Blockchain Architecture

Purpose

The goal of this paper is to explore the blockchain implementation detail and architecture at a deeper level. It will focus on the constructs to facilitate the feature requirements of the white paper, and discuss structures and methods for dealing with the ongoing operation, security and performance of the ecosystem.

Blockchain Related Commercial / Operational Constructs

Feature and Policy to be Implemented

Commercial Fees: Transaction level fees we collect on the platform

Fiscal: Incentives we use to promote holding the Vera token or participation in the token

Monetary: Policy for handling the economic controls (mint/burn, inflation, velocity of money);

Derivative (Veraspark): Policy for Veraspark disbursements, trades, and rewards

Derivative (Totalspark): Policy for Totalspark disbursements, trades, and rewards

Policy Parameters:

- Mint Rate
- Burn Rate
- Reward Pool Drip Rate: Views
- Reward Pool Drip Rate: Verafiers
- Veraspark Hold Reward Rate
- Totalspark Reward Rate

Technical Layers of Implementation

Blockchain Layer: Hardcoded into the blockchain protocol itself

SmartContract Layer: Coded into the SmartContract itself

Application Layer: Coded into the website and backend management processes

Key Requirements

Proof of View

Proof of View is the immutable public record of all videos (ads and normal content) watched by the viewers. The history must be public and auditable by anyone.

Complex Financial Dynamics

The solution must enable complex financial transaction mechanisms. This will include monetary and fiscal policy, rewards systems, and profit sharing instruments. All the financial instruments must be tradable on the marketplace.

Spark Marketplace

The solution must provide the participants with a safe and transparent marketplace to trade all the available financial instruments.

High Performance

The solution should be capable of handling about 55k PoV transactions per second (in 6 years) as well as all the marketplace transactions.

Secure by Design

Steps must be taken to proactively reduce the attack surface for hackers. The architecture must allow privileged accounts and accounts with large balances to be kept with appropriate security mechanisms..

Decentralised and Trustless, but Managed by the Verasity Foundation

The architecture must leverage the trust and security advantages of decentralisation as well as facilitating the long-term economic sustainability measures of the ecosystem managed by the Verasity Foundation.

General Policy / Architecture Approach

To garner community trust and enhance security, it is desirable to put the Verasity Foundation into a position similar to that of the legislative branch of a government. The Verasity Foundation should guide rather than run the ecosystem. Where possible, the foundation should set the policy / rules and leave the enforcement to the community.

The most suitable implementation would involve Verasity releasing source code updates and having control over several privileged accounts that can alter certain key policy parameters.

If a centralized online account or service is absolutely necessary, sufficient risk reduction and damage mitigation provisions should be introduced to make hacking it unprofitable.

Secure by Design

Air Gap

Privileged accounts are kept offline, rarely used and remain out of reach of hackers. Verasity does not run any online services essential to the functioning and integrity of the financial system.

Verasity does not transfer money to anyone directly. Payments, VRA distributions and rewards are distributed by smart contracts, keeping the keys of accounts with large balances offline and unhackable.

Security Through Diversity

Security through diversity involves asking a number of qualified entities (motivated by profit and/or reputation) to perform identical tasks and discarding outliers. This greatly reduces the likelihood of hacking attempts and social engineering succeeding.

The proposed architecture relies on security through diversity to secure payments, VRA distributions and rewards by using smart contracts, instead of running a relatively easily hackable online service.

Privileged accounts are secured by requiring multiple signatures for transactions to take effect.

Trustless Preferable Over Audits

Even if a centralised body makes its policies and activity public and auditable, someone still has to perform audits, stakeholders then need to decide which auditors they trust and review the audits.

Verafiers are full DPoS nodes. The participants running full nodes audit everything themselves and therefore do not need to trust anyone else to verify a transaction. The participants using online wallets vote for Verafiers and trust and rely on them to audit the transactions.

