Enhanced Immutability of Permissioned Blockchain Networks by Tethering Provenance with a Public Blockchain Network

Azeem Ahmed (azeem.ahmed@consensys.net)
Jim Zhang (jim.zhang@consensys.net)

Abstract

Permissioned private chains are susceptible to attacks where legitimate parties on the permissioned network collude to create an alternate chain by rewriting the chain from some point-in-time in the past; thereby, weakening the immutability guarantees of a blockchain network. This paper discusses such an attack vector and proposes a way to tether the provenance for permissioned and private chain state transitions to a public blockchain network (which has better guarantees around immutability). This is accomplished by *tethering* a private chain to a smart contract on the Ethereum main-net that records an irrefutable commitment of the current chain state. An external service is thus, enabled to provide guarantees on the private chain state (with an appropriate proof) and allow the detection of a compromised private chain due to collusion-based history rewrite by legitimate permissioned parties.

1 Table of Contents

1 BACKGROUND 1
1.1 CONSENSUS ALGORITHMS 1
1.2 ATTACK VECTORS 1
1.2.1 Compromised Access Layer 1
1.2.2 Collusion between Permissioned Parties 2
2 STATE PROVENANCE THROUGH MAIN-NET TETHERING 3
2.1 INTRODUCTION 3
2.2 DEFINITIONS 3
2.3 ARCHITECTURAL OVERVIEW 3
2.4 DETAILS OF TETHERING A NETWORK 3
2.4.1 Deploy Arbitrator on Main-net 3
2.4.2 Register Permissioned Network 3
2.4.3 Synchronize state with the main-net periodically 4
2.4.4 Checking state agreement between nodes 5
2.4.5 Policy enforcement when state disagreement is detected 6
2.4.6 Addition or Removal of Nodes from the Network 7
3 CONCLUSION 7

1 Background

Enterprise organizations are looking to exploit blockchain based distributed data storage and compute frameworks to realize the benefits of cooperation with potentially adversarial parties for efficient execution of inter-party transactions relying on the guarantees provided by a distributed ledger. For several reasons (including but not limited to transactional privacy, trade secrets, and performance), most enterprises seek to utilize their own private permissioned network (aka consortium chains) for a distributed ledger. These networks do not employ proof-of-work for consensus and generally should not employ PoW because a PoW network is only as secure as the computing capacity behind it. Instead, variants of BFT algorithms or crash-tolerant algorithms are employed for consensus (depending on the use-case of a particular deployment). These consensus algorithms lead to significant benefits for permissioned chains as transactions can settle with significantly lower latency. As reasoned above, proof-of-work would not provide the requisite security for these networks.

Although, a permissioned network that has an access control layer backed in as a first-class citizen is not susceptible to the same attack vector as an un-permissioned network, it is susceptible to a different subset of attack vectors: adversaries getting access to such a network by compromising the access control layer or collusion of sufficiently large number of legitimate permissioned parties within the network. In the later scenario, the immutability of the chain can no longer guaranteed even with immediate finality of the blocks. The type of collusion scenario is often times handled by legal entities that reside off-the-chain but there is still not an easy way to arbitrate and prove the state of the chain at a given point-in-time in all parties on the chain collude. The concepts introduced in this paper allow any interested party to reliably detect, verify, and potentially recover from collusion between trusted parties on a permissioned network.

1.1 Consensus Algorithms

Permissioned networks tend to employ a variety of consensus algorithms that can be broadly classified as either byzantine fault tolerant or crash fault tolerant. All such algorithms provide instant finality of the blocks. While a comprehensive discussion around these algorithms is beyond the scope of this paper, it is worth it to note that the attack vectors described in here do not cover the full gamut of these algorithms and only discuss a subset of these algorithms (RAFT, IBFT, and PoA implementations like Clique, and Aura). Nonetheless, the concepts discussed can be employed for any class of algorithms that fit into these categories and have similar susceptibilities.

1.2 Attack Vectors

Before we discuss the proposed Tethering concept, it is prudent to briefly discuss the attack vectors on a permissioned network. These can generally be categorized into two areas: gaining access to the network by circumventing or breaking through the access controls, or sufficient number of legitimate permissioned parties colluding to create an alternate version of truth on the network. While we briefly discuss the first category of attacks, this paper is primarily focused on the second category namely the issue of history rewrite by sufficiently large number of colluding parties in a permissioned network.

