Cardano has a small numeric range for native tokens (maximum is 2^63) and EVM uses a much bigger range (maximum is 2^256), a conversion rate is applied. One EVM sidechain native token then stands for 10^9 tokens on the sidechain (see Transferring tokens to the sidechain, Note about the conversion rate).

During block validation, sidechain nodes query the Solidity bridge contract and the chain follower to compute expected incoming transactions and compare them to log events emitted during the execution of incoming blocks. In case of discrepancy, an incoming block is rejected.

Settlement period

Only the stable part of the main chain is taken into account when producing and validating blocks. However, it is possible for a user to view which incoming transactions an EVM sidechain node is

aware of and which it will process, with blocks recorded in specific burns. The RPC method `sidechain_getPendingTransactions` returns a JSON object containing two arrays: *pending* and *queued. Pending* items are incoming transactions discovered in a part of the chain that is not stable yet. *Queued* items are in the stable part of the chain and should be included in the next produced sidechain block.

Summary

This section provides in-depth information about the main components of an example EVM sidechain, including the main chain Plutus scripts, the role of Cardano db-sync as a chain follower, the integration of the follower into the sidechain module, the modification of the client to support sidechain mode, the function of the relay actor, and the implementation of passive and active flows.

6 Conclusion

The goal of the sidechain team was to find a way to transfer value between Cardano and another blockchain without requiring changes to Cardano. The proposed solution, based on the “Proof of Stake sidechains” paper [2], involves establishing a communication pattern between a main chain and a sidechain, with the sidechain adapted to follow the main chain. This lopsided relationship eliminates the need for committees to manage the process of transferring data between chains, as these are collapsed into the sidechain protocol and managed by the block-producing committee of the sidechain. The proposed solution introduces several key components that enable sidechain flows: a set of Plutus scripts, a chain follower, a sidechain module, a sidechain-aware consensus protocol, and a relay actor. With these components in place, it is possible to build a Cardano sidechain.

While working on an example of a Cardano sidechain, the team also began efforts to generalize the building blocks of the solution. The goal of this work is to create a Cardano sidechain toolkit, which would allow anyone to build a customized solution that meets their specific needs and requirements.

As an example of the principles of the Cardano sidechain in action, an EVM blockchain client was chosen and adapted. Instead of PoW, an OBFT-based consensus protocol was implemented and modified to run in sidechain mode. A sidechain module integrated with Cardano db-sync was built, and a set of Plutus scripts was created to maintain the sidechain on the main chain. Both passive and active flows were implemented, and a functioning testnet was set up to demonstrate how all of these concepts and decisions come together to solve the problem of transferring value between Cardano and a sidechain.

The example EVM sidechain is the first iteration of the Cardano sidechain proposal. It addresses the main problem of how to transfer value between sidechain and main chain. The design does not introduce additional trust into the system, as the sidechain and main chain are solely responsible for ensuring the scalability, decentralization, and security of the solution. The implementation establishes the initial versions of basic structures, scripts, values, and utilities that are necessary to enable iteration on the design and the solution.

The example EVM sidechain demonstrates the feasibility of integrating a fully foreign computational model (account based with Solidity) into the Cardano platform using existing tools

and mechanisms. One notable aspect of this solution is its potential to engage the Cardano staking pool community and incentivize their participation in operating sidechains.

This exploration of the sidechain problem space has provided the team with the opportunity to build an actual solution using existing or creating new tools. It has also identified new pain points and potential improvements, so that future iterations can be more robust and generalized. Presenting a working solution to the Cardano community early in the design and implementation process gives the community ample opportunity to provide feedback, voice concerns, and inform the team about use cases not currently covered by the design. As mentioned, this is the first iteration of the proposal, and the team's goal moving forward is to learn from and respond to the needs of future sidechain creators.

The proposed solution is not yet ideal. The team is aware of the cost of waiting for main chain finality and sees this as an area for improvement. There are also concerns about the selection and maintenance of the sidechain committee in terms of both security and liveness of the chain. Additionally, it is understood that the current set of Plutus scripts only provides one model for maintaining a sidechain, and expanding this area is a goal for the sidechain team. It is also understood that Cardano db-sync is not a perfect fit for the Cardano sidechain use case, as it lacks the desired flexibility.

Overall, this document sets out the concept of a sidechain, proposes a design for a Cardano sidechain, identifies the key components of a solution, highlights the rationale for the proposed design, and introduces the concept of a toolkit for creating custom sidechains. It also provides an in-depth look at how the example EVM sidechain is constructed and the reasoning behind the design choices.

References

[1] "Ouroboros-BFT: A Simple Byzantine Fault Tolerant Consensus ...." 26 Nov. 2018, https://eprint.iacr.org/2018/1049.pdf.

