Galactica Network Reputation Framework Implementation

January 20, 2023

Introduction

The following is the specification of the Galactica Network reputation framework, the backbone of the system that ties together other technological pillars of the network including: contingent transactions, Reputation Root Contract, and zkCertificates, arriving at the most complete implementation of a protocol capable of DeSoc primitives. A careful reader shall note the omission of Guardians and zkKYC: a particular use case of zkCertificates - this is intentional as from the protocol perspective, zkKYC's core purpose is that of bootstrapping a Sybil resistance that makes the concept of reputation (i.e. web3 footprint) meaningful in the first place. In other words, zkKYC and Guardians power the concept of on-chain reputation, while the Reputation Framework described below leverages it to arrive at DeSoc primitives (see the use cases for details).

Reminding ourselves of several definitions pertaining to the Reputation Framework from an earlier Medium article:

1. Following a human in the blockchain space could be referred to as a *Persistent Identity*.
2. The multitude of interactions between any such identity and the rest of the protocol could then be called one's *web3 footprint*.

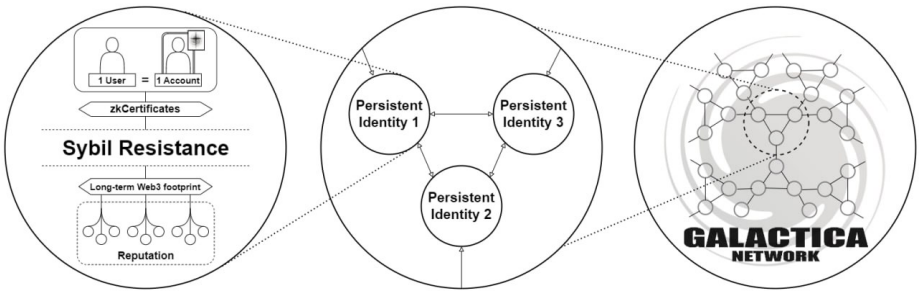


Figure 1: Reputation & web3 footprint

In this context, 'persistent' means that a protocol representation of a human, the private key, corresponds to a real-world person or identity and that such on-chain identities are costly to duplicate and this cost increases over time. But why would the cost increase in time? It is one's web3 footprint that would define this on-chain identity and naturally, this footprint increases in size and complexity as time passes making it harder to replicate or tamper with.

The Persistent Identity is defined by its web3 footprint — the relationships between an account, other accounts, and the system itself. Through one's web3 footprint, users' impact on the network can be quantified by other users, thus defining a virtual correspondence to the real-world concept of *reputation*.

Protocol's Sybil resistance, Persistent Identities, and web3 footprint together enable non-trivial societal institutions to function on-chain — in a similar fashion to how EVM's arbitrary transactional logic enabled the emergence of economic institutions in what's today known as DeFi.

The structure of this piece is different from those we have written in the past. As a reminder, the design of RRC - the Reputation Root Contract (see page 8 of our whitepaper for a reference) is such that any input available on-chain can be used, including in principle those produced off-chain and imported through any oracle. What's described below is the standard implementation that we recommend for any mission-critical application as it inherits the security guarantees of the network itself, something that off-chain oracles clearly do not - and that has been among the core goals we have considered when designing this reputation framework. We, however, also include the description of other possible designs that we have considered along with the justification for favoring the reputation framework we eventually arrived at.

To the best of our knowledge, no one has attempted to build a system of reputation contingent transactions 100% on-chain as there was hardly any point in having such a system in a non-Sybil resistant setup. Hence we view this article in particular, and this implementation in general, to be of an exploratory nature rather than a standard-setting optimal solution. As a careful reader will note, there are trade-offs between security, latency, commercial viability, a burden allocation of deciding which data to collect and how, as well as the consensus on the ultimate standard to be used throughout the network and all within constraints of maintaining ERC20 compatibility.

To understand the context of this specification an understanding of the following Galactica components is helpful:

1. Guardians & zkCertificates
2. zkKYC Design paper
3. Use Cases deck
4. Whitepaper

Target Properties

This document specifies an approach to measuring a user's web3 footprint in the Galactica Network (henceforth, Galactica). It is a central component of the reputation framework because it provides specific data concerning user behavior on Galactica. EVM blockchains have no direct way to access historical transaction data from on-chain smart contracts. This makes it a difficult task to calculate the reputation based on the user's previous behavior.

A successful solution is expressive, efficient, and accurate:

1. Expressive enough to provide the data needed for conceivable and reasonable reputation functions such that they can be flexible enough to be specified by users' reputation functions (e.g. ecosystem contribution, creditworthiness, long-time loyalty);

2. Efficient enough in terms of storage and computation to work in a decentralized and high throughput blockchain;

3. Accurate enough to provide meaningful data that cannot be faked;

4. Requires contingency options integrated into the node to provide a blockchain-wide possibility for contingent token transfers without requiring changes to the ERC20 standard;

5. In the limited info stored setup, the burden of standard creation needs to be decentralized. In other words, different data feeds must be able to exist around one dApp. What to store must not be left entirely up to dApp creators;

The structure of this document first considers the limitations of the current approach. Then we provide a brief overview of the reputation functions and the RRC, which is followed by a review of the approaches to designing the implementation framework for the Galactica Network reputation. Then we discuss the current approach in detail. This article also contains two Appendixes that focus on some of the prior (no longer considered viable) approaches.