1.2.1 Compromised Access Layer

The attack vector of a compromised node is generally addressed by ensuring proper access control and following best practices in security. It is, however, worth it to briefly discuss this attack vector and its implications. In general, we need to discuss the ability of an attacker to rewrite history and propose new rogue/invalid blocks to the network.

1.2.1.1 Using RAFT

In a permissioned network using the RAFT consensus, this attack can be successful if a rogue party gains access to the network and is elected leader for the RAFT consensus. In such a scenario, rogue new blocks can be proposed that the network will consider the truth. RAFT consensus is susceptible to this attack vector because RAFT followers are required to blindly trust the leader's proposal. There is little that can be done to prevent such an attack after a rogue party gains access to a node on the network. Additionally, any rogue node could rewrite its own chain without

proposing it to the rest of the network (leaving the party that relies on the particular node susceptible to invalid history). The first scenario of a rogue leader proposing compromised blocks to the network can't be protected against given that RAFT is simply a crash tolerant algorithm. However, the second scenario of a local rewrite of history could be detected and recovered from using the proposal in this paper.

1.2.1.2 Using BFT variants

In a permissioned network using the PBFT consensus, or one of its many variants such as IBFT, this attack is limited to a local rewrite of history. The PBFT consensus algorithm is designed to tolerate invalid block proposals by the rogue party because the honest validators will reject the block. However, new rogue blocks can be proposed and accepted if a rogue party controls the super-majority of the validators and the proposer of the blocks (which would mean that the network is severely compromised). This proposal does not address this scenario.

1.2.2 Collusion between Permissioned Parties

Our main focus is the detection of a history rewrite by a sufficient number of colluding parties on the network and to reliably reinforce immutability of permissioned chains. To that extent, let us briefly look at what collusion may look like on a permissioned network. Again, we will look at proposing of new blocks on the network and rewrite of the history.

1.2.2.1 Using RAFT

In a permissioned network that utilizes RAFT consensus, any party can collude with the leader of the network and introduce new blocks in the network that are invalid. The blocks are simply accepted by all parties when proposed. Such collusion can't be prevented, and RAFT consensus is not recommended for permissioned networks that are concerned about adversarial parties on the network. This attack is equivalent in its effect to a rogue party gaining control of the leader node by breaking through the access control layer.

Multiple parties may also collude and rebuild the chain (rewrite history). As the block finality is instant, the rewrite of the history will be limited to the parties involved and will not propagate on the network. When new blocks are proposed with the new chain, they will be rejected by the other nodes in the network because the chain is not valid. At this point, there is no way to tell who was 'honest' as the colluding parties can easily suggest that the 'honest' party is in fact 'rogue'. This proposal in this paper sufficiently addresses this later scenario.

1.2.2.2 Using BFT variants

In a permissioned network that utilizes a BFT variant consensus, multiple parties may collude and rewrite their own history and start proposing and validating new blocks based on the new rewritten chain. This requires collusion between a super majority of validators (and in a network where every node is a validator, this requires the super majority participation of the network). However, as is true with RAFT, since the block finality is instant, the rewritten chain will not propagate on the network and will continue to be rejected by the honest nodes. A history rewrite is not simple in this case as in the RAFT consensus which does not have the signature of proposer on the block (as it exists today) and identifying the honest party is problematic. The BFT variants generally have some sort of a signatory mechanism that is stored on the block, so collusion between parties would be sufficiently difficult and unless all parties in the consortium collude on the new chain, the honest parties can easily be identified since every block has the signature of the original validator(s). In theory, this minimizes the surface area of this attack vector but there is still no recourse if the whole consortium colludes to rewrite some part of the history. The proposal in this paper enables an external party to validate that no history rewrite has occurred on the network. External parties include, at minimum, distributed applications that interact with the consortium network, auditors who need to verify the veracity of the network, or proxy nodes that interact with the network using a node provided by a permissioned party on the network.

2 State Provenance through Main-net Tethering

2.1 Introduction

This paper proposes a method to maintain an irrefutable proof of the state of a permissioned network using a combination of a monitor node and a smart contract on the Ethereum main-net. The state of a permissioned network is defined as the full-chain of finalized blocks on the permissioned network at any given point in time. The main-net is significantly more hardened against tampering and can be leveraged as the source of truth when there is disagreement between parties in a permissioned network. Using this record on the main-net, any interested service or party can provide proof of network compromise based on state deviations.