The legislative branch approach makes it easier for stakeholders to trust Verasity as policies are applied transparently and there's no need to monitor day-to-day activities of the Verasity Foundation.

Decentralised Components

The degree of decentralisation of the proposed architecture is designed to maximize the uptime of the components of the system.

These fully autonomous components of the system are highly reliable and inherently designed without single points of failure:

- Financial system
- Markets
- Reward system
- VRA Distributions
- Verafier elections
- Transacting via a full node or an optional browser extension based wallet
- PoV history availability

Blockchain Implementation Selection

Graphene blockchain has been chosen as the most appropriate solution to meet our requirements:

- **Performance and scalability:** can handle PoV -- 3k TPS with a clear path towards 100k TPS
- **Production grade:** No fundamental stability or security issues uncovered by Bitshares and Steemit over the course of several years
- **Marketplace:** Provides a trustless asset marketplace that can be used as-is to trade VeraSparks and TotalSparks
- **Multi-Signature Security:** Multi-level multi-signature accounts that can model real-world power and organisational structures
- **Native smart contracts:** Faster and more flexible than any existing Virtual Machine-based implementation

- **3rd Party Vendor Reliance:** Primary selection criteria was to find an open source blockchain to reduce reliance on the success for 3rd party commercial entities.

Policy Implementation

In the Verasity architecture a policy is implemented as an algorithm or formula that takes multiple parameters as inputs. A policy algorithm is executed in a distributed fashion by Verafier nodes, validating and checking each other's outputs. Such an implementation is as close to unhackable as is possible in practice.

Policies are managed by updating these parameters (no software updates are necessary for this). Several secure approaches are detailed in the Secure Policy Management section.

Changes to the algorithm itself require a software update. The policy algorithms are being designed in a flexible way to minimise the frequency of software updates. In the case that a software update becomes necessary, we will handle it as explained in the Software Update Policy section.

Proof of View

Proof of View records are stored in the main blockchain as PoV transactions.

- To avoid synchronization challenges when tracking views, payments/ rewards, the optimal approach is to store all related transactions in the same ledger, and further, reusing the same data structure as both a view record and payment / reward transaction.
- Graphene was selected due to its ability to handle the required throughput for view/rewards transactions.
- The majority of transactions are PoV, and blockchain is a perfect fit for immutable history storage.

Creating PoV transactions

In order to ensure that PoV transactions are genuine views, multiple stages of validations will be made.

- Only viewers that are signed in to the system can generate POV transactions
- Bot detection mechanisms are natively built into the player to detect that a real human is actually interfacing the system
- Behavioral policies are put in place to ensure the activity and interaction with the system is "expected"
- VeraPlayer integrity challenges are implemented to ensure non-tampering
- A stream consumption challenge is implemented to ensure that the video has indeed been streamed and received by the VeraPlayer.

