Towards Defining Web3 Identity Stack

March 18, 2024

@sarvodaya_gala

Contents

Part I: Defining The Web3 Identity Stack

Preamble

Reputation is the core use case of the Web3 Identity Stack.

As the notion of persistence of Web3 Reputation suggests, there is a fundamental difference between the scope of Web2 and Web3 use-case design spaces around the concept of Identity. The Reputation - a notion that defines Web3 Identity comes at the forefront in most of these use cases. In Web3 Reputation is persistent, user-defined, composable and open-source. These properties will transform the broader Identity Stack just like DeFi has revolutionized financial institutions and the primitives comprising them.

Web3 Reputation and its myriad of use cases are at the forefront of the Identity Revolution that public blockchains enable.

Introduction

In this article, we aim to:

1. Elaborate on the notion of Web3 Identity Stack, and
2. Develop a framework laying down the desired properties of such a stack with respect to Web3 Reputation functions.

Both notions are quite elusive and will take some effort to unpack. We will begin Chapter I by clarifying (and updating) the vocabulary emerging around the notion of Web3 Identity Stack. Next, we will touch upon the relationships between these concepts and their web2 predecessors. We will then proceed to define what persistent reputation is and how and why it instills new meaning into the notion of online identities disrupting the very foundation of the web2 data industry.

It is essential to offer the world a flexible and holistic toolbox for working with Web3 Reputation to operationalize the disruptive potential it holds. To do so, we need to give deep consideration to the mathematical, technological, economic, and user experience-related properties of Web3 Reputation functions.

In the two years since we began working on Galactica.com, we have come all the way from drafting a concept to arriving at a viable prototype solution for on-chain reputation and this article as well as those to follow is an attempt at sharing what we have learned along the way. But it's more than just that. At the heart of our motivation to draft this series is a desire to seek feedback, align the definitions, and spark public dispute as we believe that a responsible approach to designing permissionless reputation systems is paramount to the internet's evolution no less.

In Part 2, we attempt to define what a well-behaved reputation function could be from a mathematical viewpoint. We will touch upon the philosophical foundation and desired product properties of such functions. We will then proceed to the formal derivation of what a general purpose function would look like.

In Part 3 we address the technological implementation of a framework capable of instilling practical meaning to all these notions.

We hope that those who spend time delving into the entirety of these articles will find them thought-provoking.

Before proceeding further, we recommend revisiting some of the older papers we have published in the past, including Protocol Citizenship and The Cypher State for some of the most fundamental definitions of what relates to reputation and web3 identities and how they form a reciprocal relationship between each other; Galactica's Reputation Framework Design for some of the earlier drafts of what a reputation framework could look like mathematically; Galactica's Reputation Framework Implementation for a deep dive into the technological implementation of a viable reputation framework; Galactica's Governance Framework Paper for a deep dive into Galactica.com's governance framework for which Reputation is the beating heart, and finally, Galactica's for an actionable roadmap towards the ultimate vision of governance framework.

Chapter I: Context for Web3 Identity Stack

Let's start with two quotes that each address a part of what the Web3 Identity Stack can be defined as:

1. *"Web3 is based on the premise that each internet user will have a unique internet identifier, like an email address, that can be natively linked to any piece of software and stored on a blockchain. As part of someone's "decentralized identity", a portion of a person's online activity would then be "on chain", meaning that it would be public and easily searchable via their individual crypto wallet.*

   With such a decentralized identity — a readable history unique to each person — one's crypto wallet would function as a sort of profile, similar to Facebook or LinkedIn. But unlike web2 profiles, decentralized identities are backed by hard evidence: a permanent, timestamped record of a person's accomplishments, contributions, interests, and activities to date

   If decentralized identity were widely adopted, people would be able to carry their full selves with them as they traverse cyberspace: their affinities and experiences reflected by what they've created contributed to, earned, and owned online, no matter the specific platform. This would bring us closer to how things work in the physical world, where our possessions and reputations are attached to us, rather than to the spaces we occupy; we can take them with us and use them however we please." - Scott Duke Kominers and Jad Esber for a16zcrypto

2. *"Reputation systems present an opportunity for platforms to recognize—and thus incentivize—participants' high-quality contributions, including content creation, moderation, community building, and gameplay. This is*

crucial to the growth and sustainability of any web3 project. Yet designing reputation systems requires complex considerations around reputation supply, distribution, credibility, and more." - Jad Esber and Scott Duke Kominers for a16zcrypto

These two quotes give us a good idea of both, the potential that decentralized identities and web3 footprint enable and the challenges that one has to face to arrive at a viable technological solution for them.