2.2 Definitions

This paper employs several terms that are defined here for ease of consumption.

- **Environment**: An environment is a permissioned network which has one or more nodes.
- **Network***:* Equivalent to an environment and is used synonymously in this paper.
- **Node**: A node within a permissioned network is owned by a specific party in the permissioned network and has a unique identity. Each node also has a public/private key pair that is maintained securely to sign all messages that originate from the node.
- **Monitor Node**: A read-only node within a permissioned network run by the auditor (or a platform provider) in a security hardened container with no external access and audited maintenance access.
- **Main-net Tether Service**: This service is collocated with a chain client in the node and is continuously monitoring the node for state changes. This can be implemented using web3 (or a variant) to monitor all blocks being finalized or a lower level client can be built that monitors the leveldb store directly.
- **Relay**: A relay service is an off-chain service that may be leveraged by nodes to transact with the main-net (and reduce gas costs). The relay also has access to a funded account for transactions on the main-net.
- **Notary**: A notary service is an optional off-chain service that periodically checks the main-net for state agreement between all nodes within an environment ((including the monitor node) and raises a red flag if a disagreement exists. The notary service also has a funded account for transactions on the main-net.
- **Sentry**: A sentry service is an off-chain real-time analytics engine that continuously monitors the network (and the monitor node). Based on a pre-defined policy, the sentry can take an immediate action on a permissioned network a.k.a Environment when collusion is detected.
- **Arbitrator**: The arbitrator is a set of smart-contracts on the main-net that together accumulate state of all nodes within a permissioned network and allow retrieval of previously reported states for any given environment and node within an environment. The Arbitrator smart contract provides a notarized proof of state agreement between all nodes in a permissioned network (including the monitor node).

2.3 Architectural Overview

The diagram in Figure 1 depicts all components in a sample tethering implementation.