Public commentary and criticism are expected and are warmly welcomed.

Reputation function and RRC

Reputation functions are defined by dApps and users. They can either create a custom function based on the available on-chain data or reuse such functions provided by other parties.

Available inputs for reputation functions are a ZKP by the user being rated and on-chain data.

A ZKP can include the following components:

1. Holding a zkKYC or other zkCertificates as well as proving statements based on these, for example being at least 18 years old, living in a specific area, or having a university degree.

2. Alice's humanIDs for dApps relevant to the reputation function. This guarantees that Alice has to include all of her on-chain activity with specific dApps without disclosing which address she used for it.

3. On-chain input data from various sources, such as:

    a. Smart contract state (see Continuous state checking approach)

    b. Rating agency oracles (see rating agency approach)

c. Indexer data passed through the node (see reputation calculation in the node)

4. Merkle proofs for input data (private to not disclose the link between humanIDs)

5. Calculation of final score (calculation private, result public)

The ZKP is sent to the reputation smart contract, verified for ZK validity and consistency of the Merkle roots with chain data.

The **Root reputation contract** (RRC) is a special smart contract on the Galactica blockchain. It works as a registry for reputation requirements of contingent transactions. Every address can register a minimum score and reputation function in the form of an address to the smart contract defining the reputation function. Whenever someone wants to send funds to an address, the protocol looks up the reputation requirements in the RRC to determine if the transaction is contingent. For more information please consider the Reputation Root Contract (RRC) section of the whitepaper.

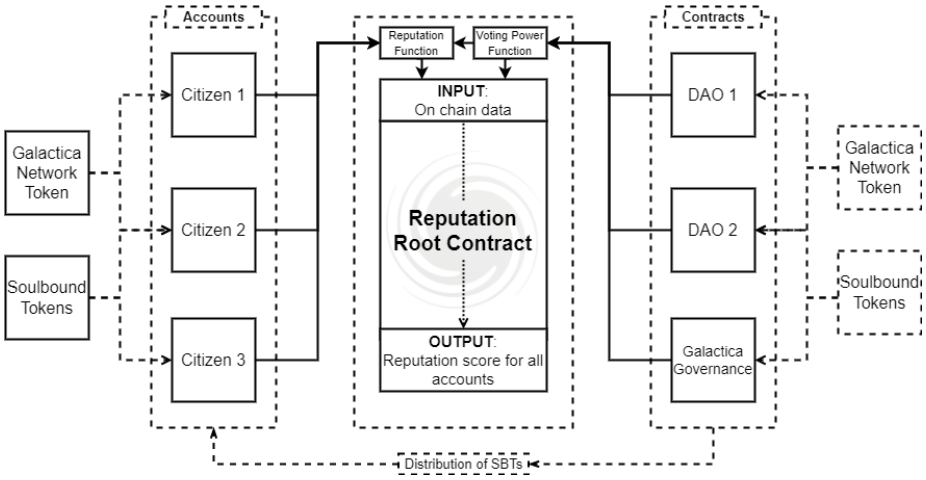


Figure 2: Galactica Network's Reputation Root Contract

Use Case Specification (GalaLend)

In this paragraph we will introduce a use case that will be a reference to explaining the difference between the approaches we have considered, and this use case is included in some of the approaches for further clarity.

1. Alice wants to take a loan in the Galactica Network.

2. Alice likes Galactica because she can prove her KYC without letting others know who she actually is;

3. She visits the front end of a DeFi lending platform;
4. It is called *GalaLend* and checks the reputation of its users to determine the required collateral rate;
5. Undercollateralized loans are only given to trustworthy users;
6. *GalaLend* uses a self-developed reputation function to estimate creditworthiness using the Galactica reputation framework;

Our task is to describe the system that collects and provides the data about Alice's web3 footprint. It is the input for GalaLend's reputation function and should show that Alice has not honored her obligations in the past on the Galactica network.

Approaches Considered

There are various ways to measure a user's web3 footprint and determine their reputation. Ideally, it takes the whole transaction history into account and offers expressive reputation functions while still being efficient enough to be verified by a distributed blockchain. This is a complex task with conflicting goals.

To find a viable solution, multiple approaches were considered. In the following text, some of these approaches are discussed in detail alongside their advantages and disadvantages.

1. Continuous dApp state checks;
2. Off-chain Indexing with third parties (Rating Agencies);
3. Smart contracts accessing transaction history;
4. Current solution: calculating reputation inside a node (validators calculate reputation iteratively while processing blocks);

An exhaustive list can be found in Table 1 - we had to forego part of them right at the outset, as they do not possess the target properties mentioned above.