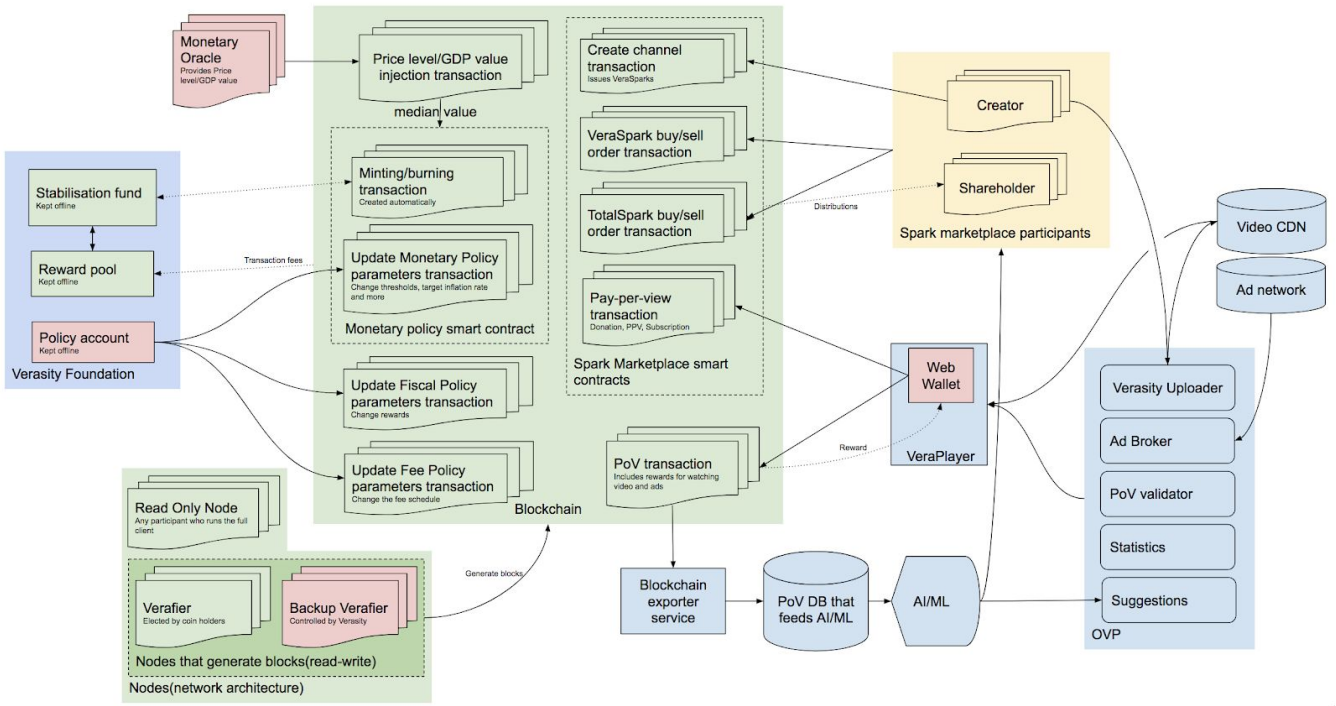
Post validation, the OVP signs the PoV record to confirm interaction validity and that the viewer has watched the video. The viewer's signature confirms that the view has indeed taken place. The OVP and VeraPlayer individually sign the PoV record to create a valid PoV transaction.

Ensuring Transaction Submission to the Blockchain

After a PoV transaction is created, both the OVP and VeraPlayer attempt to push identical copies of it to the blockchain. This ensures that neither party can censor / block a submission of a PoV record to the blockchain. Once at least a single copy reaches a Verafier, the transaction is added to a block, and serves as both an immutable record of the view event and as a reward / payment transfer transaction that debits the Reward Pool / payment wallet and credits the viewer's account. Duplicate transaction copies are discarded by the blockchain.

Implementation of the Required Features

High Level Process Overview



The Verasity process architecture is broken up into multiple participant groups.

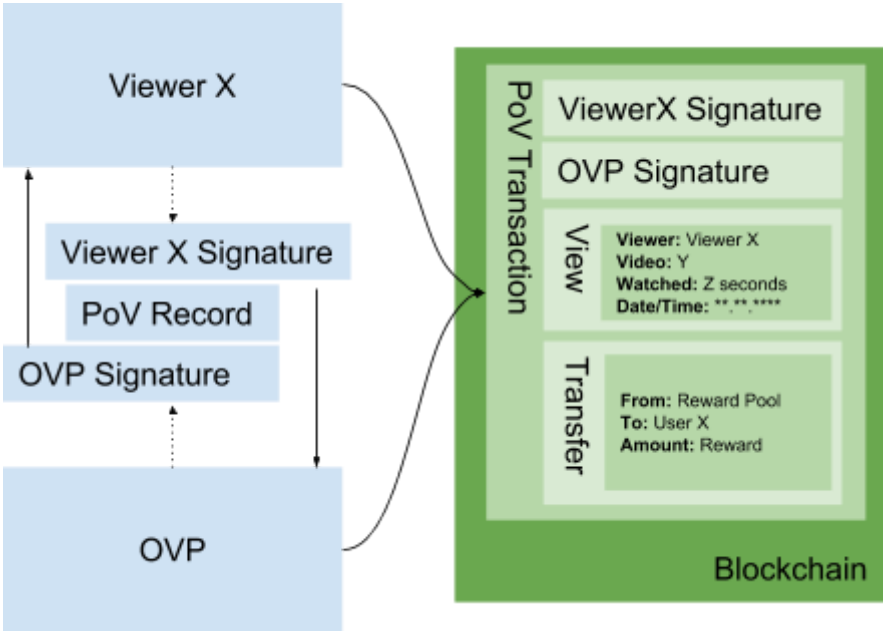
- Green: Fully Decentralized Components
- Red: Components administered by Verasity Foundation
- Blue: OVP (Online Video Platform) Components and Participants
- Yellow: Spark MarketPlace Participants
- Dotted line: Financial flows
- Solid line: Transaction submission