The problem space spans from definitions of on-chain reputation and entities comprising the conceptual plane surrounding this notion, to defining desired properties of the functional form of reputation functions, such that the output is well behaved across all corner cases and is attainable computationally given limitations of today's blockchain systems. In other words, one has to invent a new narrow vocabulary and a standalone mathematical and economic framework to solve for a viable on-chain reputation.

What are the relationships between Identity, Identification, Web Footprints, Reputation, and Web Paradigms (Web2 vs Web3)? As defined below, all these concepts lie semantically close to each other, and understanding them goes a long way toward solving some of the internet's biggest long-standing challenges.

The Definitions

Let's revisit (and update) the definitions:

1. **Identification:** The core primitives that collectively (and individually) identify a real-world person - embodying characteristics such as demographic information and skill sets, consumer preferences, geolocations, and entertainment interests. Identification necessarily originates off-chain and can then be on-ramped to a blockchain protocol. It's instructive to think of identification as most of the data that fuels Google's business model.

2. **Persistent Identities:** A Persistent identity consists of all the individual's attainable data points originating off/on and cross-chain that exist (originated or on-ramped) on a blockchain protocol. A sufficiently generalized definition of a human from the perspective of a blockchain protocol is the data that a real-world human has generated on the protocol itself or has otherwise on-ramped. A human on a blockchain can be referred to as a Persistent Identity. Today's most popular blockchain protocols are not sufficiently equipped with features to reliably associate all data points generated by a given human to a single Persistent Identity.

3. **Web3 Footprint:** A Persistent Identity is defined by its Web3 Footprint - it is basically another word for all the data points that a human has generated or on-ramped.

4. **Social Graph:** The multitude of interactions between any such persistent identity and the rest of the protocol could then be called one's Social

Graph. In other words, a subset of one's Web3 Footprint scoping all the interactions with other protocol entities is one's Social Graph. Social Graph does not include the Identification. It is instructive to think about it this way: *Identification + Social Graph of a Persistent Identity = Web3 Footprint.*[1]

5. **Web3 Reputation:** Finally, let us define what we mean by Web3 Reputation. The definition is quite formal and at first, might appear counterintuitive. Web3 Reputation is an arbitrary function of the Web3 Footprint of a Persistent Identity evaluated over an arbitrary subset of the data points comprising it. For example, one's Web3 Footprint can consist of his or her zkKYC, a set of NFT collections one owns, one's activity across all dApps comprising the wider ecosystem, and many other things. An example of Web3 Reputation then could be something like:

   *SQRT(age of account) * [0 if defaulted on a lending protocol in the past, 1 otherwise] * [1 if has KYC, 0 otherwise].*

   Of course, it can also be any other function that can be computed given the availability of data and technological properties of the protocol.

6. Web3 Identity Stack: This encompasses the suite of technological primitives underpinning the concepts outlined above, enabling practical use of identities and reputations within the Web3 ecosystem.

Before we proceed to define desirable properties of on-chain reputation, let us digress a little and give some valuable historical context that would enable us to frame these esoteric concepts in more conventional web2 terms.

Chapter 2: The Web2 Identity Stack

"Shoshana Zuboff in The Age of Surveillance Capitalism, refers to user interactions on platforms like Google as behavioral value surplus. Historically, a firm had limited resources that it had to employ immediately to produce the goods it sold you. Or it paid ridiculous amounts in storage fees. A pencil manufacturer had to ship pencils. Ford's car factories had to sell cars. They could not endlessly stockpile the timber or rubber for the process.

With the advent of the internet, however, this equation changed. A player like Google or Meta could keep your data for a decade until it could be monetised for their benefit. I could go to Facebook now and download all the cringe texts I may have sent my crush back in 2011 (and so could you).

