Obscuro

Confidential Smart Contracts for Ethereum

> /obˈskuː.roː/
>
> 1. (Latin) To conceal, hide;
> 2. A spell used to conjure a blindfold over the eyes of the target.

V0.10.0, November 2021 *Note: This document is under development, so please check for updates.*

James Carlyle, Tudor Malene, Cais Manai, Neal Shah, Gavin Thomas, Roger Willis; with significant additional contributors.

Abstract

We present Obscuro, a decentralised Ethereum Layer 2 Rollup protocol designed to achieve data confidentiality, computational privacy and prevent Maximal Extractable Value (MEV) by leveraging hardware-based Trusted Execution Environments (TEE).

The design of Obscuro ensures that hardware manufacturers do not have to be trusted for the safety of the ledger. If one manufacturer turns malicious or there is a breach in the TEE technology, the protocol falls back to the behaviour of a public blockchain that preserves the ledger's integrity but makes the transactions public.

The design also focuses on preserving privacy for the limited period when it matters most, which removes the need for a privacy technique that is robust against all adversaries in perpetuity.

Obscuro sits in what we believe is a sweet spot between the existing rollup-based L2 offerings Optimistic and ZK Rollups. The use of confidential computing techniques coupled with economic incentives allows Obscuro to retain the performance and programming model simplicity of Optimistic rollups, and on top of that attain confidentiality, short withdrawal periods, and address MEV.

Motivation

Public blockchain networks have experienced a period of strong growth in 2020-2021, with new use-cases for smart contracts encompassing the finance and art worlds. Decentralised Finance (DeFi) has seen enormous inflows of capital, with the top applications seeing the equivalent of $10 billion (summer 2021) of liquidity added, and this has helped push the overall capitalisation of cryptoassets above $2 trillion (summer 2021). Meanwhile, *Non-Fungible Tokens* (NFTs) surged in value to $10 billion (autumn 2021).

Public blockchains rely on the entire network seeing all transactions in order to be able to validate them and secure the network. This makes them *transparency engines*. Unfortunately, this creates a front-running issue, known as *Maximal Extractable Value* (MEV), where miners or stakers and block proposers may steal value by observing user transactions and then preempting them. For example, a miner or bot may observe a user's desire to buy an asset at market price with an automated market maker, insert their purchase ahead in the processing queue by paying a higher transaction fee, causing the price to go up for the user, and then sell their purchase at a higher price and extract a profit from the user.

By some estimates, front-running was valued at $1.4 billion annually in early 2021. This means users of public blockchain networks are not deriving the full economic benefits of the technology. In addition, the transparent nature of the technology makes them inappropriate for many commercial and personal use-cases, where the confidential nature of interactions and deals should be maintained.

Obscuro is a decentralised Ethereum Layer 2 Rollup protocol designed to address the above problems, introduce new use-cases, and unlock blockchain technology's full potential and economic advantages.

Differentiators

- Obscuro leverages Ethereum, a public blockchain with the greatest adoption, legitimacy, security, and liquidity, as a base layer to handle security and data availability and manage the inflow and outflow of value.
- Obscuro keeps all transactions and the internal state of application contracts encrypted and hidden, and so provides a credible solution to MEV.
- By providing an *Ethereum Virtual Machine* (EVM) compatible VM, deploying existing contracts to Obscuro with minimal change may be possible.
- Obscuro is trustless and decentralised. It takes processing from the Ethereum Layer-1 (L1) and allows lower transaction costs similar to other Layer-2 (L2) networks.
- Obscuro leverages TEEs for privacy but not for integrity and is not affected by the limitations of hardware-based confidential computing.
- Obscuro guarantees quick finality by synchronising the publishing of rollups to the cadence of the L1 blocks.
- Obscuro introduces a novel mechanism to allow application developers to balance the need for user data privacy (and MEV prevention) with the need to deter long-term illegal behaviour.

Challenges

As well as preserving the confidentiality of user data, the other main goals of this protocol are to be fully permissionless, decentralised and a generic smart-contract execution engine compatible to the greatest extent with the EVM.

In this section, we enumerate the key challenges we faced when designing the Obscuro protocol.

- Relying on hardware-based TEEs for applications where significant value depends on the security of the hardware poses several challenges. A system designed to manage value should not allow an attacker capable of compromising secure hardware to take ownership of the value under any circumstances. In other words, the ledger's integrity should not depend on TEEs being 100% hack-proof or the hardware manufacturer being 100% trustworthy. Obscuro uses the security of Ethereum combined with game theory to detect and correct eventual TEE hacks.

- A system where everything is encrypted all the time is not usable. There must be a way for users to query their data or prove it to third parties. Additionally, an existing application contract that reveals internal state (such as the balanceOf(address) function of the ERC-20 standard used to look up anyone's holding) need to be considered carefully; since while Obscuro would prevent the state of the contract from being visible, the functions might not.

- Another critical challenge for this protocol is the prevention of MEV. Because user transactions and execution are not visible to Obscuro nodes, one might naively assume that this problem is solved. Unfortunately, that is not strictly true since aggregators might gain useful information through side-channels and use that to extract value. For example, an aggregator might own some accounts and submit transactions to them in critical moments and then query for results. Obscuro addresses this by introducing delays in key moments to prevent aggregators from performing replay-attacks which can generally be used for side-channels.
- A privacy-preserving platform should consider illegal usage and design mechanisms to help application developers avoid and prevent it. An important insight in this direction is that the value of confidentiality decays over time, to the point where transactions may just be of historical interest. For many transactions involving value, it is critical that they are not public when processed and cannot be front-run, but for others, they are price-sensitive for a longer period. Obscuro uses this insight and implements a flexible policy for delayed transaction revelation. The knowledge that transactions become public in the future is a deterrent for users to engage in criminal behaviour because law-enforcement agencies will eventually catch up. Alternative options have been considered.
- One crucial challenge of such a system is ensuring that some catastrophic event cannot leave all the value locked. The mechanism that prevents this is covered in the Threat-Model analysis.
- High transaction fees are one of the main barriers to entry for Ethereum. Obscuro addresses this by introducing a novel approach to calculate fees based on the actual costs of the running nodes.

Technical Background

This section briefly covers the key technologies on which Obscuro relies.

We recommend reading through the "Trusted Execution Environment" section because it introduces concepts and notations used throughout the paper.

Ethereum

Ethereum is a public, transparent, permissionless blockchain system and network of nodes, supporting account-based smart contracts, where business logic can be deployed as code to create an immutable and uncensorable contract which can hold and control the flow of value. The Ethereum mainnet went live in 2015 and is the most mature and adopted smart contract system. Read more on the official website.

Trusted Execution Environment

A TEE is a secure area of a central processor or CPU. It guarantees code and data loaded inside to be protected with respect to confidentiality and integrity as it is processed. Obscuro focuses initially on Intel's SGX, based on the team's 5 years of experience developing a confidential computing product with it. The TEE data cannot be read or processed outside the enclave, including processes running at higher privilege levels in the same host.

An SGX-capable CPU has two device root keys that are fused into it by the manufacturer, the *Root Provisioning Key* (RPK) and the *Root Sealing Key* (RSK). The RPK is known to Intel and used to prove a CPU is genuine via remote attestation and the RSK is not known to any entity outside the CPU. These keys can be used to create other CPU specific keys. In this whitepaper, we will refer to them as the *Enclave Key* (EK). Processes and users outside the enclave encrypt data that is only meant for the enclave using keys generated inside the enclave. When the enclave wishes to store data, it is again encrypted so that the host (the server which stores the data) is not able to see it.

Attestation allows user verification that the enclave is running on a genuine SGX capable CPU that is properly-patched, and the application running inside the enclave matches a particular codebase and is un-tampered before the user shares confidential data with it. This allows the user or someone trusted by them to audit the code of the application in advance and know for sure that only that code will see that data.

In Obscuro's case, the SGX application is a virtual machine largely compatible with the EVM, allowing execution of existing Ethereum smart contracts, along with the rollup functionality necessary to interact with the L1 contract.

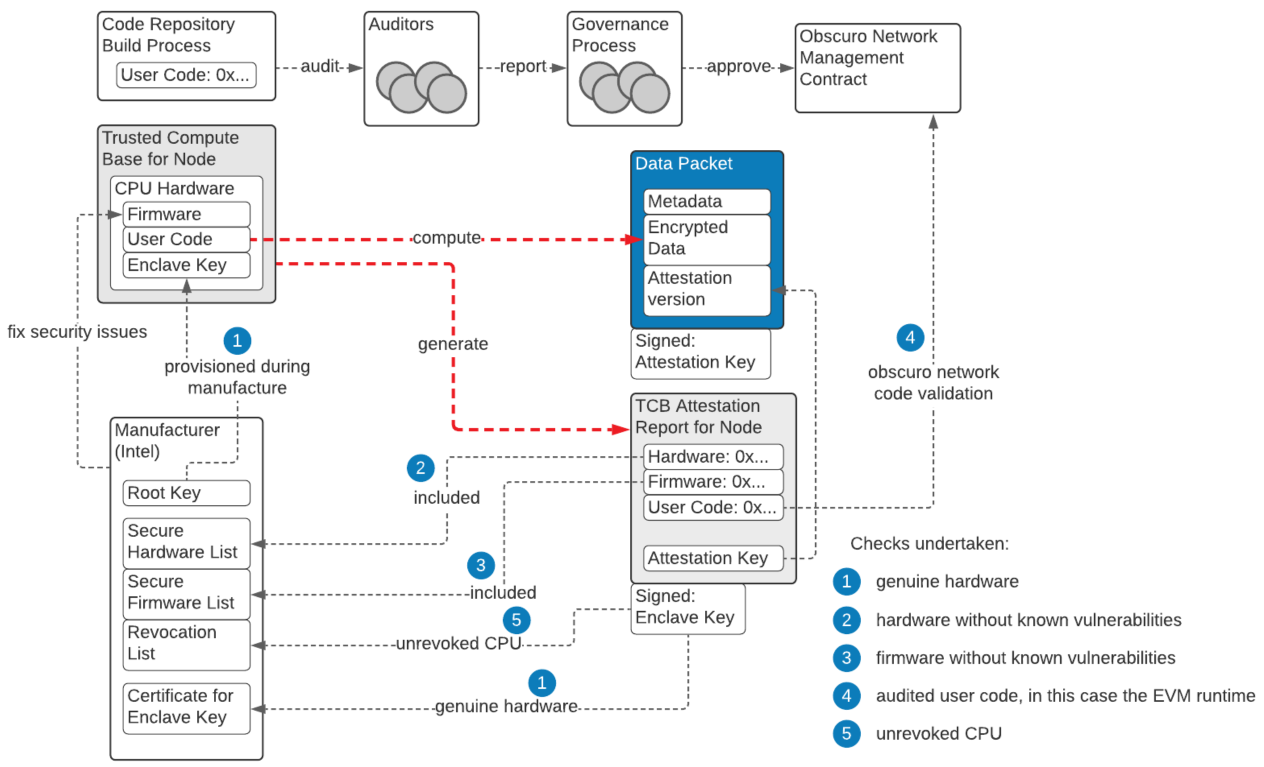
The *Trusted Computing Base* (TCB) is defined as the set of computing technologies that must be working correctly and not be malicious or compromised for a security system to operate. The TCB is composed of the hardware TCB (the CPU) and the software TCB (the CPU microcode and the application). Attestation provides to the Verifier a report containing the details about all the components of a TCB, like CPU type, the SGX security version number (CPUSVN) and the version of the application.

An attestation report that was deemed as secure could become insecure if a vulnerability is disclosed. At that moment, the system needs to be re-secured, a process which is called TCB recovery.

This whitepaper refers to the *Attestation Report* (AR) as a generic object that describes the TCB and also contains an encryption key referred to as the *Attestation Key* (AK), and as *Attestation Constraints* (AC) to a set of constraints that a report must satisfy to be considered secure at a point in time. The constraints will change over time as vulnerabilities are discovered, the software is upgraded with new features or to keep up with the evolution of the EVM. The Obscuro nodes will have to upgrade to continue participating in the network.

Any message originating from an enclave can be signed with the AK. This is a guarantee for the recipient that it must have originated only inside a valid enclave.

The diagram below is a conceptual high-level overview of the mechanism by which a TEE manufacturer and a group of security auditors propagate trust to the output of the computing performed inside the TEE.



A signature from the EK attests that a signed data packet originates from a genuine CPU. That is not enough for the output of typical confidential computing use cases, as clients have to know what program runs inside the CPU and what firmware.

To solve this problem, the TEE generates a new key (the AK) derived from the RSK, which is then included in the attestation report, together with the software and hardware versions, and signs this report with the EK.

By this mechanism, data packets signed with the AK include the trust from the genuine CPU and the hash of the program attested by the group of auditors.

Rollups

The two approaches to scaling L1 blockchains are to improve the capacity of the blockchain, or move processing away from it but tie back to it.

The first approach can make the blockchain more centralised, as the cost of node infrastructure increases (limiting the number of participants able to afford it) or the number of nodes involved in consensus decreases. A variation splits the accounts into shards, allowing validation to happen in parallel, and this is the current approach on the Eth2 roadmap.