Proof of View

A PoV transaction serves as:

- an immutable and unforgeable record of a view event, signed by both the viewer and OVP
- a transfer of a reward to the viewer
- option: a payment for watching ads

The following diagram shows the interactions between the Viewer/OVP and the PoV transaction, as well as the proposed transaction structure.



Damage Mitigation

Reward Restriction by Policy

Errant reward distributions are mitigated through policy restrictions that invalidate out-of-bounds reward amounts. These smart contract based policies are a check-and-balance governance via a range of parameters. The policies will restrict the maximum reward amount and the maximum reward amount per second of video watched. These policies will be augmented as necessary with additional parameters.

Temporary Accounts

Temporary fund limited hot wallets will be used to administer reward pool distributions. These hot wallets will be periodically replenished from offline Reward Pool cold wallets. This distribution will limit exposure to nefarious attempts to drain the Reward Pool.

An Alternative Implementation: PoV as a Smart Contract

PoV is implemented as 2 transactions:

1. Verasity posts a PoV record, a set of challenges and a reward to the blockchain. The challenges contain cryptographic hashes of the correct (expected) responses.
2. The Viewer/VeraPlayer confirms the record and claims the reward by posting a transaction with the challenge responses that match the correct ones. Any node in the system can verify that the responses match the correct ones by calculating and comparing hashes. Any PoV confirmation transaction that doesn't contain a complete set of correct responses is discarded as invalid by Verafiers.

Paid Content (Pay Per View, Donation or Subscription)

Payment for content is implemented as a regular transfer to a channel account with a specially formatted memo to indicate what is being paid for to enable automated processing by the OVP View Authorization Component.

VeraSparks and Channels

A Channel is implemented as a special Graphene asset class. The class can be created by any participant in the ecosystem.

VeraSpark are an instrument that represent future revenue on a Channel. VeraSparks can only be issued by the channel creator. All the VeraSparks are issued on channel creation and deposited into the creator's account. The creator then has the ability to decide how many VeraSparks to offer/sell.

Each channel has an associated channel account to receive payments from viewers. All funds that are transferred to a channel account are automatically distributed to the holders of VeraSparks via a smart contract. If a channel owner doesn't sell any VeraSparks, all funds from the channel account end up in channel owners private account as they are holding 100% of the VeraSparks.

In order to distribute VRA distributions, the system must maintain an up-to-date list of distribution recipients. To keep resource usage in check, the system must take measures to limit the number of VeraSpark holders per channel.

VRA Distributions

Design considerations:

- As a blockchain transaction, VRA distribution is not free. To limit resource consumption, we have to either limit the number of VeraSparks issued per channel to 100-1000 or charge a fixed distribution fee (that can potentially consume the whole VRA distribution value thus encouraging participants to hold sufficient shares of channels to make it worthwhile).

- Several VRA distribution implementations need to be benchmarked to find out the optimal share number limit and/or distribution fee.
- VRA distributions might have to be aggregated somehow in the UI to reduce noise.