[1]Note that there is a broader definition of a Social Graph that spans beyond the Web3 domain. Social Graph as defined by Decentralized.co "Put simply, it is the network of people and their relationships on an emergent platform. A social graph can be formed outside a digital native platform, like a university. Or it can be powered through algorithmic discovery like on Twitter. A graph can be public, where you can see who interacts with whom." For the purpose of this article, think of the 'emergent platform' as a given blockchain network, such as, for example, Galactica.com. To requote, a social graph is the network of people and their relationships on a given blockchain platform.

In the early 2000s, most dot-com projects were what AI websites are today: an abundance of inbound traffic with little or no business model. You could license your search engine to a larger corporation or sell sponsored ads like Yahoo did. [...] So Google had to find a different way of selling ads altogether.

Instead of allowing people to bid on and list ads based on their assumptions about what the audience would click on, the data scientists at Google could measure and predict which ad would best suit which person. Instead of a brand's ad manager working on assumptions, you had data scientists targeting users, allowing the brand to see a clear RoI on each click from Google."

In other words:

1. Data is sticky, hence our conviction that Data is the new TVL;
2. As we all know in retrospect, a consequence of this stickiness is the incredible marginality of Google's business model - it is without a doubt the most successful business model ever created;
3. At the heart of it are the digital (i.e. web) footprints of real people.

Let's move on.

"...The perfect storm was in place for the evolution of the web. A business was realizing the possibility of generating and storing a resource (user data) with next-to-no marginal costs and had the pipelines (targeted ads) to monetise it. All that was missing, in the style of most venture-backed companies, was a mechanism to scale it. This is where social graphs came into play.

...[Then came Facebook.]...

More users effectively meant you had a critical mass that could be divided and sold all kinds of goods. Users belonging to similar social graphs could be bucketed and served similar content. This became the basis for the algorithmic feeds in which we currently find love, jobs, giggles, despair & hope.

In Web2 networks, the social graph is the moat. If you allow users to interact with a graph via third-party applications, your chance of capturing user data diminishes. After all, the user then won't be on a product that you control. If users can simply port their network of friends and family to a different application, they will have no incentive to return to yours, either.

We don't have a shared protocol for social applications that work at the scale of Meta or Twitter because of how incentives are structured for existing behemoths. A Web2 product with open social graphs opens itself to competition and declining revenue. Both of which may not be a desirable outcome. [...]"

Summarizing the extract above, the web2 frontier as we know it today is largely a product of business model-driven ecosystem evolution along two key axes:

1. Social graphs as in X, Facebook, etc., and

2. Other individual data footprints or *identifications*, such as search history, location, other metadata, etc.

Following the logic of definitions we have introduced above, these two combined would make up Web2 Footprint, the fuel of the data economy.

Now, once we are on the same page regarding social graphs and the broader Web2 Identity Stack, let's try to unpack the evolution of the concept of reputation.

Quoting a16z:

"In contrast to most web2 profiles, decentralized identity is not ephemeral. This means that an NFT of a diploma in your crypto wallet, for instance, would turn into a permanent academic certification. Likewise, each piece of content you post online would be permanently linked to you (unless you choose to delete it). Moreover, with public histories it would be possible to prove that you were early to a trend or active in a project before it took off — like, say, being into Taylor Swift before she was popular or reading this article while web3 is still in its genesis.

This persistence establishes new incentives for reputation-building: instead of creating temporary profiles like we do in web2, web3 promotes long-term thinking. If people were empowered to build and maintain permanent identities online, we believe that on-chain systems could encourage people to more carefully curate the reputation markers that they carry with them into the future. In this case, curating a permanent library of NFTs is higher stakes than, say, curating a series of social media posts, in that it's a reputation marker that you carry with you across cyberspace."

As one might observe, the concept of 'persistence' is quite ubiquitous when we try to unpack the Web3 Identity Stack. We would even go as far as to say that the added property of persistence of the web footprint in the Web3 space effectively turns any data points associated with an 'identity' into reputation. It is a nice observation, but is there more to it? Can we outline a formal framework for the desired properties of Web3 Reputation systems?

In what follows we will attempt to do just that.

We would like to point out that what follows should be treated as an open-ended discussion: the Galactica.com team welcomes thoughtful feedback and rigorous disputes around a topic as important as Web3 Reputation.