Approach	Description	Advantages	Disadvantages
Continuous dApp state checks	• Predetermined metrics encoded in dApp logic • dApp SC updates metrics on each tx	• Efficient to calculate • Straightforward to implement • Secure on-chain computation	• Fixed metrics: can not calculate reputation about something not observed • dApps need to implement metric updates to be available • No enforced standardization
Rating score contract	• dApps rate users in special contract • E.g. flags for good/bad behavior and counter for interactions • Querying score given by a set of dApps • Use Merkle tree to hide link between addresses in ZKP	• Technically feasible • Secure on-chain computation	• dApps need to implement rating • Rating criteria defined by dApps giving the rating, not by the dApps requiring the rating • Handling users with multiple addresses requires ZKP to hide connection between dApp specific human IDs
Rating agencies	• Rating agencies analyzing tx history off chain • Providing their rating on-chain as oracle • dApps relying on reputation can aggregate ratings from different agencies • Similar to indexer oracle	• Creates business model for rating agency • Outsources rating to the market fostering innovation of rating DAOs • Allows expressive reputation functions over whole tx history	• Depends on availability of humanID to cross reference addresses • Intransparent rating agencies have a bad reputation on tradFi.
Indexer oracle	• Using off-chain tx history storage in indexer • Running reputation calculations off-chain on indexer data • Publishing results through oracle on chain	• Flexible for all kinds of functions on tx history • Data already available on indexer	• Computationally expensive (crunching through whole tx history, large database, complex functions) • Requires trust mechanism for oracle accuracy (too complex to be provable through ZK) • Indexer to large to be run by average user
On-chain tx history in patricia roots	• Protocol stores patricia roots of every block and makes it available in EVM • Generating ZKP about history off-chain with indexer data • Verifying ZKP on-chain in smart contract	• Secure on-chain validation • Neglectable on-chain storage requirements	• Complex ZKP over patricia trees and tx history • Proof generation requires indexer • Technical feasibility with SNARKs unclear
dApp specific on-chain tx history	• dApp saves history relevant for it on-chain • dApp history available on-chain through ABI	• Secure on-chain computation • Simple to implement	• Large on-chain storage requirement • Needs to be developed and provided by dApps
General on-chain tx history	• Protocol level history db • Storing subset of tx interactions (Signer, Counterpar Funds moved) • Special non-EVM query functions	• Universal solution not requiring effort by each dApp develope	• Storage space vs. saved details • EVM compatible? • Maybe not enough details for some use cases • High computational effort on-chain

Table 1: Galactica Network's Reputation Root Contract

Continuous dApp state checks

DApps that want to contribute web3 footprint data of their users do so using continuous state checks.

The continuous state assigns each dApp-specific humanID some data. This can be data the smart contract uses anyway for its business logic, a user rating, or some other measurement.

The current state is always available on-chain and is continuously updated whenever a call processed by the smart contract influences the state.

This state is made available to other smart contracts through an ABI function according to the GIP-42 Standard defined below. To preserve the privacy of the user (which humanIDs he/she has across different services), the smart

contract also provides the root of a Merkle tree holding the data as well as event logs to construct Merkle proofs.

Advantages

1. Condenses the complexity of the transaction history into an aggregated state for each user of a dApp;
2. Lowers the complexity of the input data for the reputation function;
3. State often already available as part of the dApp logic;
4. Secure on-chain computation;

Disadvantages

1. Requires developer labor/man-hours to follow the standard;
2. Dependencies between contracts could lead to exploits;
3. Hard to adjust or introduce metrics retroactively to her reputation score. It takes the relevant dApps, her zkKYC;
4. Each dApp can decide what data to track leading to a heterogenous quality of the base data for reputation calculation;

GalaLend Example

1. Each lending dApp Alice used in the past keeps track of a continuous state for her dApp-specific humanID. For example:
    a. Lending dApp 1 saves her current outstanding loan, the amount of collateral, and if any of her positions have been liquidated before.
    b. Lending dApp 2 saves the current outstanding loan, her next due date for repayment, and how often she missed a repayment date.
    c. Lending dApp 3 manages Alice's state in a private manner and only exposes a trust score between 0 and 255.
2. Whenever Alice interacts with one of her lending dApps, the smart contract updates her continuous state accordingly.
3. When Alice needs to use her reputation for a 'creditworthiness' score-like function, she presents a ZKP for her reputation score. It takes the relevant dApps, her zkKYC, humanIDs, and Merkle proofs for her state in each dApp as input and outputs her reputation score as well as the Merkle roots for on-chain verification. The ZKP proves the correctness of the score without revealing any humanID or concrete values of her state.

Rating agencies

Rating agencies use off-chain indexers to compute complex functions over the full transaction history of users. The results are committed on-chain through an oracle smart contract.

Rating agencies could be incentivized by on-chain fees implemented in the oracle contract, off-chain by services relying on them, or through protocol-level tokenomics.

Advantages

1. Off-chain computation allows complex reputation functions with much input data from the indexer;
2. Rating results can retroactively be changed to implement new models or sanction lists on historical transaction data;
3. Creates a business model for rating agencies;
4. Outsources rating to the market fostering innovation of rating DAOs etc;

Disadvantages

1. Depends on the availability of humanID to cross-reference addresses in a later step;
2. Lack of transparency on the side of rating agencies having earned them an unfavorable reputation in TradFi;
3. Off-chain input can be manipulated with consequences for on-chain dApps;
4. "Chicken and Egg" problem between rating agencies and projects requiring reputation input data; If there is no rating agency providing the data a project needs, the project is likely to implement the off-chain indexer check itself and not share it with other projects on-chain;
5. Rating agencies may present a centralization force;
6. Rating results are delayed on-chain due to the potential computation effort and oracle submission time. This leaves room for malicious activity before the rating is updated, for example through flash loans;

GalaLend Example

1. Alice uses lending dApps on Gala. Some of them query her dApp-specific humanID in the smart contract to uniquely identify her. (The humanID is useful in this approach to allow computing a user's reputation when his activity is split over multiple addresses.);
2. A rating agency runs an indexer to track Alice's on-chain activity;

3. The data from the indexer and on-chain state is aggregated by the rating agency into a rating result. This can be a score, such as 'AAA', a detailed statistic about her lending history, or a blacklist containing addresses and humanIDs that are not creditworthy;

4. The reputation result is published on-chain through an Oracle contract. It stores data in a Merkle root and collects a fee for the agency when being called from a smart contract;

5. When Alice wants to present her reputation, she calculates it on-chain. The input is a ZKP that takes Alices zkKYC, her humanIDs for specific dApps, and the rating agency's results for these humanIDs. The ZKP proves Alice's reputation score without revealing her concrete addresses, humanIDs or intermediate ratings;

In Appendix I one can find further information on the implementation issues related to this approach.