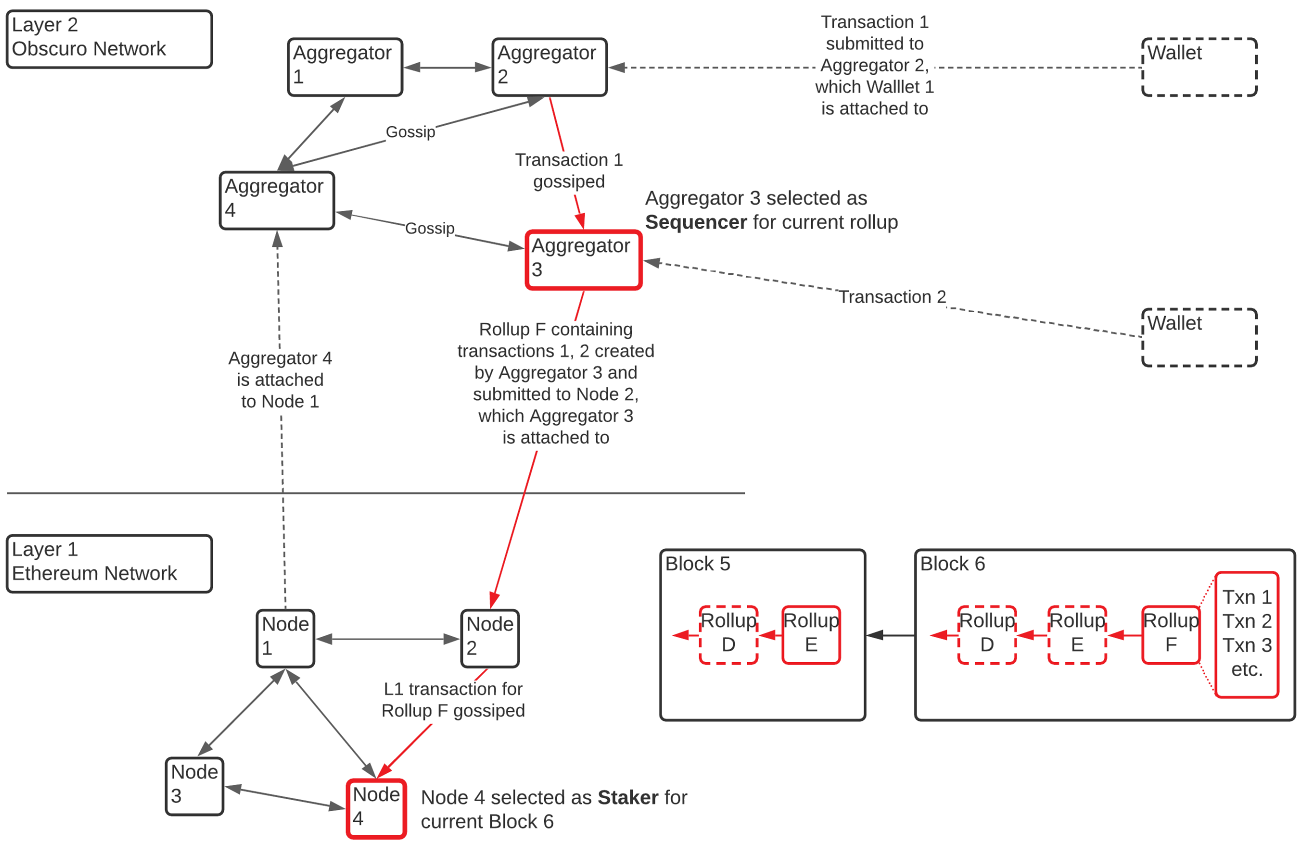
The second approach is to allow users to engage with contracts on a second-layer network of nodes, where the majority of the processing work is undertaken. One example of the second approach is *rollups*, where the L2 transactions are verified and posted in compressed form in a single rollup transaction to the L1 blockchain. There is an L1 contract which processes deposits and withdrawals. In zero-knowledge rollups, L1 nodes can undertake a lightweight process of verification of the correctness of activity on the L2 network, whereas in optimistic rollups, the L2 transactions submitted are assumed to be correct, but another L2 node may disprove them during a challenge window.

High Level Design

Obscuro is designed as an L2 protocol, where user activity is moved *off-chain* from the L1, and it follows the increasingly common rollup pattern to store transaction data on the L1 chain to achieve censorship-resistant data availability. This is leading to proposals to reduce calldata storage costs on Ethereum. Most rollup implementations exist to provide scalability for L1 networks, but the prime objective of Obscuro is to provide confidentiality. The rollups contain the entire encrypted transaction data.

L2 networks have a unidirectional dependency on an L1 network: while the L2 network relies on the L1 network to provide an immutable and public record of transaction data and to provide censorship resistance, liveness and availability, the L1 network is unaware of any individual L2 network. L2 submitted rollups are just normal L1 transactions.

The following diagram shows the interactions between the two decentralised networks, Ethereum (L1) and Obscuro (L2): Obscuro is formed of Nodes called Aggregators, who compete to process user transactions, roll them up, and submit for inclusion in Ethereum blocks. Ethereum, through its protocol, leverages its own nodes to produce Ethereum blocks containing, amongst other things, the submitted Obscuro rollups.


Layer 2
Obscuro Network
Aggregator 1
Aggregator 2
Aggregator 4
Aggregator 3
Gossip
Gossip
Transaction 1 submitted to Aggregator 2, which Walllet 1 is attached to
Wallet
Transaction 1 gossiped
Aggregator 3 selected as **Sequencer** for current rollup
Transaction 2
Wallet
Aggregator 4 is attached to Node 1
Rollup F containing transactions 1, 2 created by Aggregator 3 and submitted to Node 2, which Aggregator 3 is attached to
Layer 1
Ethereum Network
Node 1
Node 2
Node 3
Node 4
L1 transaction for Rollup F gossiped
Node 4 selected as **Staker** for current Block 6
Block 5
Rollup D
Rollup E
Block 6
Rollup D
Rollup E
Rollup F
Txn 1
Txn 2
Txn 3
etc.

On the bottom right, this diagram also depicts the state of a simple rollup chain as it is found in the sequential L1 blocks.

L1 Network

On the L1 network there are several regular Ethereum contracts, referred to as Management Contracts.

Note: the L1 design is covered in more detail in L1 Contracts.

Network Management

This contract is the gatekeeper for the protocol. Any Obscuro node wishing to join the network will have to interact with this contract and prove it is valid. This contract will also manage the TEE attestation requirements and will be able to verify attestation reports.

It will also manage the stake of the participants able to submit rollups known as Aggregators.

Note: The stake is a piece of the game-theory puzzle that ensures that Obscuro participants have the right incentives to follow the protocol.

Rollup Management

This module accepts rollups submitted by L2 nodes and includes them in the rollup-chain structure. It works together with the bridge in processing withdrawal requests from users.

Obscuro Bridge

This contract is very important for the solution's security since it will protect all liquidity deposited by Ethereum end-users, and reflected in the confidential Obscuro ledger.

L2 Network

The goal of the L2 design is to create a fair, permissionless, and decentralised network of nodes with valid TEEs who cannot see the transactions they are processing while collaborating to manage a ledger stored as rollups in the L1. The ledger should preserve its integrity even in the face of catastrophic TEE hacks.

All Obscuro nodes have to go through the attestation process with the Network Management contract before receiving the shared secret and participating.

Note: the shared secret is covered in the cryptography section.

{% include_relative aggregators-verifiers.md %} {% include_relative rollup-data-structure.md %}

Consensus - Proof of Block Inclusion

Obscuro uses a novel decentralised round-based consensus protocol based on a fair lottery and synchronisation with the L1, designed explicitly for L2 rollups, called *Proof Of Block Inclusion* (POBI). It solves, among others, the fair leader election problem, which is a fundamental issue that all decentralised rollup solutions have to address. POBI is inspired by Proof Of Elapsed Time.

High-Level Description

The high level goals of the POBI protocol are:

1. Each round, distribute the sequencer function fairly among all the active registered Aggregators.
2. To synchronise the L2 round duration to L1 rounds. Because the L1 is the source of truth, the finality of the L2 transactions is dependent on the finality of the L1 rollup transaction that includes them, which means there is no advantage in publishing multiple rollups in a single L1 block. It is impossible to decrease the finality time below that of the L1, and, on the other hand, publishing L2 rollups less

frequently means that L2 finality is unnecessarily long. The optimum frequency is to publish one rollup per L1 block.

To achieve fairness, the POBI protocol states that the TEE can generate one random nonce each round, and the winner of a round is the Aggregator whose TEE generates the lowest random number from the group. The TEEs generate these numbers independently and then gossip them. The Aggregators who do not win the round, similar to L1 miners, respect this decision because it is rational to do based on the incentive mechanism. If they choose to not respect the protocol, they are free to submit a losing rollup to the L1, which is ignored by all compliant Aggregators, meaning such an Aggregator has to pay L1 gas and not get any useful reward.

The second goal is achieved by linking the random nonce generation, which terminates a round, to the Merkle proof of inclusion of the parent rollup (which exists as a transaction in the L1 transaction Patricia trie) in an L1 block. This property is what gives the name of the protocol. This means that an Aggregator can obtain a signed rollup from the TEE only if it can show the rollup is based on a published rollup in a prior L1 block. Furthermore, this feature links the creation of L2 rollup to an L1 block, thus synchronising their cadence.

A party wishing to increase its chances of winning rounds must register multiple Aggregators and pay the stake for each. The value of the stake needs to be calculated in such a way as to achieve a right decentralisation and practicality balance.

It is straightforward for all the other Aggregators to verify which rollup is the winner by comparing the nonces and checking that the rollup signature is from an approved Aggregator.

Note that the L1 Management Contract is not checking the nonces of the submitted rollups, but it checks that the block inclusion proof is valid. The L1 contract rejects rollups generated using a proof of inclusion that is not an ancestor of the current block.

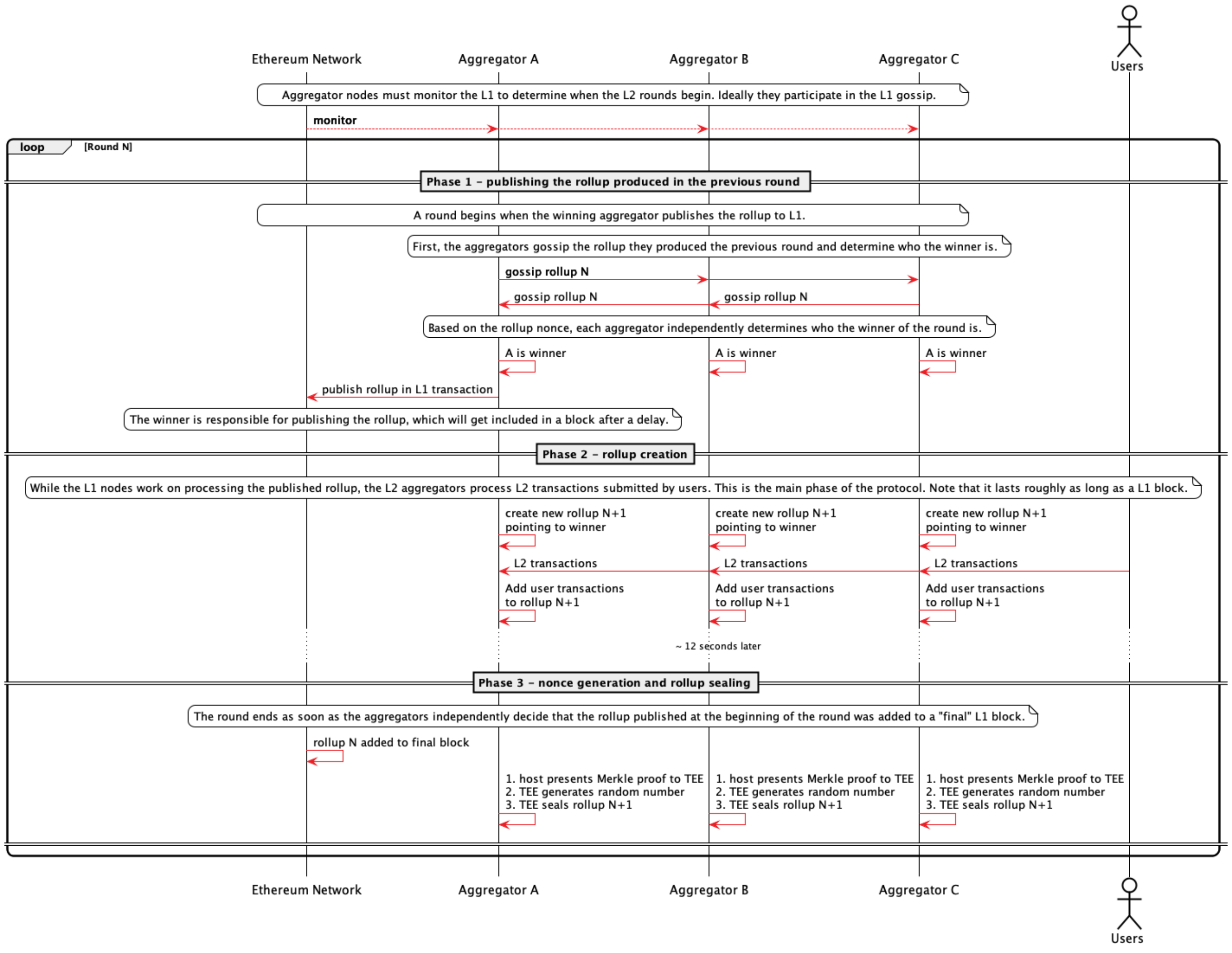
A further issue to solve is to ensure that the host cannot repeatedly submit the proof to the TEE to try to get a lower nonce, explained here.

Typical Scenario

1. A new round starts from the point of view of an Aggregator when it decides that someone has gossiped a winning rollup. At that point, it creates a new empty rollup structure, points it to the previous one, and starts adding transactions to it (which are being received from users or by gossip).
2. In the meantime, it closely monitors the L1 by being directly connected to an L1 node.
3. As soon as the previous rollup was added to a mined L1 block, the Aggregator takes that Merkle proof, feeds it to the TEE, who replies with a signed rollup containing a random nonce generated inside the enclave.
4. All the other Aggregators do roughly the same thing at the same time.
5. At this point (which happens immediately after successfully publishing the previous rollup in the L1), every Aggregator has a signed rollup with a random nonce which they gossip between them. The party with the lowest nonce wins. All the Aggregators know this, and, after a short waiting period, a new round starts.
6. The winning Aggregator has to create an Ethereum transaction that publishes this rollup to L1.