Chapter 3: General Desired Properties of Web3 Reputation

As we have outlined above, the Web3 Reputation is a function over data points a Persistent Identity has generated. For example, Alice's idea of reputation might be a performance someone has shown when paying back one's debts to a lending dApp. Alice's idea could then be to use such a reputation when deciding on a collateral rate she likes to set on a new loan to this person. Bob's definition of reputation on the other hand can be some derivative of Expert

Conference POAPs and Education SBTs that would together determine a voting power someone should have in his expert DAO. It is therefore fair to say that Reputation Functions are ubiquitous in Web3 systems, and are use-case driven.

Now, it's important to tackle the question of how a system adaptable enough to accommodate such diverse use cases might be structured.The following is a non-exclusive list of properties that in our view a Web3 Reputation must possess in order to be production/adoption viable. Some of these properties are purely technological in nature, others have to do with required properties of system economics, yet others concern something else altogether.

Here they are:

1. **Private and verifiable:** nascent properties of public blockchains and the problem space around negative reputation dictate that any holistic stack for Web3 Reputation absolutely needs to possess two core properties: (a) a reasonably equipped outside observer cannot infer the reputation of a single user by observing the blockchain, and (b) The reputation calculation is correct, complete and verifiable. As we shall see below, albeit trivial at the outset, this property poses very significant technological challenges. Only a combination of ZKP (Zero-Knowledge Proofs) and FHE (Fully Homomorphic Encryption) can reasonably solve this challenge.

2. **Origin-agnostic:** Data points comprising one's Web3 Footprint (and by implication, the reputation) can originate from cross-chain, on-chain, and off-chain sources and can be reliably on-ramped into the on-chain reputation design space. In other words, in order to create the most holistic view of one's Reputation, one has to allow for real world (e.g. Education), web2 (e.g. X or Discord), and web3 (e.g. Ethereum, Solana activity) data points to represent a person on-chain.

3. **Real-time:** Real-time feed of reputation data is essential for many critical on-chain use cases. The framework resistance to reputation flash attacks where an adversary uses transaction bundles similar to flash loans to take out undercollateralized loans is of the essence. Real-time in this case shall mean 'same block'. Same shall apply to cross-chain and off-chain data sources, albeit arguably the best one can hope for here is t-1 block.

4. **Time-series and cross-section:** Reputational functions need to be sufficiently general-purpose to incorporate two dimensions: arbitrary temporal and cross-sectional logic. In econometrics or statistics this would mean 'panel data' view of the Persistent Identity activity on a blockchain. The cross-sectional dimension is what we mean elsewhere by Heterogeneous Accounts: every Persistent Identity can have a unique set of data points associated with it at every observable point in time (i.e. every block).

5. **Expressive functional form:** In a perfect world, the domain of Reputation functions shall include any elementary function, subject to a constraint of computational feasibility. If one is to examine the functions used

by financial institutions to evaluate the quality of a loan portfolio, it becomes clear that they can be immensely complex. The domain of use cases of Web3 Reputation is far greater than what TradFi financial institutions do, hence it is to be preferred that attainable Reputation Functions are as diverse as possible. As in many other cases, the major impediment to realizing a perfect world use case is the available compute and speed of data dissemination in a global decentralized network. The next property deals with defining 'computationally feasible' in this context.

6. **Computational feasibility:** Arguably, the only solution to defining which reputation function is computationally feasible is to create a Market for Reputation Compute. In other words, similar to how Gas addresses the issue of competition for block space, the demand and supply must mediate the inclusion of any new Web3 Reputation Function.

7. **User-defined:** There needs to exist a democratic access to a market for Reputation Functions mentioned in point 6 above in order for it to be efficient. No monopoly, no undue barriers to entry.

8. **Composable:** Any user-defined Reputation Function needs to be available for use by any other user of the network. Any component of such Reputation Function needs to be open-source to be examinable by other network participants. This property is essential for forming an efficient demand side of the market for Reputation Functions elaborated upon above.

The points above serve as our best attempt to lay down a vision for a perfect Reputation dimension of the Web3 Identity Stack. In our view, this article is instrumental to creating a theoretical foundation for those to follow.