On-chain access to TX history

Multiple approaches were considered here, however, the main issue that persists is that data produced by an address (e.g. transactions) is challenging to be analyzed on-chain. The transactional history is not available to smart contacts to reduce the complexity and keep the working memory manageable. Blockchain analytics companies utilize state-of-the-art AI approaches that are computationally heavy in order to convert the data into useful information. We believe that at this point there is no solution for efficiently analyzing transaction history data in on-chain smart contracts alone. The base data would require nodes that are as heavy to operate as indexers and EVM smart contracts computation is too costly to work with transaction data as it needs to be recomputed a lot in a distributed blockchain.

Using zkProofs on TX history

ZKPs can be utilized to resolve the complexity of analyzing transaction data from a smart contract. In general, if the transaction history is stored as a Merkle tree on-chain (fairly easy to do), the smart contract can request the user to generate a ZKP on a given condition about their past activity. For example, it may request them to prove that all their loans are closed within a certain time frame and that they have never received funds from non-KYCed accounts.

While this approach may appear viable, the following issues persist:

1. The condition that reliably confirms the aforementioned (or similar) requirements will be relatively large, given that it has to check many interactions with various protocols.

2. Furthermore, calculating a ZKP on such a condition requires significant computation time - it may take days on the average user's device.

Simplified version of this approach

We have attempted to devise a framework that will allow us to store on-chain events in a simple format (rather than transaction history) - that is, pre-analyzed. The implementation is described in Appendix II, and while relatively simple we believe that it will be avoided by the community as it requires every dApp developer to store the usage data of their dApp in a special smart contract.

Advantages

1. Data available for expressive reputation functions;
2. Secure on-chain computation;
3. Possibility to recompute modified reputation functions on tx history;

Disadvantages

1. Heavy burden on node operation because computation is repeated for decentralized verification;
2. Complex computation either in smart contract or in ZKP generation leading to high gas costs or long proving times;

GalaLend Example

1. Alice uses a lending platform and during tx processing, a record of this tx processing is made available on-chain
    a. Either directly in memory accessible by smart contracts;
    b. Or as patricia root of each new block;
    c. Or in a custom record of the lending smart contract itself;
2. When Alice wants to present her 'creditworthiness' reputation to a new lending dApp, a smart contract computes the reputation using
    a. The on-chain database of her previous transactions with other lending dApps;
    b. A ZKP disclosing her reputation based on previous transactions with lending dApps. The ZKP was computed off-chain by Alice. The correctness is verified on-chain using saved Patricia roots of previous blocks.
    c. Customized ABI functions of other lending dApps returning Alice's previous actions;
3. Based on the computed reputation score, Alice is permitted to use the new lending dApp or gets a customized offer;

Subchains Approach

While it appears challenging to implement, as it requires node development and also every validator will have to store more than just blocks, this approach is decentralized, trustless, and easy to use for everyone. As it will become clear from the information provided further in this document, this approach furthermore addresses all of the limitations described in the previous two solutions considered.

Cosmos Subchains

Subchain is an independent network that is connected to the mainnet. The connection usually includes a bridge and technology that allows the exchange of information between the subchain and the mainnet. In this case, the IBC protocol will be utilized, allowing synchronization between the states of EVM-compatible networks. Subchains are often used to reduce the load on a mainnet by delegating specific features to them. They receive inputs from the main network, carry out computations and send the results back.

It was concluded that it is beneficial to use subchains for reputation metrics calculations. The subchain reads data from the mainnet blocks through RPC, validates block hashes, calculates reputation, then submits the calculation results back to the mainnet. The first subchain will be launched by Galactica and used to calculate the reputation metric that the governance framework will utilize to determine users' voting power.