Note that by introducing the requirement for proof of inclusion in the L1, the cadence of publishing the rollups to the block times is synchronised. Also, note that the hash of the L1 block used to prove to the TEE that the previous rollup was published is added to the current rollup such that the Management Contract and the other Aggregators know whether this rollup was generated correctly.

The following diagram depicts this sequence:



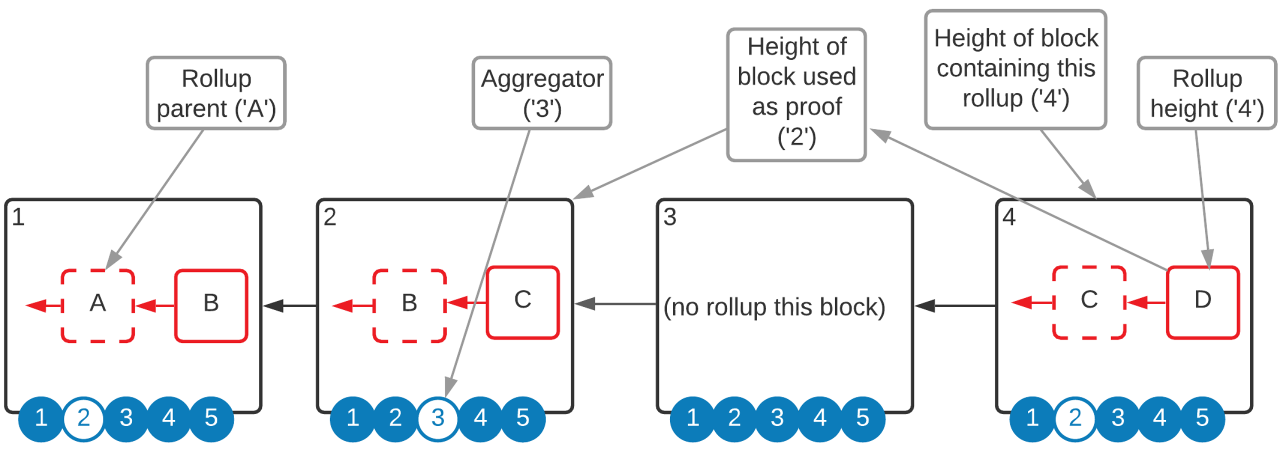
Ethereum Network
Aggregator A
Aggregator B
Aggregator C
Users
Aggregator nodes must monitor the L1 to determine when the L2 rounds begin. Ideally they participate in the L1 gossip.
monitor
loop
[Round N]
Phase 1 – publishing the rollup produced in the previous round
A round begins when the winning aggregator publishes the rollup to L1.
First, the aggregators gossip the rollup they produced the previous round and determine who the winner is.
gossip rollup N
gossip rollup N
gossip rollup N
Based on the rollup nonce, each aggregator independently determines who the winner of the round is.
A is winner
A is winner
A is winner
publish rollup in L1 transaction
The winner is responsible for publishing the rollup, which will get included in a block after a delay.
Phase 2 – rollup creation
While the L1 nodes work on processing the published rollup, the L2 aggregators process L2 transactions submitted by users. This is the main phase of the protocol. Note that it lasts roughly as long as a L1 block.
create new rollup N+1 pointing to winner
create new rollup N+1 pointing to winner
create new rollup N+1 pointing to winner
L2 transactions
L2 transactions
L2 transactions
Add user transactions to rollup N+1
Add user transactions to rollup N+1
Add user transactions to rollup N+1
~ 12 seconds later
Phase 3 – nonce generation and rollup sealing
The round ends as soon as the aggregators independently decide that the rollup published at the beginning of the round was added to a "final" L1 block.
rollup N added to final block
1. host presents Merkle proof to TEE 2. TEE generates random number 3. TEE seals rollup N+1
1. host presents Merkle proof to TEE 2. TEE generates random number 3. TEE seals rollup N+1
1. host presents Merkle proof to TEE 2. TEE generates random number 3. TEE seals rollup N+1
Ethereum Network
Aggregator A
Aggregator B
Aggregator C
Users

Notation

There are six elements that define a rollup :

1. The rollup parent.
2. The rollup height (Nth generation).
3. The Aggregator who generated it.
4. The height of the L1 block used as proof (L1_Proof_Height).
5. The height of the L1 block that includes this rollup (L1_Block_Height).
6. The nonce.

The following diagram depicts these elements:



The notation is the following: *R*$Rollup_Height[$Aggregator, L1_Proof_Height, L1_Block_Height, $Nonce]_.

Note that the value of *L1_Proof_Height* can only be lower than *L1_Block_Height*.

Example: *R_15[Alice, 100, 102, 20]* means the rollup height is 15, the aggregator is *Alice*, the height of the L1 block used as proof is 100, the height of the L1 block that included the rollup is 102, and the nonce equals 20.

The Canonical Chain

The POBI protocol allows any Aggregator to publish rollups to the Management Contract, so short-lived forks are a normal part of the protocol. The forks cannot be long-living during normal functioning because the ObscuroVM running inside the TEE of every node deterministically selects one of the forks as the canonical chain and only appends a rollup on top of that.

Because the logic is identical and attested on all nodes and the TEEs receive all the relevant content of the L1 blocks (which means they process the same input data), there cannot be any competing forks more than one rollup deep unless there is a hack.

The rules for the canonical chain are the following:

1. The genesis rollup is part of the canonical chain and will be included in an L1 block by the first Aggregator.
2. An L1 block containing a single rollup whose parent is the head rollup of the canonical chain included in a previous L1 block is on the canonical chain if no other rollup with the same parent was included in an earlier block. Any other sibling rollup included in a later block is not on the canonical chain. This is the *Primogeniture* rule, where a rollup is born when included in an L1 block.
3. If an L1 block contains multiple sibling rollups created in the same round using the same L1 proof, the one with the lower nonce is on the canonical chain.
4. If an L1 block contains multiple sibling rollups created using different L1 proofs, the one created using the more recent proof is on the canonical chain.

Using the notation, for the same *Rollup_Height*, the rollup on the canonical chain is the one with:

1. The lowest L1_Block_Generation.
2. In case there are multiple matches, use the highest L1_Proof_Generation.
3. In case there are multiple matches, use the lowest nonce.

Given that the nonce is a random number with sufficient entropy, we assume there cannot be a collision at this point during normal functioning. In the situation where it happens, the rollup on the canonical chain will be the one with the lowest hash.

Preventing Repeated Random Nonce Generation

In phase 3 of the protocol, the TEE of each Aggregator generates a random nonce which determines the winner of the protocol. This introduces the possibility of gaming the system by restarting the TEE and generating multiple numbers.

The solution proposed by Obscuro is to introduce a timer in the constructor upon every startup of the enclave. A conventional timer, based on the clock of the computer, is not very effective since the host can game it. Instead, the enclave must calculate serially (on a single thread) a large enough number of SHA256 hashes, which it would not be able to do faster than an average block time even on powerful hardware.

This solution is effective since the code is attested and does not rely on any input from the host.

A node operator wanting to cheat would restart the enclave and quickly feed it the proof of inclusion, only for the enclave to process it after 15 seconds, which means the operator has already missed the time window for that rollup.

This built-in startup delay is also useful in preventing other real-time side-channel attacks, which could be used for MEV.

Aggregator Incentives

All successful decentralised solutions need a robust incentive mechanism to keep the protocol functioning effectively.

Compared to a typical L1 protocol, there is an additional complexity to consider. In an L1 like Bitcoin or Ethereum, once a node gossips a valid block, all the other nodes are incentivised to use it as a parent because they know everyone does that too. In an L2 decentralised protocol like POBI, there is an additional step: the publication of the rollup to L1, which can fail for multiple reasons. Furthermore, the incentive design must also consider the problem of front-running the actual rollup. For a rollup to be final, it has to be added to an L1 block, which is where an L1 miner or staker can attempt to claim the reward that rightfully belongs to a different L2 node.

Note that rollup finality will be covered extensively in the Obscuro - Ethereum interaction section.

The high-level goal is to keep the system functioning as smoothly as possible and resist random failures or malicious behaviour while not penalising Obscuro nodes for not being available. We believe that penalties for availability increase the barrier of entry, and thus make the system centralised over the long term.

Obscuro introduces the concept of *claiming rewards* independently of the actual canonical rollup chain. The great advantage is increased flexibility in aligning incentives at the cost of increased complexity. Rewards can be awarded in full, split between Aggregators or just enough to cover the cost of gas.

To achieve this, the protocol has to maintain a pool of tokens. Users will pay fees into this pool, while nodes will be paid from it. During bootstrapping, the protocol will have the ability to add newly minted tokens to the pool. Once the network picks up, the protocol will be able to burn excess tokens.

Note, that an important assumption is that the reward from publishing a rollup will never exceed twice the gas cost.

These are the Aggregator rewarding rules:

1. The first Aggregator to successfully published a rollup without competition in an L1 block will get the full reward. This is the most efficient case that is encouraged. *Note: Competition means another rollup with the same parent.*

2. If multiple rollups generated with the same L1 proof and different nonces are published in the same block (the target block), the one with the lowest nonce is on the canonical chain, but the reward is split between them in a proportion of 75/25 (this ratio is indicative only). The reason for this rule is that it incentivises Aggregators to gossip their winning rollup such that no one else publishes at the same time.

    - There is no incentive for the losing Aggregator to publish as 25% of the reward will not cover the cost of gas, so they will make a loss.
    - There is an incentive for the winning Aggregator to gossip the rollup to everyone else to avoid having this unwanted competition.
    - In case of a genuine failure in gossip (i.e. beyond designed latency), the losing Aggregator receives something. This is to reduce the risk of Aggregators waiting more than necessary to receive messages from all the other Aggregators.

3. If multiple sibling rollups generated using different L1 blocks as proof are included in the same block, the one created with the most recent proof receives the full reward.

    The original winning rollup that did not get published immediately does not receive any reward since more recent competition exists. This rule is designed to encourage publishing with enough gas, such that there is no risk of competition in a further block. The rule also encourages Aggregators to not wait for rollups published with insufficient gas or not at all.

    This mechanism ensures rounds reset when new L1 blocks are available and the reward is up for grabs. An actor controlling multiple Aggregators with malicious irrational behaviour can only slow down the ledger because the rational actors will publish the rounds they win.

4. If two consecutive L1 blocks include each a rollup with the same height created from the same L1 proof, but the rollup from the second block has a lower nonce, split the reward evenly between the two Aggregators.

    Note that the rollup with the higher nonce is on the canonical chain.

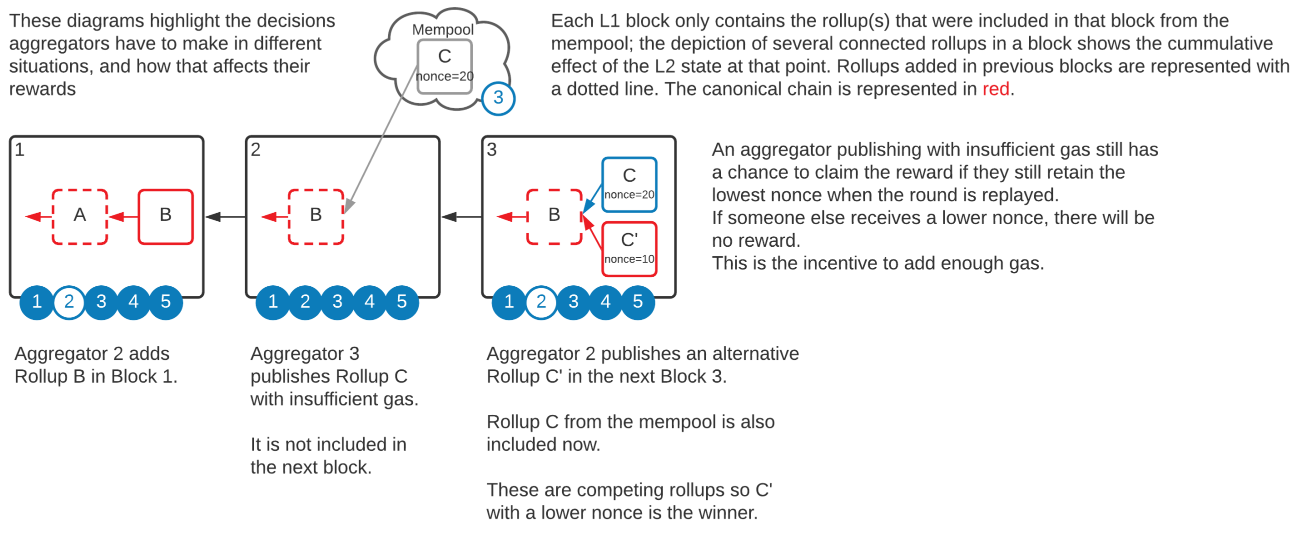
    The reason for this rule is that this scenario is possibly the result of rollup front-running, which is thus discouraged as the frontrunner is consuming precious Ethereum gas and the reward will always be less than the cost.

    The even splitting of the reward also encourages the Aggregator that wins a nonce generation round to publish as soon as possible, because publishing a block later will at best result in a small loss.

5. If two sibling rollups created from the same L1 proof are published more than one block apart, where the first published rollup has a higher nonce, then pay the reward in full to the first published rollup. The reason for this rule is that the winner most likely added too little gas, and someone else spotted

the opportunity and contributed to earlier finality, which is rewarded. It adds an incentive to monitor gas prices and pay enough to ensure their rollup is published.