TotalSparks

TotalSparks are planned to be an instrument that represent a portion of future net revenue in the Verasity foundation.

The instrument will be implemented as a special Graphene asset class. The class can be issued only by a privileged account that is kept offline and is dedicated to the issuance of TotalSparks. TotalSpark ownership may offer rewards for certain holding periods. Verasity Foundation decides on the maturity date and nominal amount of issued TotalSparks.

Care must be taken to reduce the quantity of TotalSparks issued with different maturity dates and nominal amounts to avoid market fragmentation.

TotalSpark distributions are calculated and disbursed automatically via a smart contract. The distributions are calculated as a fixed percentage of the future net Verasity Foundation revenue. The fee schedule is determined by the Commercial Fee Policy and is managed as detailed in the Secure Policy Management section.

Spark Marketplace

Graphene provides a trustless decentralised asset marketplace that can be used as-is to trade VeraSparks and TotalSparks. The order book is open and transparent. Participants post limit buy and sell orders directly to the blockchain and the orders are matched and fulfilled at the protocol level.

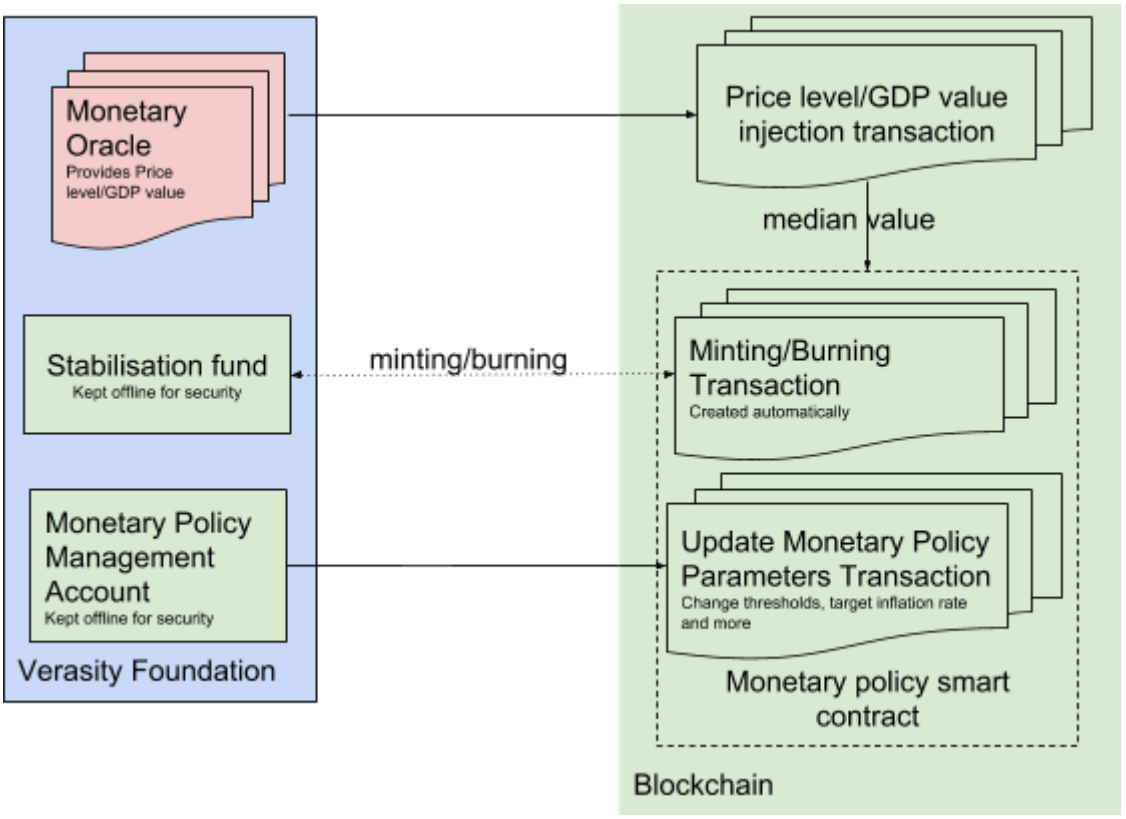
The user interface of the marketplace needs to be augmented with VeraSpark market value data calculated by an algorithm that takes into account PoV records, channel revenue streams and other parameters.