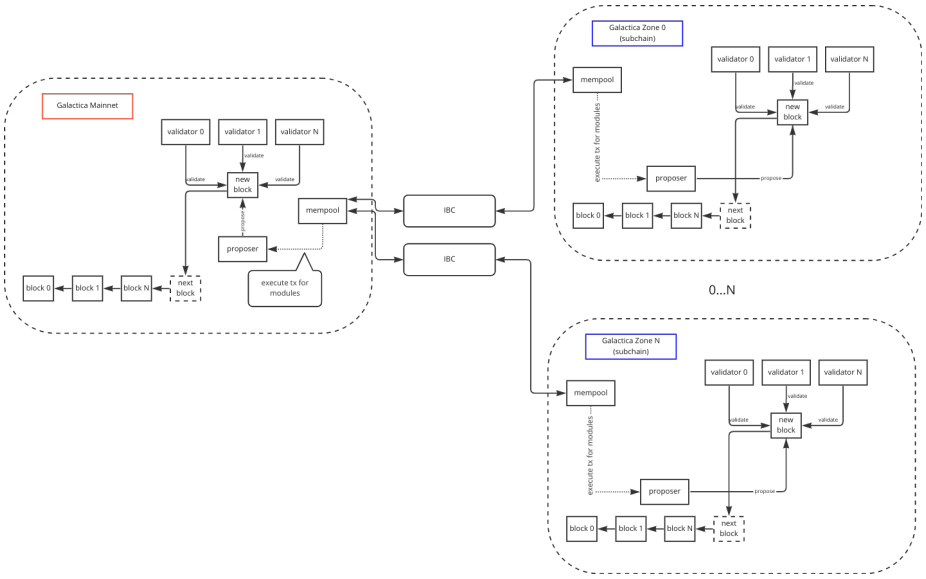


Figure 3: Galactica Mainnet and the Galactica Zones/Subchains

Reputation Module

The *Reputation module* is a typical Cosmos SDK and is embedded at the core of Galactica Network. The on-chain reputation calculation method utilized here allows the use of the logic present in a smart contract based on the Reputation value. This logic can be used to reach a consensus among Validators based on block production results, in addition to amending Reputation values.

Each Validator calculates Reputation values independently and only validates Merkle root hashes from the data they used for calculations.

The module operates in two synchronization modes:

1. *Sync (Historical)* - a state in which the Reputation has not yet been calculated from historical data and the Reputation is actively indexing. In this state, the Reputation value cannot be used in other smart contracts;
2. *Live (On-head)* - Reputation is calculated for the head block and it is possible to use the Reputation in other smart contracts and modules.

Stages of Historical Data Synchronization

During historical data indexing, a Validator performs the following actions:

1. Reads the subsequent block;
2. Validates blockchain hashes;
3. For each transaction:
    a. Decodes transactions and their events;
    b. Carries out the formation of source data;
    c. Carries out data aggregation by event type or another group;
    d. Calculates Reputation based on aggregations calculated in the past;
    e. Saves the result of the Reputation calculation in a smart contract.

Validators store source and aggregated data off-chain; however, they keep the Merkle root hashes of that data on-chain. To add new functions which build the source data and data aggregations, special EVM smart contracts (repositories) are provided. It makes it possible to add functions that expand the functionality of the Reputation module.

Utilizing multiple intermediate steps in the indexing pipeline makes storing data off-chain possible, allowing for lower on-chain storage. The smart contract that considers the reputation value is public - anyone can observe what the reputation consists of.

Live Blocks Processing

Live blocks processing utilizes the same algorithm as historical processing to calculate reputation metrics in every new block. It implements EVM hooks *PostTxProcessing* to subscribe for incoming transactions.

Contingent Transactions

When the live processing module receives a transaction, a check is carried out to verify its compliance with the established reputational requirements. This includes two mechanisms:

1. An address on the Galactica Network can be set up with a threshold of a minimum reputation value needed by senders to be able to send funds to this address. In case the reputation of a sender is insufficient, the transaction will fail. This will work with native asset transfers as well as with erc20, erc721, and erc1155 token standards.
2. A smart contract may set up reputation requirements for every public mutable method. The contract needs to have a view method, containing a dictionary of:
    a. Method id;
    b. Reputation metric id;
    c. Condition;

If the executor does not meet the listed reputation requirement(s) - the transaction will fail.

Calculation Process

The implementation of a subchain (*blockchain node*) includes a special Indexer specifically for on-chain events. This Indexer can collect, store, and aggregate specific events described in a *root on-chain contract*.

For example, it can store and aggregate trades from multiple DEX protocols on the mainnet. The description for such logic would resemble the following:
Step 1:

- List of contract addresses of these DEXs
- List of trade methods from these DEXs ABIs (a list for every DEX)
- List of fields which should be stored in Indexer

Step 2:

- Using the resulting entries and a formula compute a specific value (e.g. total volume traded, number of trades, total fees paid).

The Reputation value, which is a result of an aggregation, will be re-calculated every block by all subchain Validators following given rules.

Using the Indexer, we can calculate metrics for every address, at every block. What remains is to actually calculate the Reputation. The formula (or formulas)

for calculating it are located within the *root smart contract*, and Validators of a subchain will calculate Reputation based on values obtained from the Indexer, then push those values to the mainnet using IBC.

The role of subchains in this process is to avoid the overload of Indexers and only store on-chain what is required for critical calculations.

Delays

There is a delay between events that happen on the mainnet and those subsequently delivered to a subchain. This delay may be up to N blocks in duration (subject to further testing to estimate the exact delay), meaning that even in a subchain the Reputation is delayed, and thus, unsuitable for time-sensitive operations such as flash loans.

This delay may lead to a state where certain subchains may be utilized solely by specific protocol subcategories - e.g. a dedicated subchain for lending protocols. This specialization of subchains will allow them to track the interactions of a given user among themselves without delay.

For example, provide undercollateralized loans to those users who have a lot of positive activity on the mainnet (delayed info) and never failed to pay for a loan (instant data, located on a subchain). Such a hybrid Reputation metric may help lending protocols merge a user's overall score from the mainnet, with a trustless lending rating that lives on the said subchain.