The following diagrams depict some of the rewarding rules:



These diagrams highlight the decisions aggregators have to make in different situations, and how that affects their rewards

Each L1 block only contains the rollup(s) that were included in that block from the mempool; the depiction of several connected rollups in a block shows the cummulative effect of the L2 state at that point. Rollups added in previous blocks are represented with a dotted line. The canonical chain is represented in red.

An aggregator publishing with insufficient gas still has a chance to claim the reward if they still retain the lowest nonce when the round is replayed.
If someone else receives a lower nonce, there will be no reward.
This is the incentive to add enough gas.

Aggregator 2 adds Rollup B in Block 1.

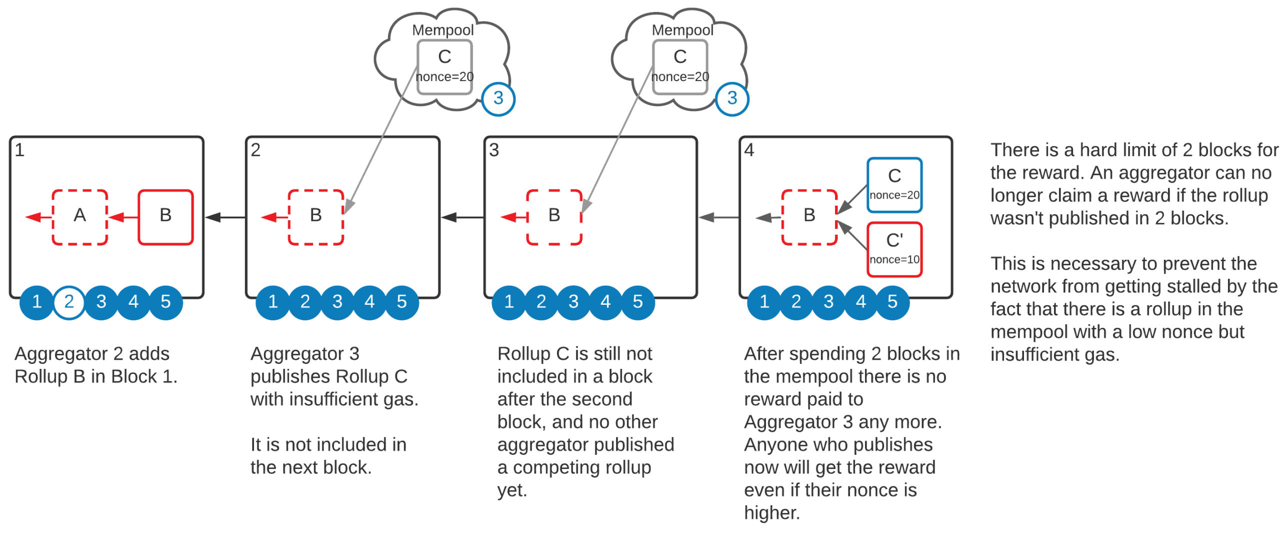
Aggregator 3 publishes Rollup C with insufficient gas.

It is not included in the next block.

Aggregator 2 publishes an alternative Rollup C' in the next Block 3.

Rollup C from the mempool is also included now.

These are competing rollups so C' with a lower nonce is the winner.



Aggregator 2 adds Rollup B in Block 1.

Aggregator 3 publishes Rollup C with insufficient gas.

It is not included in the next block.

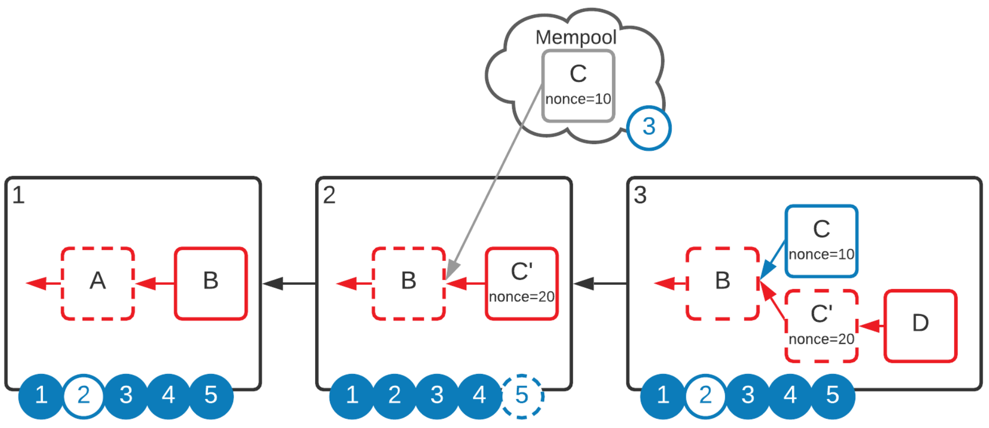
Rollup C is still not included in a block after the second block, and no other aggregator published a competing rollup yet.

After spending 2 blocks in the mempool there is no reward paid to Aggregator 3 any more. Anyone who publishes now will get the reward even if their nonce is higher.

There is a hard limit of 2 blocks for the reward. An aggregator can no longer claim a reward if the rollup wasn't published in 2 blocks.

This is necessary to prevent the network from getting stalled by the fact that there is a rollup in the mempool with a low nonce but insufficient gas.

The following diagram depicts rules in the case of front-running:



Aggregator 2 adds Rollup B in Block 1.

Aggregator 5 adds Rollup C' in Block 2, and this transaction is added by a colluding L1 miner/staker, front-running the rightful Rollup C transaction submitted by Aggregator 3. Rollup C's transaction is left in the L1 mempool.

Since there is no competitor for rollup B in the next block, it means Aggregator 2 can claim the full reward.

Block 3 includes Rollup D, added by Aggregator 2 and appended to the only Rollup C' published in Block 2, as well as rightful Rollup C which is added late from the mempool.

Since Rollup C appears as a sibling with a lower nonce, Aggregator 5 (C') does not receive any reward, while Aggregator 3 (C) can claim back the cost of gas.

Note that rollup C's transactions do not affect the state as they are not on the canonical chain.

Note that each L1 block only contains the rollup(s) that were included in that block from the mempool; the depiction of several connected rollups in a block shows the cummulative effect of the L2 state at that point.

Rollups added in previous blocks are represented with a dotted line.

The canonical chain is represented in red.

This is python-like pseudocode to calculate the rewards that can be claimed by an *Aggregator* for a *Rollup_Height*. Note that it is not comprehensive, and there may be many competing aggregators.

```
# The rollup height for which we calculate the rewards
height = N

# 'heightN_L1_Blocks' is a list of all L1 blocks starting with the
_L1_Block_Height_ of the head
# of the canonical chain of the previous generation, until the block where you
encounter the
# first valid rollup of _Rollup_Height_ plus one extra L1 block.
heightN_L1_Blocks = calculateBlocks()

# List of rollups of height N found in the last block
rollups_in_last_block = heightN_L1_Blocks[-1].rollups.filter(r.height == height)

# List of rollups of height N found in the target block
rollups_in_target_block = heightN_L1_Blocks[-2].rollups.filter(r.height == height)

if rollups_in_target_block.size == 1 and rollups_in_last_block.size == 0:
    # There is no competition for the target rollup
    fullRewardTo(rollups_in_target_block[0].aggregator)

elif rollups_in_target_block.size == 1 and rollups_in_last_block.size == 1:

    # There is competition for the target rollup in the next rollup
    # Which means there is suspicion of frontrunning
    target_rollup = rollups_in_target_block[0]
    competition_rollup = rollups_in_last_block[0]

    if competition_rollup.L1_Proof_Height == target_rollup.L1_Proof_Height and
competition_rollup.nonce < target_rollup.nonce:
        # This is possibly front-running or failure to gossip
        # All parties involved in this will make a small loss
        partialRewardTo(target_rollup.aggregator, '50%')
        partialRewardTo(competition_rollup.aggregator, '50%')
    else:
        # The target has the lower nonce or is generated with a different proof
        fullRewardTo(target_rollup.aggregator)

elif rollups_in_target_block.size == 2:
    # Two competing rollups in the target block
    # This is not a front-running situation, so eventual rollups published in the
next block do not matter
    rollup1 = rollups_in_target_block[0]
    rollup2 = rollups_in_target_block[1]

    if rollup1.L1_Proof_Height == rollup2.L1_Proof_Height:

        # According to rule #2 the competing rollups will split the reward
        if rollup1.nonce < rollup2.nonce:
            partialRewardTo(rollup1.aggregator, '75%')
            partialRewardTo(rollup2.aggregator, '25%')
        else:
            partialRewardTo(rollup1.aggregator, '25%')
            partialRewardTo(rollup2.aggregator, '75%')

    elif rollup1.L1_Proof_Height > rollup2.L1_Proof_Height:

        # According to rule #3 the rollup generated with the more recent proof
gets the reward
        fullRewardTo(rollup1.aggregator)

    else:
        # According to rule #3 the rollup generated with the more recent proof
gets the reward
        fullRewardTo(rollup2.aggregator)

else:
    pass
```

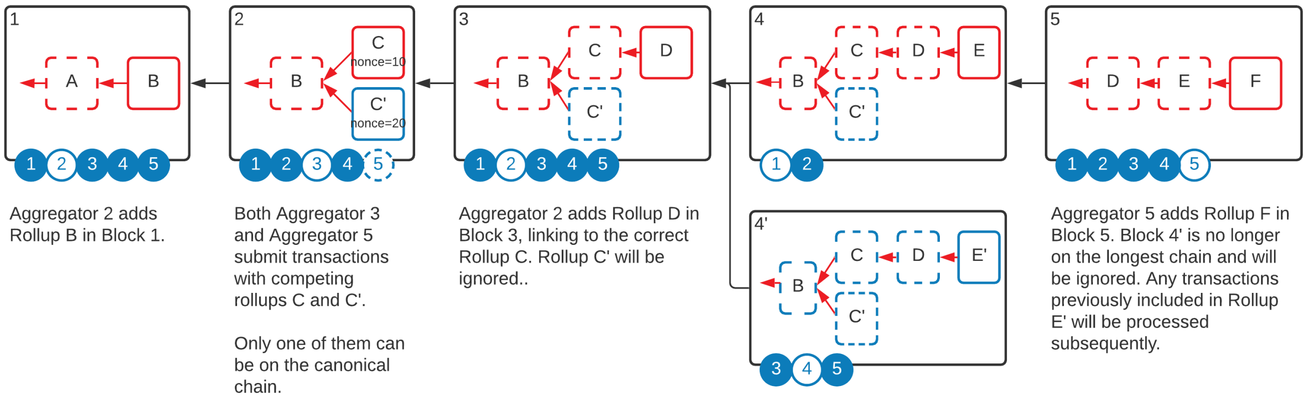
Note that these rules are subject to adjustment based on production observations.

Rollup Evolution

This diagram shows a series of edge cases and what happens.

Note that each L1 block only contains the rollup(s) that were included in that block from the mempool; the depiction of several connected rollups in a block shows the cummulative effect of the L2 state at that point. Rollups added in previous blocks are represented with a dotted line. The canonical chain is represented in red.

Aggregators 1,2 are attached to L1 nodes holding a forked Block 4, and Aggregator 1 adds Rollup E in Block 4.



Aggregator 2 adds Rollup B in Block 1.

Both Aggregator 3 and Aggregator 5 submit transactions with competing rollups C and C'.

Only one of them can be on the canonical chain.

Note that the incentive mechanism does not encourage this scenario.

Aggregator 2 adds Rollup D in Block 3, linking to the correct Rollup C. Rollup C' will be ignored..

Aggregators 3,4,5 are attached to L1 nodes holding a forked Block 4', and Aggregator 4 adds Rollup E' in Block 3, linking to the correct Rollup D.

Aggregator 5 adds Rollup F in Block 5. Block 4' is no longer on the longest chain and will be ignored. Any transactions previously included in Rollup E' will be processed subsequently.

Detailed Technical Design

This section describes key Obscuro component designs.

{% include_relative cryptography.md %} {% include_relative account-based-state.md %} {% include_relative l1-contracts.md %}

Obscuro and Ethereum Interaction

Obscuro is a confidential extension to Ethereum, and thus assets have to move freely between the two networks.

All sidechains and L2 solutions have developed solutions to the mismatches between the different models of the two networks, and typically there is a bridge contract that safeguards assets. The difference between sidechains and L2 solutions is that mismatches are more significant for sidechains because they have their own finality and security mechanisms, and thus the bridge logic is either very complex or centralised.