Token Mint / Burn Policy

As explained in the Economic white paper, the Verasity ecosystem manages a strict monetary policy. Mechanisms of managing that policy are primarily instrumented via minting and burning of tokens. Token minting and burning is implemented as a smart contract and happens automatically. Newly minted tokens are deposited into the Stabilisation Reserve. Excess tokens are withdrawn from the Stabilisation Reserve and burned.

Monetary policy parameters are adjusted periodically by a dedicated privileged account that is kept offline. If the monetary policy algorithm requires parameters that change frequently and can’t be calculated using the information available in the blockchain (e.g. the level of nominal expenditures), the required parameters can

be injected using the multiple constrained oracles. This approach is detailed in the Secure Policy Management section.



Secure Policy Management

Multi-Signature Offline Accounts

Multi-signature offline accounts will be used to manage policy parameters that don't change often and/or that require manual intervention. Only a privileged account can push a transaction that changes policy parameters. The account requires multiple signatures. The keys are stored offline. The transaction is signed offline using computing devices not (and if possible, never) connected to the internet.

Hardcoding of Parameter Values

Parameter values are set (hardcoded) in the source code. This approach provides an equal level of security to multi-signature offline accounts. This mechanism will be used for parameters that are planned to have infrequent rate of change.

Multiple Constrained Oracles

An oracle is an automated process that injects data (policy parameters in this case) into the blockchain via transactions.

The difference in this method from the parameter injection using multi-signature offline accounts is that only a single signature is required and the computing device is always connected to the internet. Unfortunately, this

means that the oracle has a large attack surface, and has the possibility of its account keys being stolen, thus making this approach impractical for the management of critical policy parameters and a more secure approach has to be devised. The constructs will be used very sparingly if at all, as they are the highest overhead to manage.

Risk Reduction: Multiple Oracles and Outlier Rejection

In this approach, multiple identical oracles inject the same parameter values and are run by independent entities (employing security through diversity). Parameter values injected by the oracles are averaged in a way that discards outliers (hacked or malfunctioning oracles) such as taking the median value.

Damage Mitigation: Constraints for Values Set via Oracles

The permissible ranges (the upper and lower bounds) of parameter values are managed using the multi-signature offline account approach. The injection transactions that attempt to set parameter values that fall outside of the permissible ranges are discarded as invalid by the blockchain. In this way, even hacking the majority of oracles doesn't allow an attacker to raise a reward (that they are in a position to claim) to a level that would make an attack of this nature attractive or profitable.

Gaining Trust: Oracle Elections and Verification

If oracles calculate parameter values using only publicly-available information, it is possible to hold oracle elections and let participants run an oracle in a verification(read-only) mode.