In Part II of this series we will delve deeper into the desired mathematical properties of reputation functions and how they relate to the attainable economic and game theoretical properties of reputation systems they enable.

Part III will concern the technological implementation of a system that would possess all the properties we will have discussed. It will be the first semi-formal specification of Galactica.com Reputation Framework.

Part II - Defining Desired Mathematical Properties of a Reputation Function

Introduction

The concept of reputation function plays a pivotal role in assessing immutable merit. Persistent Identities generate Web3 Footprint - the totality of data points generated by a real-world person on-chain. Simply put, Reputation functions are slices of one's Web3 footprint evaluated for a particular purpose be that one's credit rating, academic achievement, applicability to be a member of a given DAO, one's reputation as a founder, etc. Although there are some limits to the number of use cases of reputation, there is an infinite number of possible reputation functions.

Reputation scores are variables that a reputation function consists of. Age of account, number of X posts, time-weighted liquidity provided to a dApp - all these are examples of reputation scores. Technically, a reputation function aims to aggregate specific reputation scores, condensing factors such as importance, value, and difficulty, into a single metric.

So what are the desired mathematical properties of Reputation Functions, such that these functions can both reward and incentivize as broad a spectrum of economic activities as possible? In what follows we will perform a more formal derivation of a generalized reputation function. Before that, however, we would like to summarize our considerations when designing it.

Henceforth, we will be using the concept of Galacitca.com meritocratic UBI distribution when giving examples of a process to be optimized using Reputation Functions. The core idea is simple - a tokenized diversified portfolio of assets is distributed across all Citizens of Galactica.com to both incentivize and reward positive contributions to the network. An important consideration for anyone designing incentive mechanisms should be the system's propensity to a concentration of resources one tries to distribute, be it money, power, status, or some other sort of social capital. In the end, we, the people of the web3 space are no strangers to ill-designed incentive and distribution systems. Therefore, to ensure fairness and prevent monopolization by early adopters, certain considerations must be addressed. These considerations include but are not limited to:

1. **Bound reputation:** firstly, it's crucial that specific reputations, as well as the total reputation, have upper bounds. This prevents disproportionately large reputation scores from skewing the system, especially in the early distribution stages. In other words, there must be some sort of decreasing returns to work performed. In this case, there will eventually be an asymptote towards which the Reputation converges.

2. **Dynamic reputation:** additionally, distinguishing between active participation and inactivity is essential. While an additive point system is intuitive, inactive users' scores should slowly dilute away in favor of active participants.

3. **Time decay within specific reputation functions:** this penalizes inactivity among early adopters while allowing any users parity through sustained effort and skill.

4. **Democratic reputation:** creating a total reputation function presents its own set of challenges, primarily due to the diverse needs of the community. While the technological stack of modern blockchains enables the creation of reputation functions based on statistical and elementary functions, determining the most appropriate total reputation function requires community input and democratic decision-making.

5. **Predictability of incentives:** furthermore, the total reputation function must be predictable, ensuring that users understand which behaviors are most rewarded.

6. **Static vs Dynamic Pay-offs:** it's important to differentiate between one-off and continuous pay-off functions, with one-off rewards being fixed and continuous rewards decaying over time.

Ultimately, the total reputation comprises both one-off and continuous pay-offs, with the latter subject to time decay. For example, considering the Galactica.com UBI distribution, by incentivizing users (with UBI) to align with the network goals (reputation scores), the total reputation function serves as a mechanism for preserving meritocracy. However, it's crucial to recognize that incentives may evolve, requiring flexibility in defining the total reputation function as an increase per block rather than in absolute terms. This approach ensures that users who best align with the *current* incentives can earn a reputation at an optimal rate based on their *ongoing contributions* to the network.

Context and Definitions

First off, let us introduce some definitions:

1. **Reputation type:** Total reputation is a function of work done. Different types of work are collected in different types of reputations. This definition is only of practical importance but does not have an input to the model itself.

2. **Action/Task:** The only way that one can quantify work that has been done is through verifiable proofs of work within predetermined conditions that are tied to predetermined rewards.

3. **Table:** A list of all actions/tasks within a reputation type

4. **Weight:** A number that specifies the sensitivity of a global reputation function to incremental units of work done. Each reputation type has a weight within total reputation.