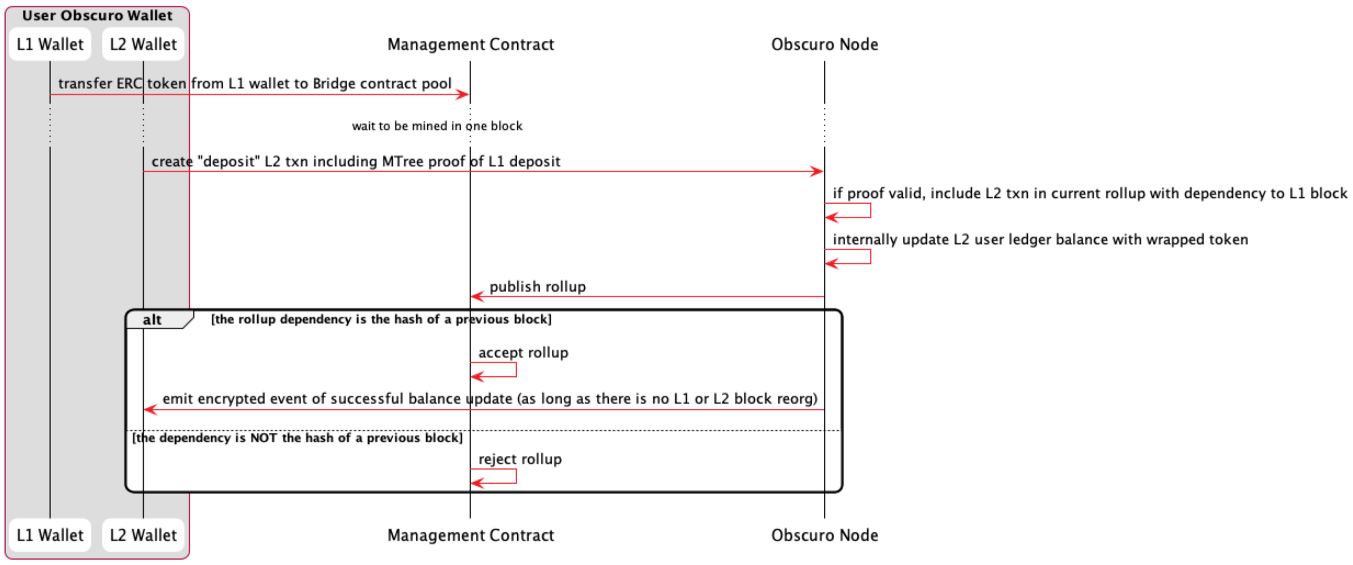
Deposits

The user deposits supported ERC tokens into the well-known address of the Bridge contract, and once the transaction is successfully added to a block, the Obscuro-enabled wallet automatically creates an L2 transaction, including proof of the L1 transaction. The exact amount is credited with wrapped tokens on the user's account on Obscuro.

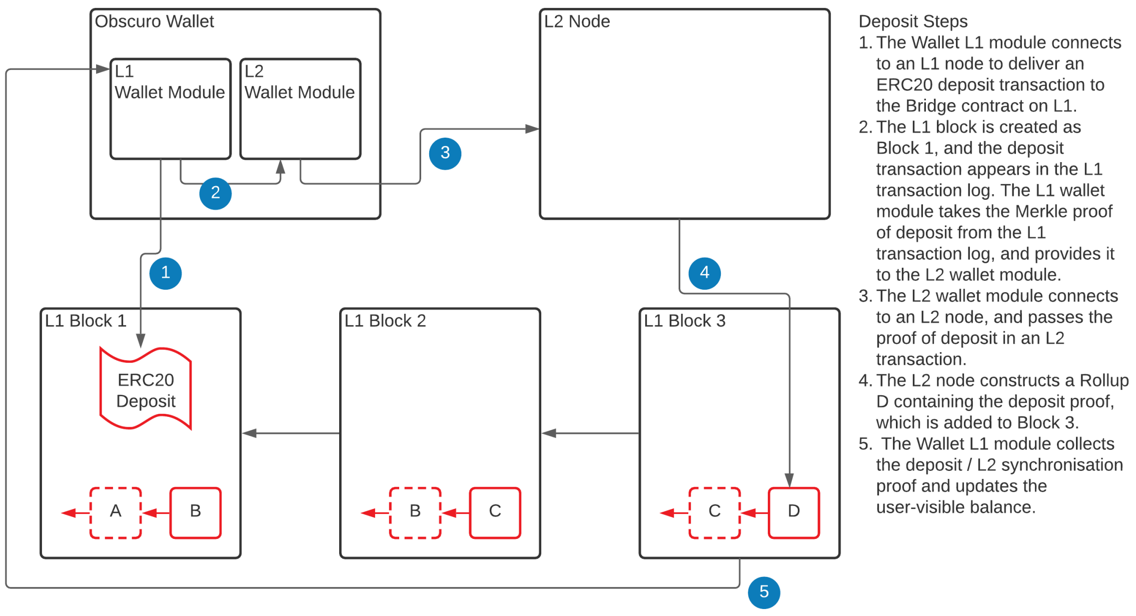
The fact that the finality of L1 transactions is probabilistic makes crediting the L2 account not straightforward. Most solutions solve this problem by waiting for a confirmation period before crediting the account. Obscuro takes a different approach and introduces a dependency mechanism between the L2 rollup and the L1 blocks.

The rule is that the L2 rollup that includes the transaction that credits the Obscuro account has a hard dependency on an L1 block, and the Bridge contract enforces that it is one of the ancestors of the current block. If the L1 deposit transaction is no longer in the canonical L1 chain, it automatically invalidates the rollup that contains the L2 deposit transaction, and the L1 deposit will only be recognised as the basis for an L2 rollup credit transaction when it has been included in the canonical L1 chain.

The interaction is shown in the following diagram:


User Obscuro Wallet
L1 Wallet
L2 Wallet
Management Contract
Obscuro Node
transfer ERC token from L1 wallet to Bridge contract pool
wait to be mined in one block
create "deposit" L2 txn including MTree proof of L1 deposit
if proof valid, include L2 txn in current rollup with dependency to L1 block
internally update L2 user ledger balance with wrapped token
publish rollup
alt
[the rollup dependency is the hash of a previous block]
accept rollup
emit encrypted event of successful balance update (as long as there is no L1 or L2 block reorg)
[the dependency is NOT the hash of a previous block]
reject rollup
L1 Wallet
L2 Wallet
Management Contract
Obscuro Node

See also the Data model section and the following dependency diagram.



Deposit Steps
1. The Wallet L1 module connects to an L1 node to deliver an ERC20 deposit transaction to the Bridge contract on L1.
2. The L1 block is created as Block 1, and the deposit transaction appears in the L1 transaction log. The L1 wallet module takes the Merkle proof of deposit from the L1 transaction log, and provides it to the L2 wallet module.
3. The L2 wallet module connects to an L2 node, and passes the proof of deposit in an L2 transaction.
4. The L2 node constructs a Rollup D containing the deposit proof, which is added to Block 3.
5. The Wallet L1 module collects the deposit / L2 synchronisation proof and updates the user-visible balance.

Note: The deposit L2 transaction cannot be fully encrypted because the Aggregator has to decide whether to include it in the current rollup based on the chances of the L1 block it depends on being final.

Withdrawals

The high-level requirement for the withdrawal function is simple: allow Obscuro users to move assets back into the Ethereum network. The problem is that this is where the most significant threat against such a solution lies because there might be a large amount of locked value.

The challenge is to implement this functionality in a decentralised way by defining a protocol and economic incentives.

Due to the sensitivity of this function, many sidechains and L2 solutions rely on multi-signature technology to control the release of funds. Optimistic Rollups rely on a challenge mechanism during a long waiting period before releasing funds, powered by economic incentives.

Obscuro uses TEE technology, but it cannot leverage it for this aspect because of our threat model. The Bridge Contract could release funds based on a signature from an attested TEE if it were invulnerable, but since that is not the case, the solution is to use economic incentives on top of the POBI protocol.

Rollup Finality

The general rule is that withdrawals can be processed only when a rollup is *final*. This means this is the protocol for the finality of the Obscuro chain relative to the Ethereum chain.

Rule 1 - The standard delay period

In the usual case, a rollup from the canonical chain (see POBI protocol) is final if a standard number of blocks corresponding to a period of 1 day has passed from the Ethereum block where it was published.

- Note 1: The period is measured in Ethereum blocks because the delay is stable on average between blocks.
- Note 2: The reason for this period is to give honest nodes the chance to "challenge" the rollup if it is malicious.
- Note 3: The period is inverse to the number of L2 nodes. It should be long enough to give honest participants the chance to react and publish in the face of aggressive censorship attempts against them, but short enough not to degrade the user experience. We estimate that once the network reaches a healthy number of nodes, we can reduce it to 50-100 blocks (~ 10 minutes).

Rule 2 - The competing forks

Assuming the period chosen at rule #1 is enough, the only possible write attack performed by an actor that could hack the TEE manifests as multiple parallel forks at least two rollups deep. This is because all valid TEEs run the same attested code that chooses the same canonical chain from the rollups published in the L1 block presented as proof. If the Management Contract notices multiple forks, the rule is that finality is suspended on all forks, thus, withdrawals are suspended. Likewise, if one of the forks becomes inactive, the rule is that all rollups on the alive fork become final once a standard period of 1 day has passed from the last L1 block that contained a rollup published on the inactive branch.

- Note1: This rule degrades a *write-attack* into a Denial of Service attack on the withdrawal function.
- Note2: Assuming there are honest participants, the actual canonical ledger keeps growing, including user transactions.
- Note3: The attacker has to spend Ethereum gas to keep the malicious fork alive.

Rule 3 - addressing the DoS on finality

Since rule #2 transforms any attack into a DoS attack, the protocol has some mechanisms to keep user experience satisfactory even in the extreme case of a TEE hack.

1. The ultimate backstop is the "Attestation Constraints" rules. Forks in the canonical chain are clearly a breach of protocol, caused either by a TEE hack or a protocol hack. This is ultimately resolved with software or, at worst, hardware updates. Once the management Contract forces an upgrade, the attacker will no longer be able to create malicious rollups, and thus the fork becomes inactive, and finality resumes on the valid fork.

2. For any users with an L2 node, it is obvious which is the canonical chain, as it is the one that does not fail. Market makers operating on both L1 and L2 can step in and absorb the withdrawal requests of users at a slight discount without taking any actual risk.

The above rules will, in practice, prevent this type of attack, and if it happens, offer a practical solution for users. In addition, the protocol has yet another backstop to address the extreme case of a very persistent attacker.

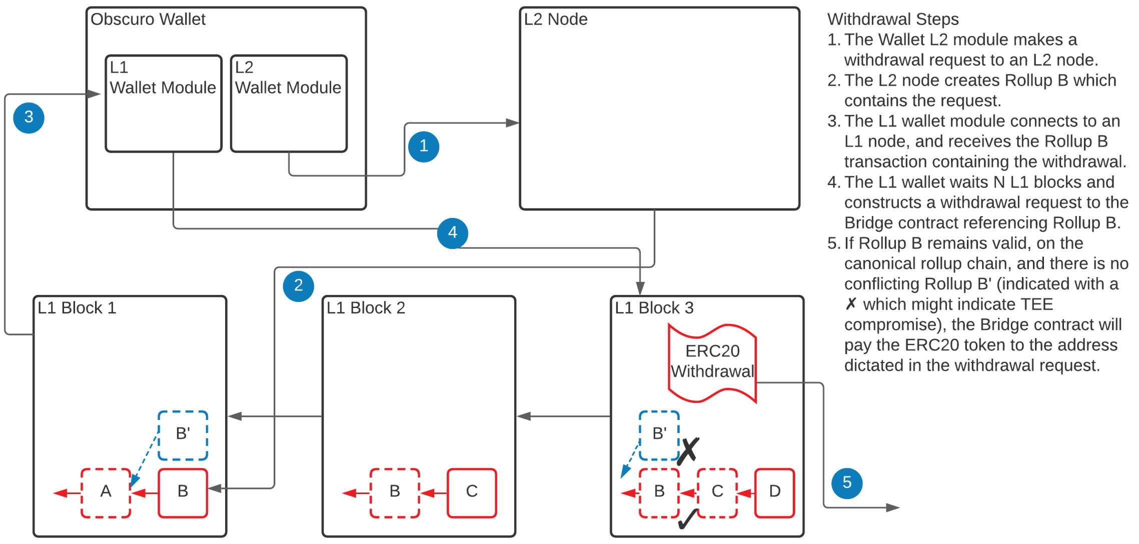
The network governance model allows any user to trigger the *forced finality procedure* by staking or voting on one of the competing rollup chains. The minimum stake is a percentage of the amounts being withdrawn on that branch, set through governance. Backers of the other chain are obliged to stake a similar or higher value to compete. The decision process is run as an auction, where the party that loses also loses their bids. When concluded, all rollups on that chain are considered final, and withdrawals are executed.

Withdrawals protocol

Each TEE signed rollup contains a plaintext list of withdrawal requests. See: Data Model.

The Bridge contract keeps track of these requests and executes them at different times, based on the finality status of that rollup.

The withdrawal process is indicated in the following diagram:



Withdrawal Steps

1. The Wallet L2 module makes a withdrawal request to an L2 node.
2. The L2 node creates Rollup B which contains the request.
3. The L1 wallet module connects to an L1 node, and receives the Rollup B transaction containing the withdrawal.
4. The L1 wallet waits N L1 blocks and constructs a withdrawal request to the Bridge contract referencing Rollup B.
5. If Rollup B remains valid, on the canonical rollup chain, and there is no conflicting Rollup B' (indicated with a ✗ which might indicate TEE compromise), the Bridge contract will pay the ERC20 token to the address dictated in the withdrawal request.

Obscuro public events

Ethererum application developers can use a confidential L2 like Obscuro for some jobs that are not possible otherwise.

For example, an L1 smart contract organises a fair lottery that needs a reliable random number generator that the miners cannot game.

Another example is publishing the result of a poker game played inside Obscuro, which the L1 contract can use to make a payment or update the tournament results.

The challenge for achieving this functionality is that the data originating in L2 has to be final. Luckily Obscuro has this mechanism already in place for processing withdrawals. Applications running inside Obscuro can emit special types of events called *Public Events*, which the OVM will add in plaintext into a dedicated data structure in the rollup. The _Rollup Contract _ first processes the rollup, and then once they reach finality, it exposes these events to external contracts.

Note: The fair lottery can be implemented in two steps to avoid any possible influence. The implementation can use the submarine technique and first publish the hash of that number in an event, and a few blocks later publish the actual number in a different event.

Threat Model

Obscuro is different from traditional L1 or L2 solutions primarily because data is stored and processed privately in trusted execution environments, which brings a new set of threats. Compared to other L2 solutions, the decentralised nature of the POBI protocol also brings some new threats.