[2] "Proof-of-Stake Sidechains" 18 Dec. 2018, https://eprint.iacr.org/2018/1239.pdf

[3] "Hydra Head protocol: an open source solution for scalability." 16 Dec. 2022, https://cardanofoundation.org/en/news/hydra-head-protocol-an-open-source-solution-for-scalability/.

[4] "cardano-foundation/CIPs - GitHub." https://github.com/cardano-foundation/CIPs

[5] "CIP-0381 - Cardano Developer Portal." 11 Feb. 2022, https://developers.cardano.org/docs/governance/cardano-improvement-proposals/cip-0381/

[6] "CIP 9 - Protocol Parameters - Cardano Improvement Proposals." https://cips.cardano.org/cips/cip9/

[7] "Learn about native tokens - Cardano Docs." https://docs.cardano.org/native-tokens/learn/

[8] "Plutus scripts - Cardano Docs." https://docs.cardano.org/plutus/plutus-scripts

[9] "input-output-hk/cardano-db-sync - GitHub." https://github.com/input-output-hk/cardano-db-sync

[10] "(Re)introduction into Cardano." 21 Sep. 2022, https://developers.cardano.org/docs/stake-pool-course/introduction-to-cardano/

[11] "Staking is the bedrock of Cardano - IOHK Blog." 27 Jul. 2022, https://iohk.io/blog/posts/2022/07/28/staking-is-the-bedrock-of-cardano/

[12] "A Scala based client for Ethereum-like Blockchains." https://github.com/input-output-hk/mantis

[13] "Plutonomicon/cardano-transaction-lib: A Purescript library ... - GitHub." https://github.com/Plutonomicon/cardano-transaction-lib

[14] "Plutus scripts - Cardano Docs." https://docs.cardano.org/plutus/plutus-scripts

[15] "The Plutus platform - YouTube." 16 Jul. 2020, https://www.youtube.com/watch?v=usMPt8KpBeI

[16] “CIP 31 - Reference inputs”: https://cips.cardano.org/cips/cip31/

[17] "plutonomicon/stick-breaking-set.md at main - GitHub." https://github.com/Plutonomicon/plutonomicon/blob/main/stick-breaking-set.md

[18] "On-Chain Association List With Constant Time Insert/Removal" https://mlabs.slab.com/public/posts/on-chain-association-list-with-constant-time-insert-removal-sh8z2xzy

[19] "Collection of APIs provided by Ethereum execution layer clients." https://github.com/ethereum/execution-apis

[20] "Ethash - Ethereum.org." https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/mining-algorithms/ethash/

[21] “EIP-3198: BASEFEE opcode.” https://eips.ethereum.org/EIPS/eip-3198

[22] "Ethereum Virtual Machine Opcodes." 29 Jul. 2021, https://ethervm.io/

[23] "Patricia Merkle Trees | ethereum.org." https://ethereum.org/en/developers/docs/data-structures-and-encoding/patricia-merkle-trie/

[24] "Cardano Docs - Cardano.org., Cardano Sidechains section." https://docs.cardano.org/

[25] "Ethereum Yellow Paper - GitHub Pages." https://ethereum.github.io/yellowpaper/paper.pdf

[26] "History and Forks of Ethereum - ethereum.org." https://ethereum.org/en/history/

[27] "EIP-1559: Fee market change for ETH 1.0 chain." 13 Apr. 2019, https://eips.ethereum.org/EIPS/eip-1559

[28] "EIP-3541: Reject new contract code starting with the 0xEF byte." 16 Mar. 2021, https://eips.ethereum.org/EIPS/eip-3541

[29] "EIP-3529: Reduction in refunds - Ethereum Improvement Proposals." 22 Apr. 2021, https://eips.ethereum.org/EIPS/eip-3529

[30] "EIP-3554: Difficulty Bomb Delay to December 2021." 6 May. 2021, https://eips.ethereum.org/EIPS/eip-3554

[31] "debug Namespace | go-ethereum." 9 Dec. 2022, https://geth.ethereum.org/docs/interacting-with-geth/rpc/ns-debug

[32] "Basic traces | Go Ethereum." https://geth.ethereum.org/docs/evm-tracing/basic-traces

[33] "Custom EVM tracer | go-ethereum." 30 Nov. 2022, https://geth.ethereum.org/docs/developers/evm-tracing/custom-tracer

[34] "Providing Authentication and Integrity in Outsourced Databases ...." https://people.eecs.berkeley.edu/~raluca/cs261-f15/readings/merkleodb.pdf

[35] "OPEN-RPC Playground" https://playground.open-rpc.org/?schemaUrl=http://faucet.sidechain.evmtestnet.iohkdev.io

[36] "GearBox: Optimal-size Shard Committees by Leveraging the Safety ...." 2 Mar. 2021, https://eprint.iacr.org/2021/211.