GalaLend Example

1. Alice uses lending dApps on the Galactica Network. Some of them query her dApp-specific humanID in the smart contract to identify her uniquely. (The humanID is useful in this approach as it allows the computation of a user's reputation when their activity is split over multiple addresses.);

2. The lending dApps and other Galactica services usage history is aggregated and stored by the Galactica validators. Some of them are maintaining subchains and others - the mainnet. They use this aggregated data to calculate the reputation of every humanID in line with various formulas. The result of calculation - the reputation itself stored on every change in a special smart contract in the form of a Merkle tree.

3. When Alice wants to present her reputation, she does it on-chain. The input is a ZKP that takes Alice's zkKYC, her humanIDs for specific dApps, and the reputation results for these humanIDs from the smart contract mentioned in point 2. The ZKP proves Alice's reputation score without revealing her concrete address(es), humanIDs, or intermediate ratings;

Subchain economics

This will be discussed in a separate article specifically dedicated to subchain economics.

Further considerations

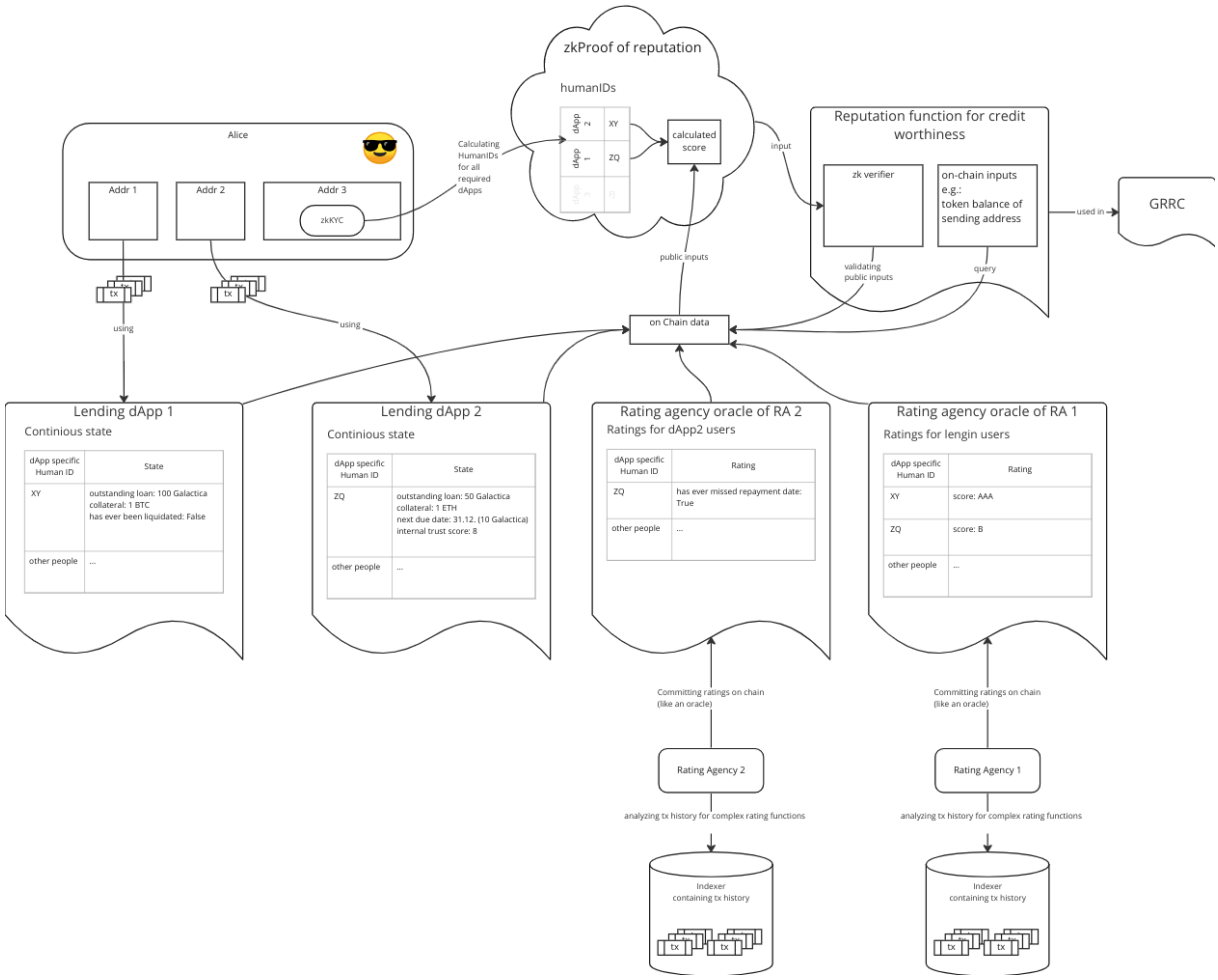
1. To use on-chain data as verification for ZKPs without revealing the data in public proof inputs (so that users do not have to expose all of their humanIDs at once), it is required to store it in a Merkle tree.;
    a. Adds complexity to smart contracts;
    b. Leads to concurrency issues (root being changed before a proof is verified). If we allow a history of roots, proofs might be validated against old states that were exploited recently;
2. Versioning web3 footprint inputs might be difficult;
3. A lot of complexity originates from having multiple addresses and dApp-specific humanIDs per user;
4. Timing issue. How long is a ZKP valid? How can conflicts be detected?