The main threat to any ledger technology is data corruption, also known as a safety failure. It could take the form of stealing assets, double spending assets, or, more generally, adding illegal entries. Leading blockchains solve this problem by implementing a *Byzantine Fault Tolerant* (BFT) consensus between independent parties and creating incentives to ensure that anyone who breaks the rules will not be rewarded or will even be penalised. The most extreme attack is a 51% *Sybil* attack, where the attacker somehow gains the majority of the decision power (computing power for *proof of work* or stake for *proof of stake*) and can rewrite the history. This attack manifests as replacing an existing valid transaction with a valid competing transaction. While the ledger remains *logically* valid, this is equivalent to stealing for the beneficiary of the first transaction. If the attacker tried to *physically* corrupt the ledger, everyone would ignore the invalid block. The best defence against this attack is to ensure that multiple independent powerful actors have no incentive to collude.

The general principle of the Obscuro protocol is that it reverts to the behaviour of a typical non-confidential blockchain in case of hacks on the TEE technology. In other words, the safety of the ledger does not depend on the hardness of the TEEs; instead, what happens is that attackers can read transactions and data. Also, Obscuro does not delegate safety to a single actor by planning to support TEEs from multiple hardware manufacturers. In case of severe attacks, there are multiple mitigation mechanisms in place, the ultimate being that the L2 ledger is frozen, and everyone has the chance to withdraw using balance proofs.

Obscuro achieves data availability in the same way as all the other rollup solutions; the L1 is the source of data truth for the L2 network. Any L2 node with a valid TEE in possession of the shared secret can download the rollup chain from the L1, calculate the entire state inside its encrypted memory, and at the same time validate all transactions.

The following sections analyse the different threats against the Obscuro protocol.

Threats to the TEE Technology

The Obscuro design considers that the TEE technology and the program inside are not easily hackable, so the protocol is not optimised to handle them. Attacks on TEEs have occurred in laboratories, so a secondary but essential concern is to prevent ultra-sophisticated actors with the ability to hack this technology from stealing funds or breaking the integrity of the ledger.

The threat model of Obscuro is that sophisticated attackers run an Aggregator node on a machine with a TEE they control, have access to the master seed and the entire ledger, and run any possible attack on it, including attacks on the physical CPU.

Assuming that such attacks are successful, the attacker can limit themselves to read-level access or an attempt to corrupt the ledger using a write-level attack.

Read-Level Attacks

Read-level hacks happen when the attacker can extract some information from the TEE. This threat is specific to confidential blockchains.

The only way to defend against these attacks is to carefully audit the code and keep the *Attestation Constraints* up to date. If this attacker is discreet, the information leak can continue until a software patch is published or until new hardware that removes this attack is released.

Another way to defend against it which will be considered in future versions, is to implement a scheme similar to key rotations.

The least severe read attack is a side-channel where the attacker can find information about a specific transaction. The most severe is when the attacker can extract the master secret and read all present and future transactions.

If such an attack is successful, the network is equivalent to the behaviour of a typical public blockchain where all transactions are public, and MEV is possible.

Write-Level Attacks

Write-level hacks are powerful in theory since they could enable the attacker to *write* to the ledger and thus be able to break its integrity if there were no other protections.

A write-level hack could happen if an attacker extracts the enclave key and signs hand-crafted rollups that contain invalid transactions or balances.

Note: This type of attack is viewed as the main threat to the protocol and thus handled explicitly.

The mechanism to prevent this attack is described in detail in the Withdrawals section.

The high level goal of the protections is to transform such an attack into a liveness attack on the withdrawal function.

Colluding Write-Level Attacks

An extreme variant of the *Write-level attack* is performed by a powerful group that hacked the TEE and was able to take complete control of all the Aggregator nodes.

The defence against this attack is to incentivise a reasonable number of Verifiers to watch the Obscuro ledger in real-time. These actors will detect a malicious head rollup and notice that no other valid fork is being published.

Note: One such actor monitoring the network will be the Obscuro Foundation, which has the mandate to keep the protocol functioning correctly. The protocol also rewards other independent parties to take on this job by assigning random rewards to Verifiers who can prove they are active.

Any L2 node can become an Aggregator quickly by benefiting from the censorship resistance of Ethereum. To counter the attack, they will have to pay the stake and publish a correct rollup.

Attacks Against The Fair Lottery That Designates The Winner Of The Round

The POBI protocol assigns a leader each round by using random numbers generated inside the TEE. An attacker that can hack the technology could generate a well-chosen number and thus win each round. This is not an attack against the safety of the ledger and is not of great concern.

If some Aggregator wins statistically many more rounds than they should, it will highlight the problem to the community.

A more dangerous variation of the attack is when the attacker can also read transactions and thus front-run and extract value.

Note: This area is under research.

A variation of this attack is when the attacker cannot directly hack the TEE, but it is restarting the TEE in the hope of generating a lower nonce and thus improving their chances. This threat is mitigated by a delay introduced at the startup of the OVM, which will cause the attacker to miss out on that rollup cycle.

Other Threats To The Protocol

This section analyses threats not directly linked to the TEE, although a hack against the TEEs might amplify them.

Invalid Rollup Attacks

The *Rollup Contract* only accepts signed rollups from Aggregators that can prove their TEE attestation, and unless the TEE itself is corrupted, it is impossible to publish invalid rollups. This means that such an attack will become a liveness attack when forks are detected in the rollup chain.

Empty Rollup Attacks

An Aggregator winning a round can freely publish empty rollups, but that would not harm the system if there were multiple independent Aggregators. It will just slow down the network. Obscuro disincentivises this attack since the reward for the publisher is linked to the fees collected from the included transactions.

Sybil Attacks

This section analyses the threats that a powerful adversary who can create many Aggregators can pose on the protocol.

The reasoning around this attack is quite different from typical public blockchains.

There are two ways to run this attack against Obscuro depending on the capabilities of the attacker:

1. The attacker sets up N CPUs with TEE and pays the stake for each of them, where (N > Total_Number_Of_Aggregators / 2).
2. The attacker hacks the TEE and can impersonate many TEEs limited only by the stake. This attack has been analysed in the "Colluding write-level attack".

Sybil Attack Without Hacking The TEE

If the attacker cannot hack the TEE, they cannot deviate from the canonical chain or insert illegal transactions, as the attested software will not let them. Having a majority on the Obscuro network will not help with this. An attacker who wants to perform a double-spend attack on Obscuro will have to change the canonical chain already published in L1 blocks. To perform a double spend, the attackers have to perform a double-spend attack on the L1 blocks themselves that contain the rollups.

Economical Sybil Attacks

Another type of attack, which a well-resourced actor can perform, is controlling many Aggregators to make a good return from the rewards. The more Aggregators someone controls, the more chance of getting a winning nonce.

There is no risk in altering the ledger or performing double-spend attacks. There is no risk of a Denial of Service attack either, by refusing to publish winning rollups since the incentives encourage other actors to quickly fill in gaps and publish rollups.

There are no risks of driving other Aggregators out of business by denying them the chance to win rollups since they will get the reward of being active nodes.

Catastrophic Events

One of the worst scenarios is a catastrophic event that leaves all the value locked.

This could happen in theory if all registered TEEs were simultaneously physically destroyed, and thus the master seed was permanently lost.

If a single TEE is not physically destroyed, and a single Ethereum node has a copy of the L1 ledger, the network can be restarted, since all the required information is stored on the L1, including the master seed encrypted with the key of the surviving enclave and all the rollups.

The defence against this is to achieve a reasonable decentralisation.

MEV By Obscuro Aggregators

Transactions and processing are hidden from node operators. Still rollups contain some information and the node operator can query the balance of accounts they control.

To make this attack impractical, Obscuro introduces a slight delay that preserves the user experience of public blockchains.

The TEE will emit events and respond to balance requests only after it received proof that the rollup was successfully published in an L1 block. This mechanism will prevent an Aggregator from probing for information while creating a rollup.

An Aggregator wishing to attack this scheme would have to quickly create valid Ethereum blocks while executing user transactions, which is highly impractical since there is a hardcoded minimum value for the mining difficulty.

Threats To The POBI Protocol

The POBI protocol handles most failure scenarios using a set of incentive rules.

1. The winning sequencer does not publish

The winning Aggregator is incentivised to publish the rollup in order to receive the reward, which means this scenario should only occur infrequently if the Aggregator crashes or malfunctions. If it happens, it will only be detected by the other Aggregators when the next L1 block does not contain the winning rollup that was gossiped about.

In this situation, every Aggregator will:

- Discard the current rollup.
- Unseal the previous rollup.
- Add all current transactions to it.
- Then seal it using the last empty block.

- Gossip it.

In effect, this means that the previous round is replayed. The winning Aggregator of this new round has priority over the reward in case the previous winner is added in the same block.

2. The winning sequencer adds too little gas, and the rollup sits in the mempool unconfirmed

This scenario has the same effect as the previous one is handled in the same way. If the rollup is not in the next block, the round is replayed.

Publishing with insufficient gas is, in effect, punished by the protocol because it means that on top of missing the rollup reward, the Aggregator also pays the L1 gas fee.

Competing L1 Blockchain Forks

In theory, different L2 Aggregators could be connected to L1 nodes that have different views of the L1 ledger. This will be visible in the L2 network, as gossiped rollups pointing to L1 blocks from the two forks. Each Aggregator will have to make a bet and continue working on the L1 fork that it considers to be legitimate, the same behaviour as any L1 node.

This is depicted in Rollup Data Structure.

If it proves that the decision an Aggregator made was wrong, it has to roll back the state to a checkpoint and replay the winning rollups.

Trust Model

The analysis in this section is based on a framework defined by Vitalik Buterin, the creator of Ethereum.

Obscuro is slightly different from typical blockchains or L2s because it introduces another actor into the trust model, the hardware manufacturer.

These are the questions that will be answered using the terminology from the framework.

1. How many people do you need to behave as you expect? Out of how many?
2. What kinds of motivations are needed for those people to behave? Do they need to be altruistic or just profit-seeking? Do they need to be uncoordinated?
3. How badly will the system fail if the assumptions are violated?

Actors

The following groups are actors in the system.

1. The Obscuro network may contain a few thousand nodes, from which a minority core set will be *Aggregators* and the rest *Verifiers*. The governance body can control this number by setting some parameters.
2. Another important group in this is the token holders, who have governance powers.
3. The supported hardware TEE manufacturers.
4. The auditors.

Notation

1. Obscuro_N - number of Obscuro nodes ~ 1000.
2. Ethereum_N - number of Ethereum nodes.
3. TEE_Manufacturer_N - number of manufacturers. Small number, but composed of large reputable companies.
4. Token_Holders_N - number of Obscuro token holders. Many thousands.

Liveness

There are multiple aspects to consider when analysing the liveness trust model. Since Obscuro is fully decentralised at the network level, as long as one single Aggregator is alive, the network is alive and processing user transactions.

For transaction processing: 1 of Obscuro_N, where the motivation of nodes is profit-seeking.

For processing withdrawals and thus reaching finality, the analysis is more complex. Since withdrawals are processed automatically from the instructions found in the rollups, the trust model for the liveness of this feature is the model for safety.

Safety

The safety of Obscuro is based on a couple of layers, which transform a safety attack into a liveness attack.

Note that the safety of the ledger is at risk only if there are hacks in the confidential hardware technology.

Given that hardware manufacturers are generally large and reputable companies, they act as the first barrier. Their motivation is ultimately profit-seeking because vulnerabilities in the hardware they create will lead to lower sales and reputational damage.

Hardware layer: 1 of TEE_Manufacturer_N, where the motivation is profit-seeking. Note that this assumes that the hardware manufacturer introduces a bug in the TEE implementation to attack the ledger. Normally the threat is lower since a single user with a valid TEE by any manufacturer will be able to stop an attack.

If there is a successful attack against the TEE, the next defence is a single active L2 node that publishes a valid rollup. 1 of Obscuro_N, where the motivation of nodes is profit-seeking.

The next line of defence are the token holders, who will vote on L1 to update the Attestation Constraints, to fix the vulnerability. They are invested in the community because they hold the token, which means they profit if it functions correctly: Token_Holders_N/2 of Token_Holders_N, where motivation is profit-seeking

Note that the attacker is not directly profit seeking because there is no possibility to withdraw assets until the fork is resolved.

How Badly Will The System Fail If The Assumptions Are Violated?

If all supported hardware manufacturers colluded, they would be able to break the safety of the ledger.

Governance

Governance for the Obscuro protocol, the reference implementation, and the network configuration will be made explicit and visible to all. Obscuro governance thinking is derived from the experience of Bitcoin and

Ethereum.

There are several types of control exercised in a decentralised system:

1. Explicit control exercised by a group of people using direct signing or voting.
2. Implicit control implemented in an immutable protocol.
3. Implicit control implemented in a protocol that itself is represented by an open-source codebase that is mutable.

Note that almost nothing is truly immutable because a codebase or even hardware executing even the most immutable protocol can change its behaviour, or it can be changed. In theory, a truly immutable system could be achieved using various hash constraints within TEEs; however, allowing for upgrades is a more desirable outcome. Ultimately, for all other cases, there is an explicit governance process somewhere.

Bitcoin miners, for example, have some power to determine the rules by choosing which version of the core code to install and to produce blocks with. If there is disagreement, there is a fork, and the user community ultimately decides what value to assign to each fork. This is only a problem if the competing forks have similar mining power, and thus security. For day-to-day upgrades, miners have the de-facto decision power, but in case of disagreements, the users have the ultimate power through free markets. This is currently the golden standard for decentralised governance, with advantages and disadvantages.