2.4 Details of Tethering a Network

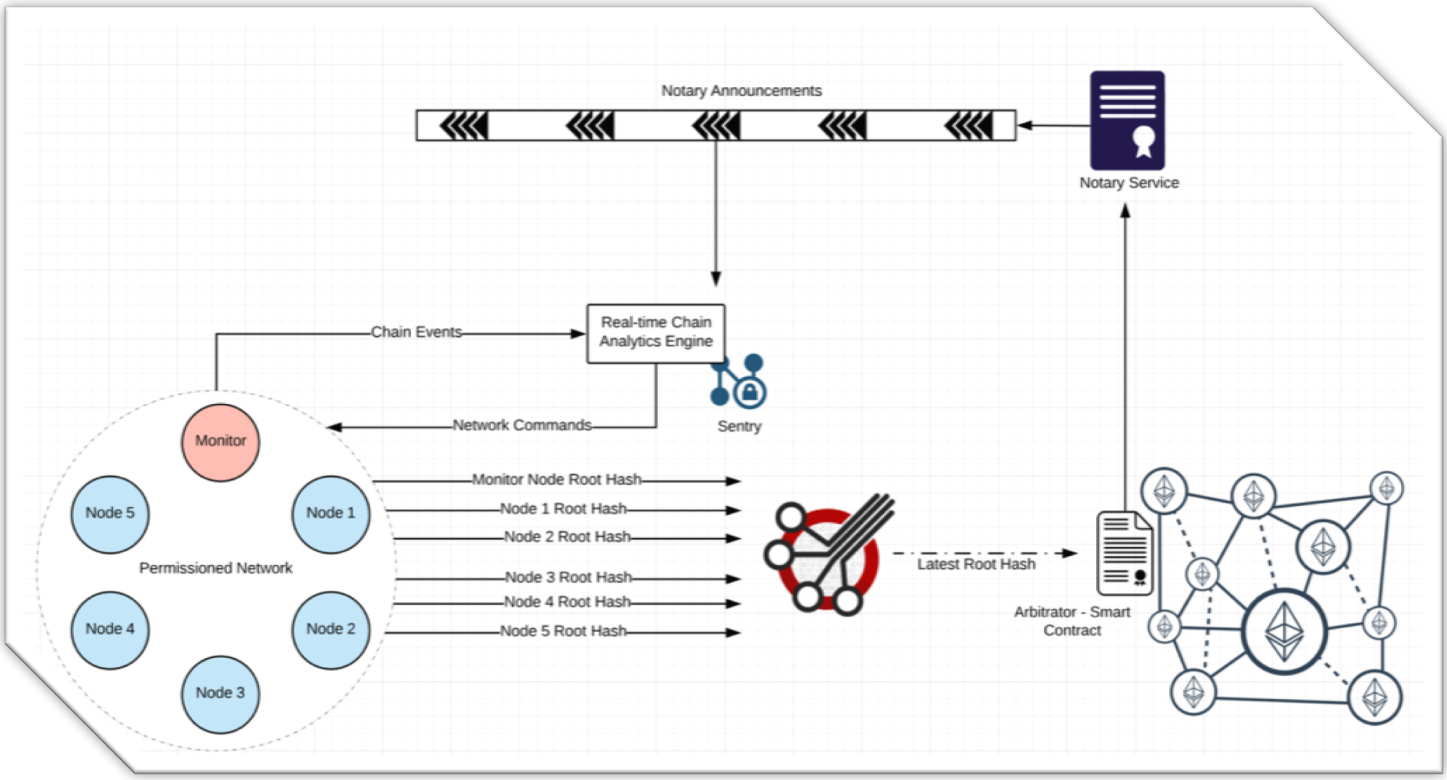
2.4.1 Deploy Arbitrator on Main-net

An arbitrator smart-contract is deployed to the main-net so permissioned networks can tether with it. This can happen any time before the first permissioned network that requires tethering is brought online.

2.4.2 Register Permissioned Network

When a tethered permissioned network is initially launched, every node on the network is configured with a private/public key pair that is used by the node to digitally sign any outgoing messages. An additional monitor node is also configured to run within the network in a read-only capacity (as an observer only). When the network is initially provisioned, a hash of a JSON object that represents the policies that govern the network is stored by the Arbitrator along with a hash of the genesis block. The policy itself is stored off-chain (and may contain any number of rules including but not limited to rules governing how often the state

Figure 1: Data flow diagram in a tethered permissioned network



must be sync'ed with the main-net, policies around additional nodes joining or leaving the network etc.).

When each node on the permissioned network comes online, the node registers itself with the Arbitrator as a valid node on the network by sending a message to the contract that is digitally signed by the node's public/private key pair. After successful registration, each node has access to and can retrieve the policy hash from the arbitrator and validate the hash of the genesis block against its own genesis block.

This protects the genesis block from being rewritten or changed (for some consensus algorithms, the initial list of validators may be encoded in the genesis block).

For privacy of the network and nodes, all IDs registered with the main-net are KECCAK-256 hashes of the ID.

2.4.3 Synchronize state with the main-net periodically

Each node on the permissioned network (including the monitor node) must collocate a Main-net Tether Service with the chain client that is processing transactions. The Main-net Tether Service continuously monitors any blocks being generated and maintains an internal *Patricia Merkle Tree* that stores the key-value pairs of block numbers and block hashes. Every single block finalized by the chain client is immediately (with some configurable delay) added to the *Patricia Merkle Tree*. This service will periodically, based on the block interval defined in the network policy:

a. Retrieve the root hash of the *Patricia Merkle Tree*
b. Create a payload with the root hash, KECCAK-256 hash of its own unique identifier, and a KECCAK-256 hash of the unique identifier for the environment
c. Digitally sign the payload and send it to the main-net.

Note that this is not using a time-based interval as a timer-based sync can't guarantee block height agreement. Instead, the interval is defined in blocks (eg: every 100th block or every 10th block).

The root hashes can be sent to the main-net using one of two methods. Each Main-net Tether service can

directly invoke the Arbitrator method to store the state at a periodic interval (Distributed Sync) or simply send its root and signature to the relay queue (Relay-based Sync). In a Distributed Sync, the cost of transacting on the main-net increases linearly with each node and the number of transactions are determined by the block growth rate (i.e. if blocks grow at an exponential rate, the transactions can also grow at an exponential rate) depending on the policy. A policy can be built to be more dynamic based on rate of growth. To avoid the high-cost of transaction a Relay-based sync can be utilized. In a relay-based sync, the Main-net Tether service will send the root hash at the block interval defined by the environment policy to an internal relay (via a reliable transport mechanism that provides at-least-once delivery guarantees). The Relay service can accumulate reports for a period of time and send a single report to the main-net for the most recent block number with reports from all the nodes in the network. If there are block numbers that do not have reports from every single node in a network, the Relay will hold those reports in persistent storage for a period of time called the *report expiration period*. The *report expiration period* is defined by the environment policy. The Relay will clear its storage and send all reports available to it if the *report expiration period* passes and the latest block reported still has some nodes that have not reported their state.