How can one translate work done to a simple number? There is no way to consistently incentivize people other than a framework mapping actions into rewards. Coincidentally, mathematically it is the same as a rating or a reputation system - the only difference being that reputation works *ex-post*, while incentive system - *ex-ante*. Actions that users can perform must be listed in advance and they must have specified rewards after completion.

Let us consider an average user within any type of incentivized system - e.g. an airdrop campaign. Our user can distribute one's time to perform work and achieve certain tasks within a given timeframe. Said work can be distributed towards finishing different tasks thus advancing different reputation types - e.g.

liking a tweet made by the project running the campaign - part of X's (formerly Twitter) reputation.

Energy/time/work the user spends can be proxied by the number of tasks accomplished within a given timeframe. Furthermore, when using such a proxy, the user's competitiveness in said work is also taken into account since higher skill yields more results. In the end, people who invest their time working, are going to gain more points, and people who are proficient in their work are going to be rewarded even more, since they can finish a higher volume of quality work within the same period.

Thus, it is fair to say that reputation points are a proxy value for the amount of qualitative work performed.

If the tables (a list of all actions/tasks within a reputation type) are created in an ideal fashion (so that points ideally correspond to the value of qualitative work - units of work normalized for time and value) and the user doing the work is ideal the functions of time distribution in real life and point distribution in the space of reputation types are the same. Since time distribution is a plane on which all times add to a constant (approximately, since the problem is discrete rather than continuous), mathematically speaking this surface is a plane that is angled at 45 degrees to each of the axes - the iso-work plane, see Figure 1.0.

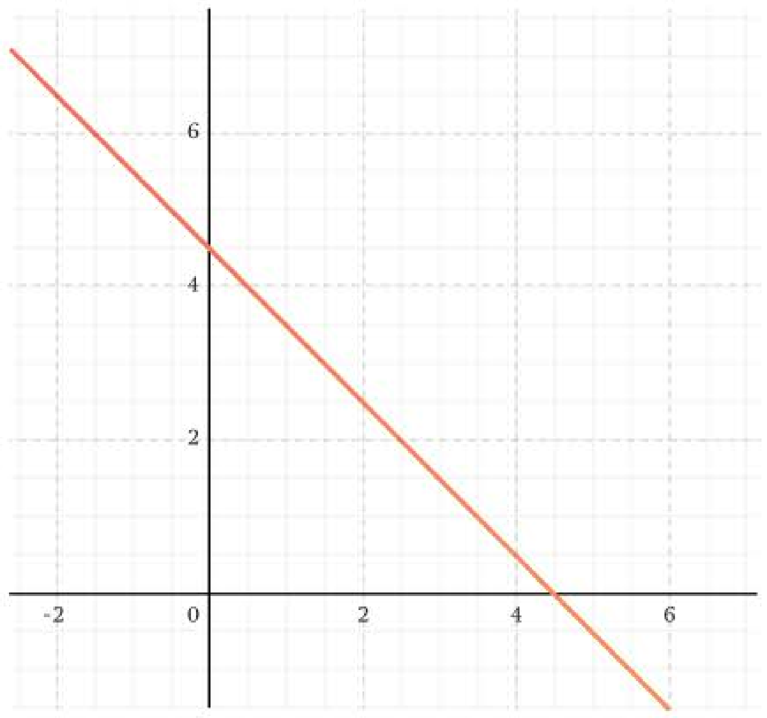


Figure 1: Iso-work plane for 2-dimensional space

It is necessary to add a note that real users are far from ideal ones. Their distribution, in reality, will be different from the ideal case since no one is proficient in all different kinds of tasks, thus their distribution of points is always going to be below the ideal case.

Reputation functions can be thought of as incentive systems - systems that incentivize a particular behavior. The generally desired property is that the highest rewards are earned by those who are aligned with the protocol's incentives.

Rewards are represented with weights and they should incentivise people to invest their time in the desired ratios accross the activites underlying reputation types. Those who comply with this rule will be rewarded the most. Imputed work is additive thus the function should reward the most those people who have their work distributed in proportion to the desired weights.