It gets even more complicated on networks like Ethereum with smart contract capabilities. On the one hand, similar to Bitcoin, the end-users decide which miners have chosen the correct version. On the other hand, the applications running on top of Ethereum have their governance requirements. In the early days, *The DAO* fell into the second category: *Implicit controls implemented in an immutable protocol.*, but it was exploited, and in addressing this by forking Ethereum and indirectly creating Ethereum Classic, it became apparent that there was actually a mutable codebase behind the immutable protocol (the Ethereum codebase itself). It also became apparent that users have the ultimate power as they indirectly voted with their wallets on the preferred approach of handling that hack, and Ethereum Classic has much lower adoption than the mutated Ethereum.

After that hard lesson, most Ethereum smart contracts have component contracts that can be upgraded through an explicit governance process since it is unlikely the community will again provide "get out of jail free" cards to application developers. Sometimes the governance is obfuscated, but generally, if the contract is *upgradeable*, it means someone is in charge.

The key difference between the golden standard of Bitcoin, and typical smart contract governance, is that the end-users no longer have any power to choose which "smart contract fork" they prefer. Using the original smart contract and adding some value to it, they are at the mercy of the application governors.

Since the Obscuro protocol is anchored in Ethereum as a smart contract, it cannot rely on Obscuro end-users to hold the ultimate power. The next best thing is to be very explicit about all the system's controls and achieve separation of decision-making (which can be devolved to token-holders and articulated in a governance specification as proposals) from execution (which relies on individuals pushing buttons).

Obscuro Controls

Building on the above, the following controls are exercised within Obscuro.

1. The TEE Attestation Constraints.

The *Attestation Constraints* (AC) control which software is allowed to run inside the TEE and can process the user transactions and create the rollups. A group of independent, reputable, and competent security auditors has to analyse the code and approve it by signing it carefully. The constraints contain the keys of the *approved auditors*.

The parties who have the power to set the AC and thus appoint auditors ultimately control the software.

This concern is not entirely different from the smart contracts security auditors, except that typically users decide which auditors they trust by using or not using those contracts.

2. Administration Of Ethereum Management Contracts.

Like most other Ethereum applications, these contracts will have upgradeable parts to cater for bugs and new features. Whatever is upgradeable means that the *administrators* have full control over those aspects.

1. Bridge logic
2. Rollup logic
3. Attestation logic

In the example above, the auditors are a fixed list. However, that might not be practical, as companies might appear or disappear. The list of approved auditors has to be managed by a proposal and vote process by the community without any requirement for central intervention. Going a level deeper, the code that manages this process might need to be upgradeable, so someone ends up controlling it.

3. Creating Rollups

Another power, equivalent to the L1 stakers or miners, is held by Obscuro Aggregators. They run attested software and hardware and have paid a stake.

They have the power to append to the L2 ledger, but they do not have the power to choose competing software and thus create forks.

4. Canonical Rollup Chain.

In a typical L1, the canonical chain is ultimately decided by its users from one of the competing forks because the ledger is ultimately coupled to the value of the coin.

In Obscuro, the Aggregators have to run attested software, which constraints their free will unless they can hack the TEE technology.

According to the rules implemented, a valid TEE does not sign a rollup building on top of a chain that is not canonical, so any hack is immediately visible.

Additional complexity involves the withdrawal process, which depends on assured finality on the canonical chain.

5. Slashing The Stake Of Misbehaving Parties.

Aggregators that hack an enclave and attempt to break the ledger's integrity are discovered by the protocol and are punished by slashing to disincentivise such behaviour further.

Slashing is an implicit process carried out by the Management Contract based on predefined rules. However, ultimately it is itself controlled by the code governance.

6. Expected Monthly Operational Cost For Nodes

Obscuro has a fee structure that delivers a predictable income for node operators and a predictable fee for users. In order to derive a fee that sufficiently compensates nodes, a value that represents the monthly operational cost for each node must be set. This variable also has the power to increase or decrease demand for running a node helping ensure a balance between decentralisation and end-user cost.

Appendix

Contributors

The Obscuro project is decentralised in nature, and this whitepaper has benefited enormously with feedback from the following contributors:

- Richard Gendal Brown
- Mike Hearn
- Moritz Platt
- Tim Brinded
- Fred Dalibard
- Stefan Iliev
- Zbigniew Czapran

Additional feedback is welcome, and all reviewers will be credited.

Glossary

Aggregator A node that participates in an L2 network and collaborates with other Aggregator nodes to manage the L2 contracts and confirm correctness of transactions. Specifically, it participates in transaction gossip, and may propose transaction rollups to be registered with the L1 blockchain.

Attestation Constraints Means of controlling which software is allowed to run inside the Trusted Execution Environment.

Automated Market Maker / AMM Uses liquidity pools to allow digital assets to be traded automatically and without permissions.

Block Reward An amount of OBX which is given to node operators to cover their costs to validate and publish blocks.

Enclave Key / EK Collection of one or more cryptographic keys used for encrypting and decrypting data unique to a specific enclave, digitally signing data and identifying a Trusted Execution Environment.

ERC-20 Ethereum Request for Comments 20, proposed by Fabian Vogelsteller in November 2015, is a token standard that implements an API for tokens within Smart Contracts.

Ethereum Virtual Machine / EVM A virtual computer whose existence is maintained by thousands of connected real-world computers running an Ethereum client.

Gas The unit that measures the amount of computational effort required to execute specific operations on the Ethereum network.

Gas Price The levy imposed for every computation executed on the Ethereum network to encourage good behaviour, e.g. prevent bad actors from spamming the network.

Genesis Enclave The first Trusted Execution Environment to join a new network. The Genesis enclave propagates the master seed to the other attested nodes by encrypting it with specific Trusted Execution Environment keys.

Host The party controlling the physical server that runs the Trusted Execution Environment. In the threat model of typical confidential computing applications, including Obscuro, the host is an adversary of the system.

L1 Management Contract The smart contract that runs on Ethereum and handles all L1 concerns.

Layer 1 / L1 The public Ethereum blockchain and network.

Layer 2 / L2 A second network built on top of an L1 network and dependent on it. An L2 network expands on the capabilities of the L1 network by increasing capacity or enhancing functionality.

Maximal Extractable Value / MEV Participants in the network may extract value by observing user transactions and then preempting them by inserting their own transaction ahead in the processing queue and influencing the price of an asset in order to extract a profit.

Non-Fungible Token / NFT A unique and non-interchangeable unit of data stored on a digital ledger.

Obscuro Public Events Special events emitted by L2 contracts that are included in the rollups in plaintext, and are exposed to L1 contracts once rollups reach finality. It is a mechanism by which Obscuro can publish information.

OBX The utility token used by Obscuro.

Off-Chain Activity happening away from the Layer 1 blockchain.

Optimistic Rollup Optimistic rollups assume that all transactions are valid and submit batches without performing any computation. They include a challenge period during which anyone can dispute the legitimacy of the data contained in a batch. If a fraudulent transaction is detected, the rollup executes a so-called fraud proof and runs the correct transaction computation using the data available on Layer 1.

Over-the-counter / OTC A venue to provide bespoke financial agreements or options negotiated between counterparties as opposed to being listed on an exchange.

Patricia Tree Root A Patricia Tree (or Trie), is a data structure used in the Ethereum model to represent the receipt trie, the world state trie, the account storage trie, and the transaction trie. Only the root node of the trie is stored in the ethereum block, and it represents a single cryptographic proof for the entire state.

Proof Of Block Inclusion / POBI Obscuro's novel decentralised round-based consensus protocol based on a fair lottery and on synchronisation with the L1 designed for L2 rollups.

Rollup L2 solutions that perform transaction execution outside the main L1 chain, but post transaction data on L1. A rollup is a batch of transactions that were executed by the L2 Verifiers.

Root Provisioning Key / RPK A cryptographic key randomly created and retained by Intel. It is the basis for how the processor demonstrates that it is a genuine Intel SGX CPU at a specific trusted computing base.

Root Sealing Key / RSK A cryptographic key that is unique to an enclave which that enclave uses to encrypt and decrypt data stored outside the enclave boundary.

Sequencer A sequencer is the selected Aggregator which builds a rollup in a round.

SGX Software Guard Extensions, a technology provided by Intel, a major CPU manufacturer. An SGX CPU has an area for encrypted computation, which the operator cannot access, secured by a private key burnt into the CPU during manufacture.

Smart Contract / Contract A user application running on a blockchain network which holds data or state, responds to user commands, and may store and manage assets or money.

Stake A non-negligible amount of value which is given over to an activity or process to demonstrate commitment to follow the rules for that activity or process.

Trusted Execution Environment / TEE An environment where contracts may be managed in a deterministic, repeatable and auditable way, based on a set of trust dependencies.

Trusted computing base / TCB The set of computing technologies that must be working correctly and not be malicious or compromised for a security system to operate.

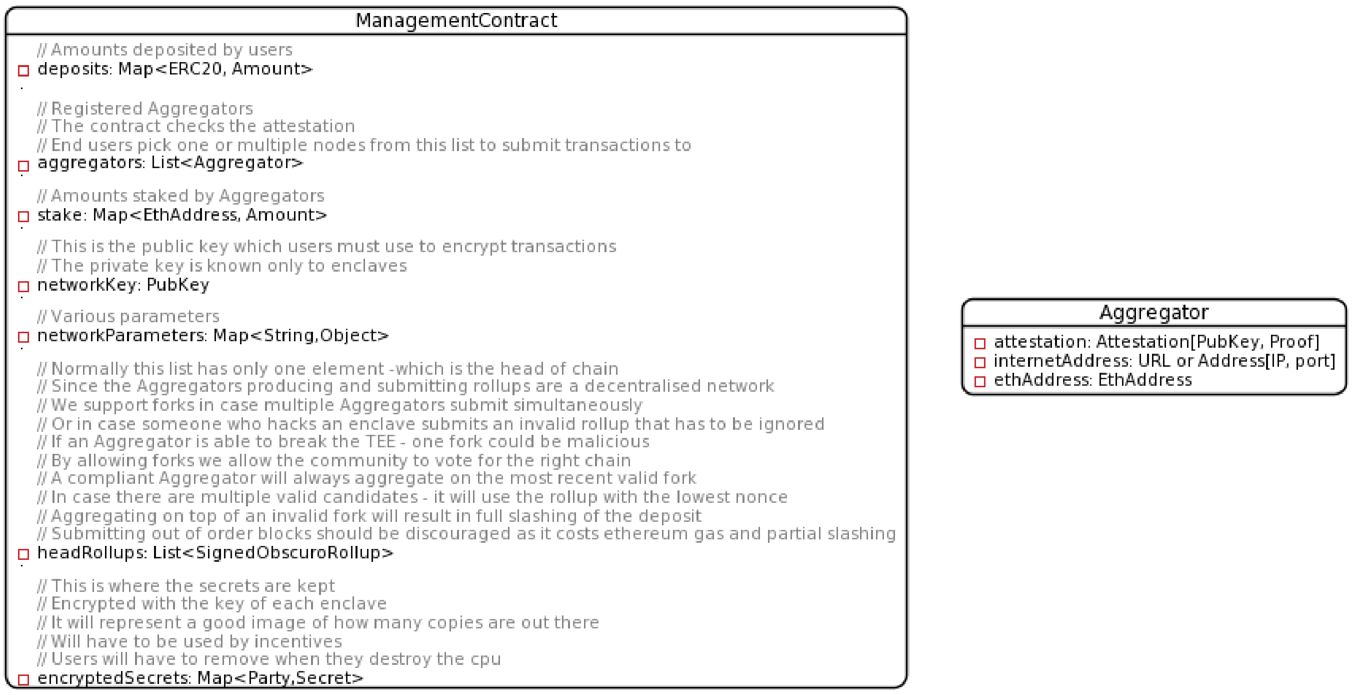
Utility Token Tokens which are intended to provide digital access to an application or service.

Verifier A *light* L2 node which observes transaction rollups published to the L1 blockchain, and can participate in possible disputes. Any Verifier can become an Aggregator by registering on the L1 contract and pledging some stake.

ZK-rollups Zero knowledge rollups generate cryptographic proofs that can be used to prove the validity of transactions.