The Main-net Tether service provides a RESTful service endpoint to generate proof for any given key-value pair in the *Merkle Patricia Tree* it maintains. When invoked, it validates the block hash against the block in the chain and its copy of the *Patricia Tree* and returns the generated proof which can be used to verify against a root hash. It also provides an endpoint to verify a given proof against its own *Patricia Merkle Tree* hash root.

Additionally, to ensure that the Main-net Tethering Service itself is not tampered with, it must be protected from all access except through recognized endpoints. The container that runs it must follow best practices for security that are beyond the scope of this document. Ultimately, there needs to be clear guarantees that no one party can access this code or modify it (including the administrators and the development teams that develop this code).

2.4.4 Checking state agreement between nodes

Each network is configured with one or more Notary services that periodically retrieves the state of network from the main-net and checks to see if all nodes are in

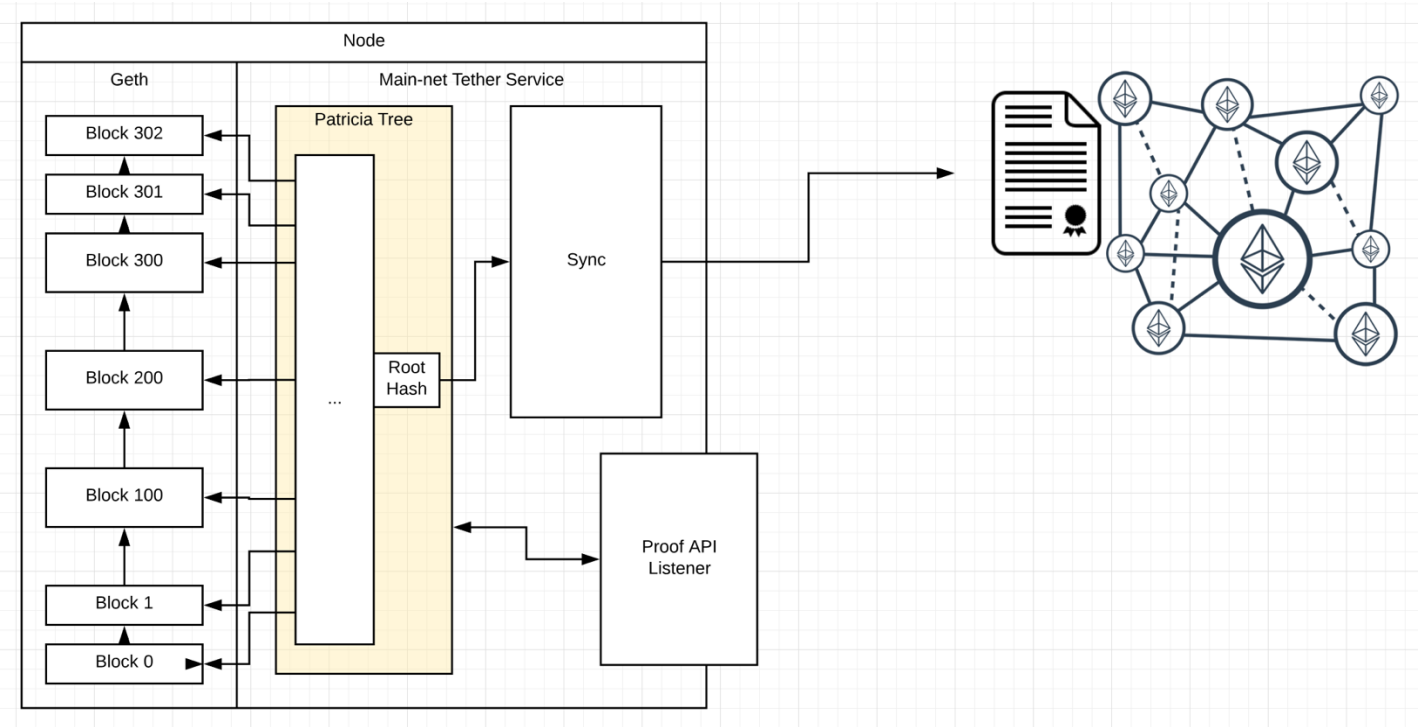


Figure 2: Main-net Tethering Service in a Distributed Sync setup

agreement. The interval and schedule that the Notary service utilizes for a network is determined by the policy of the network. If there is a disagreement detected i.e. not all nodes reported the same hash root, the Notary service sends an announcement using a reliable at-least-once delivery mechanism to all subscribers for such an announcement. This allows all interested parties to listen to such announcements to take an appropriate action. Such an announcement is digitally signed by the notary and can be validated to have originated with the notary.

Each member of a consortium may deploy their own subscribers to listen for such announcements and take action against their own node or an auditor can deploy a subscriber to listen for an announcement from the Notary.

The Notary service can also be implemented by any party interested in providing permissioned network guarantees by directly communicating with the smart contract.

An important note for the notary is that the notary is not aware of the actual unique IDs and only works off of the hashes (i.e. it is initialized with an environment ID hash, node ID hashes, and the interval derived from the environment policy).

The Notary service also exposes a set of APIs for validating that a particular block number and hash were in fact at one time part of the blockchain. The service requires the environment ID hash, the node ID hash, and a proof. It returns true if the proof is valid based on the main-net root hash or it returns false along with the root hash that was used to verify the proof. This service utilizes the main-net report for the environment (that was agreed on by all nodes) for the nearest block number greater than the block number being verified. Using the root hash stored in the report, it will verify that the proof being presented for the block number/hash being presented can be verified using the root hash stored in the main-net. If a disagreement exists for the nearest block, the Notary will verify the proof against all hashes and provides the result back to the caller.