A Formal Model of Reputation

Let us introduce some mathematical grounds that will be used later. If there are N distinct types of reputation (tables that convert user's work to points) every single user is represented by an N-dimensional array. Space of all possible reputation configurations is an N-dimensional space (X^N) and a single user is represented as a point in this space. Reputation is a function in this space that takes points from N-dimensional space and outputs a positive real number, i.e.:

$$R(x_1, x_2, x_3, ..., x_N) : X^N \longrightarrow \mathbb{R}^+$$

Generalized reputation functions can output a negative number if the function is bound from below. In this event, a simple linear transformation can always transform it into a positive case. If it does not have a lower bound then some users might not be incentivised to be further invested in the system. Therefore, it is best to make the reputation bounded by some lower limit and in this document, the reputation will have a lower bound of **1** with respect to all reputation types. No meaning other than its mathematical practicality is given to this number.

It has been mentioned that a reputation system is both a reward and an incentive system. Now let us look at what that means. Looking at the community as a whole (all subjects to a given reputation function), the collection of points that represents all users that had imputed the same amount of work W is represented by the function:

$$W = x_1 + x_2 + ... + x_N$$

This can be illustrated in the case of 3 variables:

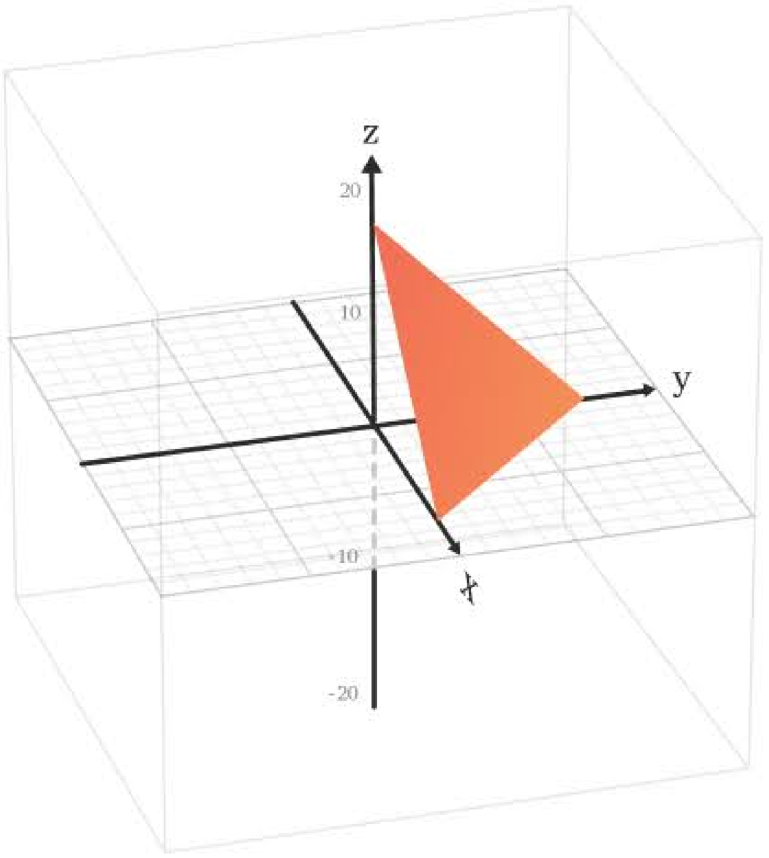


Figure 2: Iso-work plane in N=3 dimensions

If the function is predictable, for every W, there is a point that has the largest reward. Because the work is additive, the simplest incentive function is the line because it requires people to keep fixed ratios of work as they accumulate reputation points. Incentive lines defined in this way, have fixed incentives independent of how much work has been done by the users.