Appendix I: Off-chain Indexing with third parties in detail

Scenarios

Lending Use Case

The use case is the same as described earlier in the Use Case Specification (GalaLend) section. The following diagram gives an overview of the components involved in this example. Because there is a lot going on, we go through it step by step.



Alice uses multiple addresses on Galactica to prevent others from profiling her identity on-chain from the *selective disclosures* she has made in the past. With *Addr1* and *Addr2* she already used lending services on Galactica. They are different lending services to CreditGala, but still relevant for CreditGala's reputation score for creditworthiness. With *Addr1*, Alice used a service called *GAave*, and with *Addr2* the lending service *Gompound*. She proved to both services that she is a real person using her zkKYC held by *Addr3*.

Continuous State Checks

Alice has an open loan on GAave, but her past business with the platform is flawless. The dApp tracks Alice's state including the outstanding loan balance. Because the dApp does not know Alice's name, it uses her dApp-specific

humanID, which is XY. This ID uniquely identifies Alice on GAave. *So no matter which address or zkKYC Alice uses, the dApp knows that it is dealing with the human having the ID XY.* The state of the dApp is updated on every transaction, such as paying back a portion of the loan or someone executing loan liquidation. This is done anyway by the dApp to implement the lending business logic. Furthermore, the dApp keeps track of a boolean flag if Alice has ever defaulted because it charges previously defaulted users a higher fee. This flag acts as a **continuous state check** for Alice's web3 footprint. It is a statement about previous activity that is updated with every transaction. GAave chose to track this state. But each dApp is free concerning what and how to track it. So it could also have chosen to track the total volume of loans paid back.

On the Compound dApp, Alice has the dApp-specific humanID ZQ. Here Alice defaulted on a loan in the past and the dApp tracks this with an internal trust score. It is also continuously updated according to Alice's interaction with the dApp.

Because both GAave and Gompound want to be interoperable and contribute to the Galactica ecosystem, they implement Gala's web3 footprint standard for continuous state checks, called *GIP-42*. According to the standard, dApps provide their internal continuous state through an ABI function as data for reputation calculation. The data is indexed by the user's dApp-specific humanID, and represented by a Merkle tree.

CreditGala also involves other dApps, such as Lending3, in the reputation function. But Alice did not use the Lending3 dApp. Therefore she has no humanID entry there.

Rating Agencies

The continuous state of GAave and Gompound is already a good basis for CreditGala's reputation function. In terms of efficiency and accuracy, it is good because it is computed in smart contracts alongside the usual lending business logic. However, the expressiveness could be improved because it only covers fixed statements specified by the developers of GAave and Gompound. CreditGala wants to complement it with a more detailed analysis of Alice's transaction history. They want to add a measure of loan repayment punctuality that is not covered by continuous state checks. This measure can be computed with an indexer by going through all of Alice's lending platform transactions (identified by XY and ZQ). Because it is computationally heavy, this is done off-chain. CreditGala outsourced the task to a rating agency called Rating Agency 2. They act as an oracle [1] and commit the results of the analysis for each humanID [2] on the blockchain. The rating agency gets paid by CreditGala

[1] They can alter it, I guess we can have a network of reputation calculators for the Alicee standards and the onchain data aggregator contract only receives the data with enough signatures. This is one of the reasons for investigating the possibility of a Cosmos sidechain specifically for reputation.

[2] They only see addresses and dApp specific human IDs. So rating agencies can provide

directly.

Additionally, CreditGala includes the credit score of the Rating Agency 1 in their reputation function for additional robustness [3].

Reputation Proof

CreditGala published their reputation function as a smart contract on Galactica. It is used by the Galactica Root Reputation Contract (every dApp or user can define their own reputation function and register implementation address in this smart contract), and computes a score in the range [0, 1] based on the on-chain inputs of the continuous state from the lending dApps and the rating agency oracles.

To preserve privacy, users prove their reputation score in a zero-knowledge proof (ZKP). The ZKP includes the following components:

1. Human ID for CreditGala based on Alice's zkKYC (public input);
2. Alice's human IDs for all dApps relevant to the reputation function (GAave, Gompound, Lending3) → XY, ZQ, FJ (stay private in the ZKP);
3. Score function input data for those human IDs (never defaulted on GAave, trust score 8 on Gompound, missed repayment according to Rating Agency 1, scores AAA and B according to Rating Agency 2, private in ZKP);
4. Merkle proofs for input data (private to not disclose the link between humanIDs);
5. Calculation of final score (calculation private, result public);

The ZKP is sent to the reputation smart contract, verified for ZK validity and consistency of the Merkle roots with chain data.

Result

The only creditworthiness score Alice can provide with a valid ZKP is poor. Therefore the smart contracts of CreditGala can reject Alice's request for an undercollateralized loan automatically without an infringement of Alice's privacy. Based on the score, the CreditGala smart contract offers Alice a loan with a 200% collateral rate instead.

---

data for XY and ZQ. The ZKP requires Alice to put those together into a final reputation score.

[3] For CreditGala this has some advantages:

- Redundancy so that their reputation function still works if one of the rating agencies is unreliable
- Combining different aspects of credit worthiness

Architecture

Interfaces

This concept of input data for the web3 footprint is intertwined with many other components of the Galactica Network. It uses the following interfaces:

zkKYC

- zkKYC is used to generate the dApp-specific humanIDs that index data about a user on Galactica.
- A ZKP about the zkKYC is needed to force the user to include all her humanIDs in the reputation computation. The list of dApps relevant to the reputation needs to be provided by the dApp requiring the reputation check.