Data Model

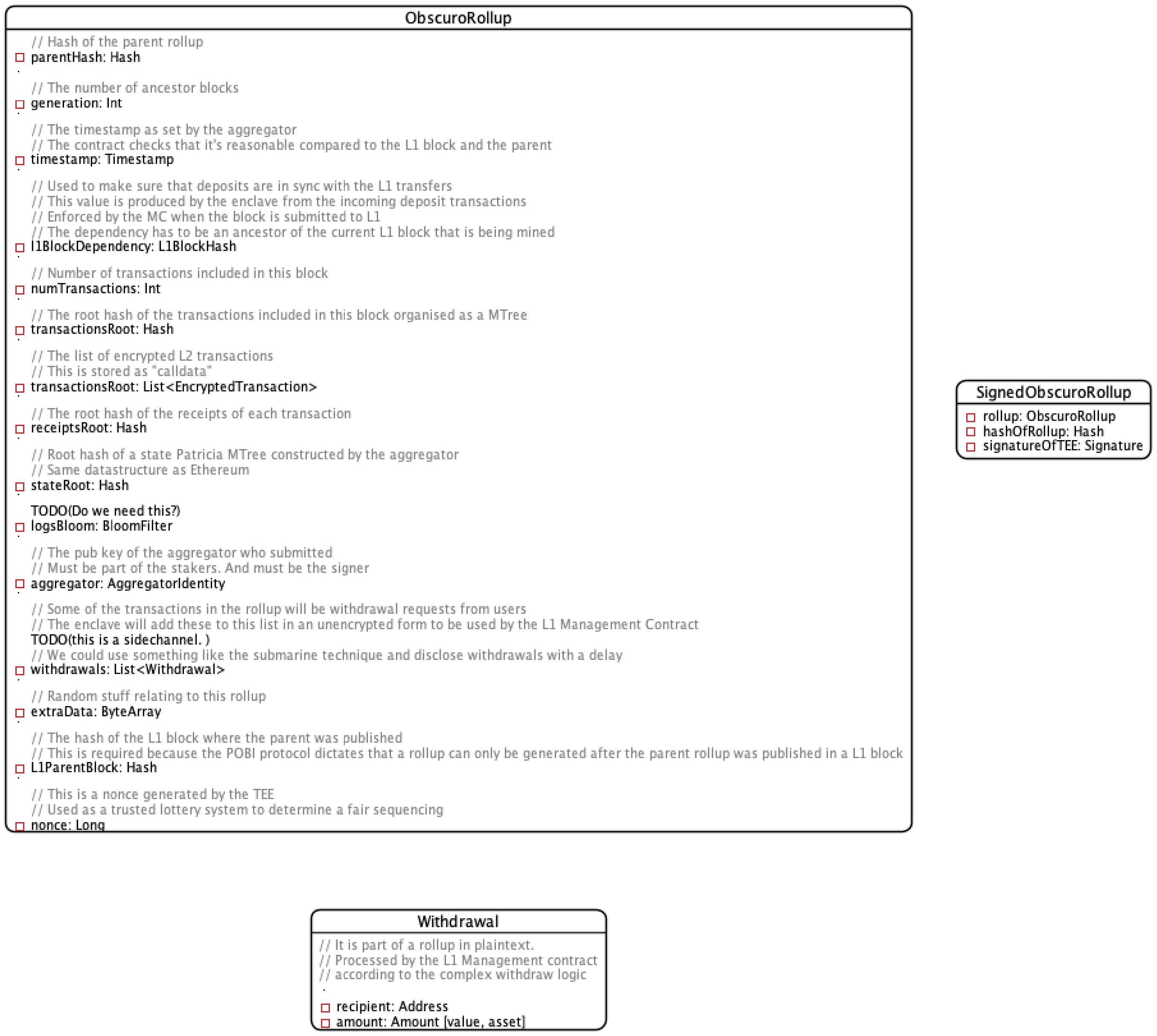
This diagram shows the data structure for the Management Contract and Aggregator:



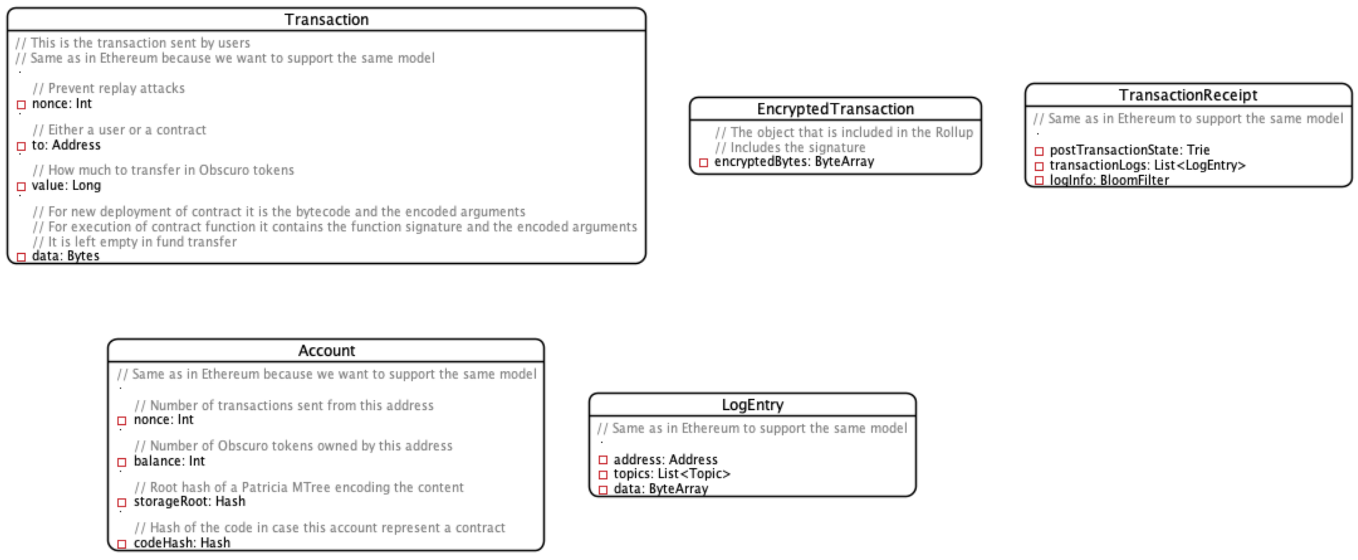
```
ManagementContract
  // Amounts deposited by users
□ deposits: Map<ERC20, Amount>

  // Registered Aggregators
  // The contract checks the attestation
  // End users pick one or multiple nodes from this list to submit transactions to
□ aggregators: List<Aggregator>

  // Amounts staked by Aggregators
□ stake: Map<EthAddress, Amount>

  // This is the public key which users must use to encrypt transactions
  // The private key is known only to enclaves
□ networkKey: PubKey

  // Various parameters
□ networkParameters: Map<String,Object>

  // Normally this list has only one element -which is the head of chain
  // Since the Aggregators producing and submitting rollups are a decentralised network
  // We support forks in case multiple Aggregators submit simultaneously
  // Or in case someone who hacks an enclave submits an invalid rollup that has to be ignored
  // If an Aggregator is able to break the TEE - one fork could be malicious
  // By allowing forks we allow the community to vote for the right chain
  // A compliant Aggregator will always aggregate on the most recent valid fork
  // In case there are multiple valid candidates - it will use the rollup with the lowest nonce
  // Aggregating on top of an invalid fork will result in full slashing of the deposit
  // Submitting out of order blocks should be discouraged as it costs ethereum gas and partial slashing
□ headRollups: List<SignedObscuroRollup>

  // This is where the secrets are kept
  // Encrypted with the key of each enclave
  // It will represent a good image of how many copies are out there
  // Will have to be used by incentives
  // Users will have to remove when they destroy the cpu
□ encryptedSecrets: Map<Party,Secret>
```

```
Aggregator
□ attestation: Attestation[PubKey, Proof]
□ internetAddress: URL or Address[IP, port]
□ ethAddress: EthAddress
```

This diagram shows the data structure for the rollup and withdrawal:


ObscuroRollup
// Hash of the parent rollup
parentHash: Hash
// The number of ancestor blocks
generation: Int
// The timestamp as set by the aggregator
// The contract checks that it's reasonable compared to the L1 block and the parent
timestamp: Timestamp
// Used to make sure that deposits are in sync with the L1 transfers
// This value is produced by the enclave from the incoming deposit transactions
// Enforced by the MC when the block is submitted to L1
// The dependency has to be an ancestor of the current L1 block that is being mined
l1BlockDependency: L1BlockHash
// Number of transactions included in this block
numTransactions: Int
// The root hash of the transactions included in this block organised as a MTree
transactionsRoot: Hash
// The list of encrypted L2 transactions
// This is stored as "calldata"
transactionsRoot: List<EncryptedTransaction>
// The root hash of the receipts of each transaction
receiptsRoot: Hash
// Root hash of a state Patricia MTree constructed by the aggregator
// Same datastructure as Ethereum
stateRoot: Hash
TODO(Do we need this?)
logsBloom: BloomFilter
// The pub key of the aggregator who submitted
// Must be part of the stakers. And must be the signer
aggregator: AggregatorIdentity
// Some of the transactions in the rollup will be withdrawal requests from users
// The enclave will add these to this list in an unencrypted form to be used by the L1 Management Contract
TODO(this is a sidechannel.)
// We could use something like the submarine technique and disclose withdrawals with a delay
withdrawals: List<Withdrawal>
// Random stuff relating to this rollup
extraData: ByteArray
// The hash of the L1 block where the parent was published
// This is required because the POBI protocol dictates that a rollup can only be generated after the parent rollup was published in a L1 block
L1ParentBlock: Hash
// This is a nonce generated by the TEE
// Used as a trusted lottery system to determine a fair sequencing
nonce: Long
SignedObscuroRollup
rollup: ObscuroRollup
hashOfRollup: Hash
signatureOfTEE: Signature
Withdrawal
// It is part of a rollup in plaintext.
// Processed by the L1 Management contract
// according to the complex withdraw logic
recipient: Address
amount: Amount [value, asset]

This diagram shows the data structure for the transactions and account:


Transaction
// This is the transaction sent by users
// Same as in Ethereum because we want to support the same model
// Prevent replay attacks
nonce: Int
// Either a user or a contract
to: Address
// How much to transfer in Obscuro tokens
value: Long
// For new deployment of contract it is the bytecode and the encoded arguments
// For execution of contract function it contains the function signature and the encoded arguments
// It is left empty in fund transfer
data: Bytes
EncryptedTransaction
// The object that is included in the Rollup
// Includes the signature
encryptedBytes: ByteArray
TransactionReceipt
// Same as in Ethereum to support the same model
postTransactionState: Trie
transactionLogs: List<LogEntry>
logInfo: BloomFilter
Account
// Same as in Ethereum because we want to support the same model
// Number of transactions sent from this address
nonce: Int
// Number of Obscuro tokens owned by this address
balance: Int
// Root hash of a Patricia MTree encoding the content
storageRoot: Hash
// Hash of the code in case this account represent a contract
codeHash: Hash
LogEntry
// Same as in Ethereum to support the same model
address: Address
topics: List<Topic>
data: ByteArray

Design Alternatives

This section describes alternatives considered and discarded.

Alternative L1 Deposit management

On a high level, a user has to deposit ERC tokens on the L1 Management Contract, and the same amount has to be credited to the user's account on Obscuro. This is not straightforward since finality is probabilistic. One option to achieve this is to wait a number of L1 blocks for confirmation. This has some clear disadvantages.

Another option is to introduce a dependency mechanism between the L2 rollup and the L1 blocks. Basically, the L2 transaction that credits the Obscuro account will be in an L2 rollup that will only be accepted by the Management Contract if the dependency is part of the ancestors of the current block. This option is discarded because in the case where the L1 deposit gets reorganised away before the rollup is created, the rollup which contains the L2 deposit transaction is invalidated.

Alternative L1 Theft Prevention

There is a pool of liquidity stored in the L1 Bridge contract, which is controlled by the group of TEEs who maintain the encrypted ledger of ownership. Some users will want to withdraw from the L2 and go back to L1, which means the Management Contract will have to allow them to claim money from the liquidity pool.

In case one of the Aggregators is able to hack the TEE, they will be able to produce a proof that they own much more and thus run with it.

To solve this we have a couple of options. We could organise the Aggregators in a BFT setup, and require that 2/3 of them sign over each rollup. The major disadvantage with this approach is that the finality of an L2 transaction will depend on both the BFT finality and the L1 finality. Another disadvantage is that a determined hacker with the means to break secure hardware could also amass the majority of staking power and be unchallenged.

Another option with a better trust model is to introduce a challenge mechanism similar to the optimistic rollups. The disadvantage is that it introduces a delay, and a concept of probabilistic finality.

The data structure containing the rollups is a chain that can have multiple heads. The Management Contract cannot evaluate which one is correct because it cannot execute the transactions inside. But there are some simple rules that can be applied. For example, if a branch does not progress for N blocks it is considered dead. If at the moment of withdrawal there is only a single active head rollup, then all the system has to do is wait for a reasonable number of blocks (20-50) to ensure that there is no censorship attempt on L1. If there is a fork, then the number of blocks has to be increased to allow one of the forks to die out naturally. If it does not then all withdrawals will be locked, and the contract will enter a special procedure.

Alternative Revelation Options

The solution will reveal all transactions after one year through a key-rotation process. An alternative policy could be to specify a ratio of transactions (e.g. 1%) are revealed either immediately or subsequently. Illegal transaction detection then becomes risk-based, but the ratio cannot be high enough to be a disincentive and yet still provide utility.

Alternative Nonce Generation

The Aggregator host must not be able to repeatedly submit the rollup proof to the TEE to get a new random nonce, and thus achieve a low nonce in order to win the Aggregator selection round. Monotonic counters were considered but an alternative is to make the nonce deterministic. The nonce is deterministically derived from the L1 block hash combined with the public key of the enclave. This achieves the same purpose of being a fair lottery assuming there is no collusion between L1 miners and L2 Aggregators. Even if there was collusion, the cost of gaming the L1 hash might be too high in a proof of work network. In a *Proof of Stake* network, on the other hand, collusion with L1 would pretty much mean that each round the L1 winner will also win the L2 round.

Alternative Privacy Revelation

There are several options for revealing private data to allow law enforcement agencies to prosecute illegal behaviour and deter criminals from taking advantage of Obscuro's privacy features:

- Not make a provision to reveal on the basis that Obscuro is a platform and is un-opinionated on what it is used for.
- The transaction encryption key can be rotated and revealed periodically with a delay, such that any interested party can view all transactions. This is the solution we chose, but with some application-level flexibility.
- A governance committee can approve some data mining enclaves that will have access to the shared secret and output suspicious activity.

From the outset Obscuro will rotate the encryption key every year and reveal historic keys in the first phase, and decide later if additional mechanisms are required. A case-by-case revelation based on authority demands is time-consuming and prone to political interference. It is also difficult to determine objectively what is a bona-fide authority which introduces a political dilemma.

[Page history]({{ site.github.repository_url }}/blame/main/{{page.path}}) FILE: ./technical-background.md

2 links checked.

FILE: ./consensus.md ✓ ./obscuro-ethereum-interaction is dead

8 links checked.

FILE: ./threat-model.md ✓ ./obscuro-ethereum-interaction#withdrawals is dead ✓ ./rollup-data-structure is dead

3 links checked.

FILE: ./high-level-design.md ✓ ./l1-contracts is dead ✓ ./cryptography is dead

5 links checked.

FILE: ./abstract.md ✓ ./appendix#contributors is dead

5 links checked.