Finally, the Notary service announces any approved changes to the network (addition or removal of nodes). See Addition or Removal of Nodes from the Network (2.4.6) for additional details.

In practical details, the Notary services invokes a method on the Arbitrator contract called *getChainStatus*. This method on the contract will return true if all nodes in an environment (identified by the parameters) agree on a state. The Notary service is simply providing a *push* based subscription instead of a *pull* implementation (which can be accomplished by directly invoking getChainStatus periodically).

2.4.5 Policy enforcement when state disagreement is detected

The Sentry is responsible for network policy processing and enforcement and will utilize the defined network policy to call the necessary services to take a specific action on the network (eg: pause the whole environment, notify administrators etc.). The Sentry registers itself as a subscriber to the Notary announcement and has an analytics engine that receives chain-level protocol events from the monitor node at a regular interval (eg: block rejections, peer communication failure etc.). It also has access to the hashes of all environments and nodes, so it can correlate the announcement to a specific network and take action on the network based on the policy. The sentry is capable of taking immediate action when the analytics engine reports possible problems with the chain (including collusion or compromised nodes). Any such actions can be further validated by the irrefutable proof of state saved on the main-net. Additionally, all actions taken by the Sentry are also recorded on the main-net to ensure there is an audit trail of the Sentry activity.

Off-chain agencies may also leverage the Notary announcements to restrict participation of the network or enforce off-chain penalties. In fact, the sentry may simply alert an off-chain entity for further investigation/inquiry. All such actions are a possibility with the design presented in this paper.

2.4.6 Addition or Removal of Nodes from the Network

When a permissioned member needs to be added or removed from the network, each node in the network sends a signed proposal to the arbitrator with the public key of the new node that is joining or leaving the network. The Notary service monitors for any approved additions or removals and announces these changes to everyone through a separate topic on the pub/sub transport that provides exactly-once-semantics. The Sentry service will listen for approved changes as well and reconfigures the network accordingly.

3 Conclusion

This paper presents a novel method of creating provenance for private permissioned network state by tethering them to the main-net. In the future, we hope to improve this method further by introducing efficiencies and remote attestation for components that currently have to run in a trusted manner (for eg: main-net tethering service). Additional future work may revolve around creating a modified client purpose built for permissioned networks that provides the benefits of the permissioned networks and simultaneously tethers the provenance for a permissioned network with the Ethereum main-net for enhanced immutability.