Software Update Policy

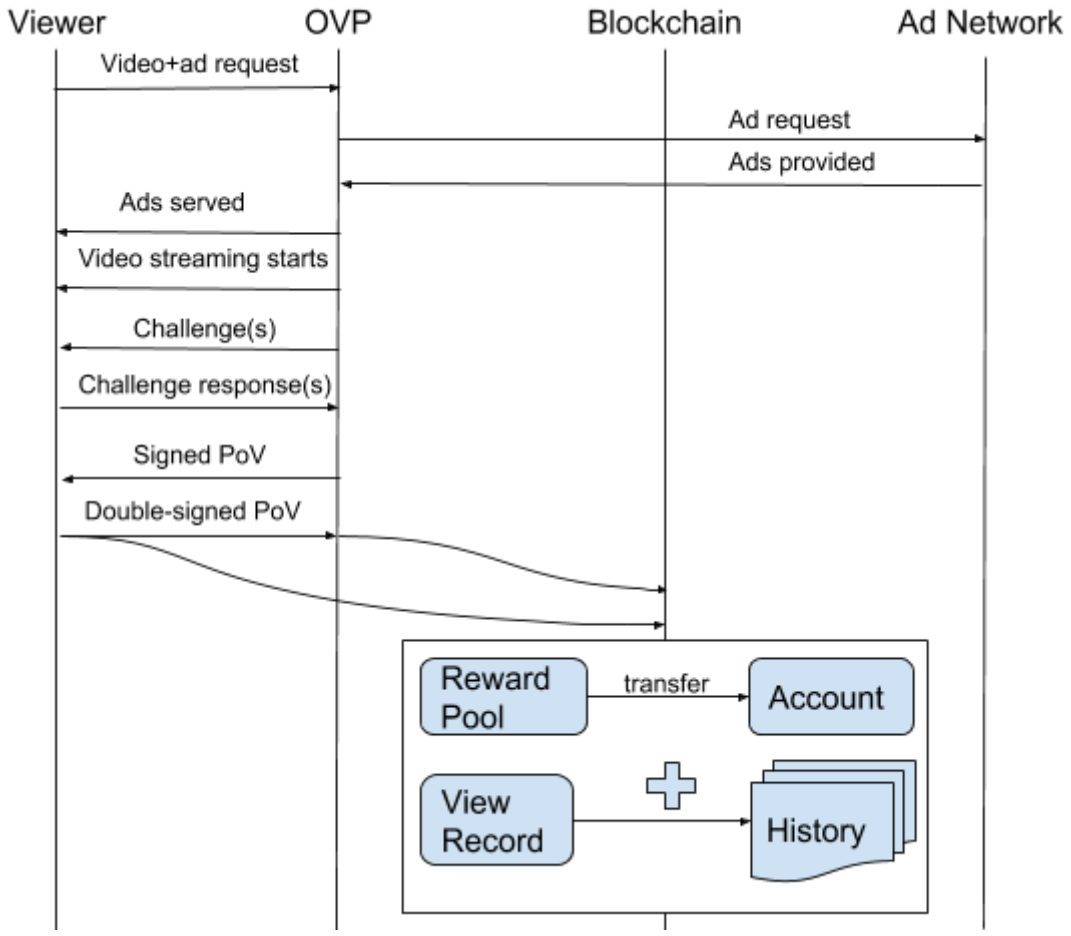
These update policies only affect participants in the ecosystem running Full Nodes. Critical security updates are released and take effect immediately. Verasity will do its best to notify the participants of the need to install a critical update.

Improvements to smart contracts that aren't essential to security are planned to take effect 60 days after the release date. This ensures that all the participants and Verafiers have installed the software versions that support the improvements before they take effect.

Use Cases

Use Case 1:

Viewer watches a video with ads and gets a reward

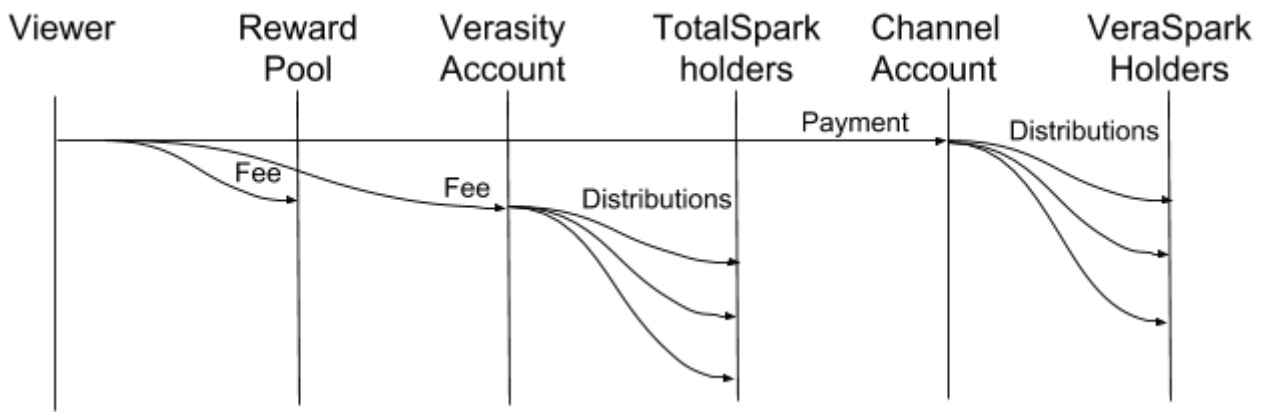


Use Case 2:

- Viewer pays for a video
- Verasity TotalSpark distributions are distributed among TotalSpark holders
- Veraspark distributions are distributed among VeraSpark holders

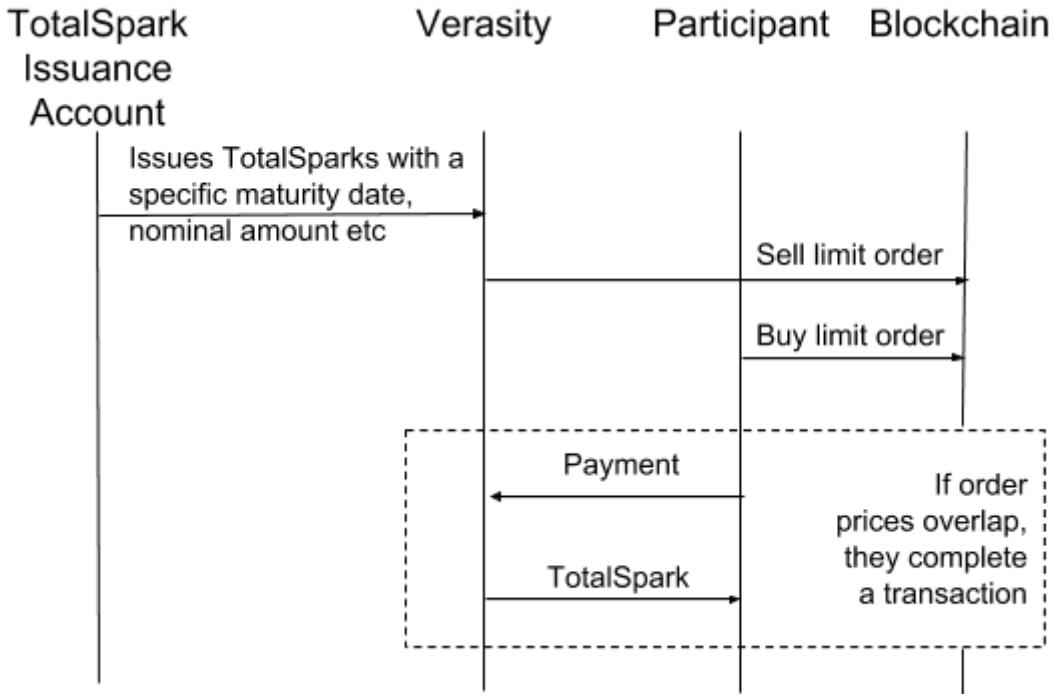
Note:

- The diagram assumes that an immediate distribution model is chosen
- The TotalSpark implementation is subject to adjustments based on the commercial expense structure



Use Case 3:

- Verasity Issues TotalSparks
- Assets (TotalSparks) are traded on the marketplace



Graphene Native Features

One of the benefits from selecting Graphene is that several features can be used as-is. They are not discussed in depth here, as they are well documented already in the Graphene documentation.

Examples include:

- Verafier Rewards (DPoS Rewards)
- DPoS Voting and Verafier selection
- VeraSpark Marketplace (Asset Marketplace)