GRRC

- To utilize the reputation based on web3 footprint, its calculation can be requested from the Galactica Root Reputation Contract.

Wallet

- To privately prove a reputation score, the user creates the ZKP on her machine [4].

Components

Out of the various options on how to measure a user's web3 footprint, we chose a combination of continuous state checks and rating agencies to provide both simple and expressive options.

Continuous state reputation

DApps that want to contribute web3 footprint data of their users do so using continuous state checks.

The continuous state assigns each dApp-specific humanID some data. This can be data the smart contract uses anyway for its business logic, a user rating, or some other measurement.

The current state is always available on-chain and is continuously updated whenever a call processed by the smart contract influences the state.

This state is made available to other smart contracts through an ABI function according to the GIP-42 Standard defined below. To preserve the privacy of the user (which human IDs he/she has across different services), the smart contract also provides the root of a Merkle tree holding the data as well as event logs to construct Merkle proofs.

[4] We should soon have first test results about a ZKP for KYC + > 18 proof.

Advantages

1. Simple to implement
2. Often already available
3. Secure on-chain computation

Disadvantages

1. Requires work by developers to follow the standard
2. Dependencies between contracts could lead to exploits
3. Hard to adjust or introduce metrics retroactively

Rating agencies

Rating agencies use off-chain indexers to compute complex functions over the full transaction history of users. The results are committed on-chain through an oracle smart contract.

Rating agencies could be incentivized by on-chain fees implemented in the oracle contract, off-chain by services relying on them, or through protocol-level tokenomics.

Advantages

1. Creates a business model for rating agencies
2. Outsources rating to the market fostering innovation of rating DAOs etc.

Disadvantages

1. Depends on the availability of humanID to cross-reference addresses
2. Intransparent rating agencies have a bad reputation on tradFi.
3. Off-chain input can be manipulated

Reputation function

Reputation functions are defined by dApps and users. They can either create a custom function based on the available on-chain data (continuous state, rating agencies, balances, etc.) or reuse general reputation functions provided by other parties.

Available inputs for reputation functions are a ZKP by the user being rated and on-chain data.

ZKP includes the following components:

1. Alice's human IDs for all dApps relevant to the reputation function (GAave, Gompound, Lending3) → XY, ZQ, FJ (stay private in the ZKP)
2. Score function input data for those human IDs (never defaulted on GAave, trust score 8 on Gompound, missed repayment according to Rating Agency 2, scores AAA and B according to Rating Agency 1, private in ZKP)
3. Merkle proofs for input data (private to not disclose the link between humanIDs)
4. Calculation of final score (calculation private, result public)

The ZKP is sent to the reputation smart contract, verified for ZK validity and consistency of the Merkle roots with chain data.

Standards

GIP-42: Web3 footprint provider standard

This standard defines how dApps provide data about the web3 footprint of users. It can be represented in a Galactica Improvement Proposal. It applies to both dApps implementing continuous state checks and rating agencies' oracles.

Projects are incentivized to use this standard for interoperability and contribution to the Galactica reputation system.

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Methods

Smart contracts implementing this standard must provide the following ABI functions:

- footprintMerkleRoot
  - Returns the latest Merkle root of the Merkle tree holding the continuous state of users.
  - Each leaf in the tree should be the Poseidon hash of the user's human ID and continuous state.
  - Merkle tree must use the poseidon hash
  - Each human ID must have at most one leaf in the Merkle tree
- footprintMerkleLevels
  - Returns the amount of levels of the Merkle tree
- footprintMerkleRootHistory(root)

- Returns true if the given root is in the last X relevant merkle trees
- This avoids concurrency issues of users proving stuff and other users changing the merkle root

Events

- Event log of modified leaves for merkle tree data availability

Appendix II On-chain access to TX history in detail

Introduction

A contract storing the fingerprint of every address is considered. The implementation of this solution is described in brief here.

Inputs

Any dApp may contribute to the contract by posting traces to it. Trace is an object containing information about an interaction between a user and some other party. A trace contains the following fields:

1. Party (dApp) unique name (e.g. Uniswap, MakerDAO);
2. Flag. Enum:
    a. *Red* - the interaction has led to unpaid loans or any other consequences that would decrease the "Financial Trust Score" of a user. Any malicious actions will be given a red flag;
    b. *White* - any ordinary interaction without the reputation change (e.g. trades, liquidity provision, stakes). Basically, white flags are just counters which track a user's activity;
    c. *Green* - paid loans or any other "good" interactions in terms of "Financial Trust Score";
3. Comment: text field for a comment;

Outputs

1. Any dApp (or other smart contracts including the reputation contract) may read traces of a given address and make some decisions on top of it.
2. When a smart contract reads traces, it may use a "mask" - a list of addresses or unique dApp names. Only interactions with listed parties will be permitted.

3. Later on, this mask may be obtained from some public dApp registry contract, which will measure dApp reputation based on their activity.

4. dApps may grant users red or green flags at their discretion. While the system is decentralized and anyone can check the dApp code and find out the situations in which it grants users with every specific flag, the reader may recognize every role with its own amendments. Some parties may be excluded, for some parties, red flags may be ignored, etc.