Let us introduce an incentive line on this graph:

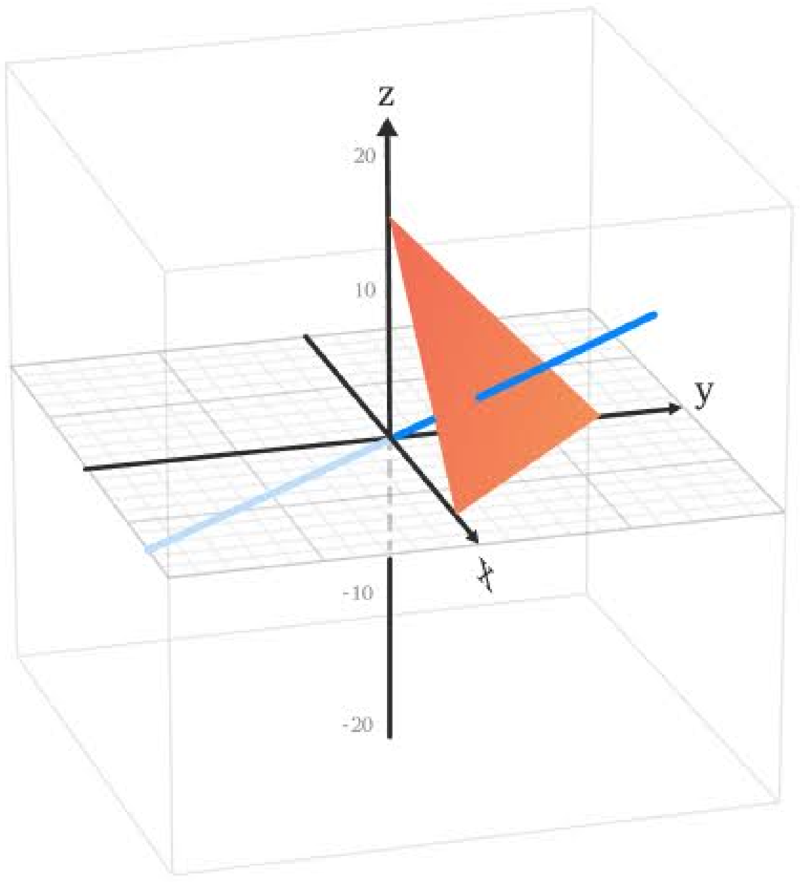


Figure 3: Iso-work domain (red) and the incentive line (blue)

The reputation function must reach a maximum on the presented domain exactly where the incentive line intersects it. A collection of points that have the same Reputation value are represented by the surface in this N-dimensional space. For the reputation to be maximal at the incentive line on a given domain the equi-Reputation surfaces must increase with value as one moves further from the origin which is guaranteed by the fact that Reputation should be monotonically increasing with the addition of more work.

Furthermore, equi-Reputation surfaces should be tangent to the domain W and touch it at only one point - the intersection point with the incentive line. This can only be possible if the reputation function is convex since if it was concave it would be also inside the pyramid the equi-work surface makes with the coordinate axis' and by definition would not have a unique incentive that does not belong to the coordinate planes. If it is convex one guarantees not only that reputation is maximal at the intersection point but it is maximal at that point for all possible amounts of imputed work that is lower than W.

To finalize the discussion let us list all properties the reputation function should have. It should be a continuous function that is monotonically increasing with the additional work imputed in any type of reputation. It should be convex and tangent to the equi-work line W and it should reach a maximum at the intersection point.

A method that we employ to prove that the conditions are met is Lagrange's multiplier. A function should reach a maximum given a constraint:

$$g = x_1 + x_2 + ... + x^N - W$$

One function that is continuous, differentiable, and has modular behavior is the Balancer DEX invariant. It is of the form:

$$R = x_1^{\alpha_1} x_2^{\alpha_2} ... x_N^{\alpha_N}$$

Let us prove that this function satisfies all necessary properties. The Lagrange multiplier method gives:

$$grad\, R = \lambda * grad\, g$$

Which gives a set of N equations:

$$\frac{\partial}{\partial x_i} x_1^{\alpha_1} x_2^{\alpha 2} ... x_N^{\alpha_N} = \lambda * \frac{\partial}{\partial x_i} g = \lambda \quad i \in [1, N]$$

Which after solving for λ gives a set of relations on variables and parameters:

$$\frac{x_i}{\alpha_i} = \frac{x_j}{\alpha_j} \quad \forall i, j \in [1, N]$$

The incentive line condition can be found as:

$$\frac{x_1}{\alpha_1} = \frac{x_2}{\alpha_2} = ... = \frac{x_i}{\alpha_i} = ... = \frac{x_N}{\alpha_N}$$

It is best to illustrate in 2D and 3D cases:

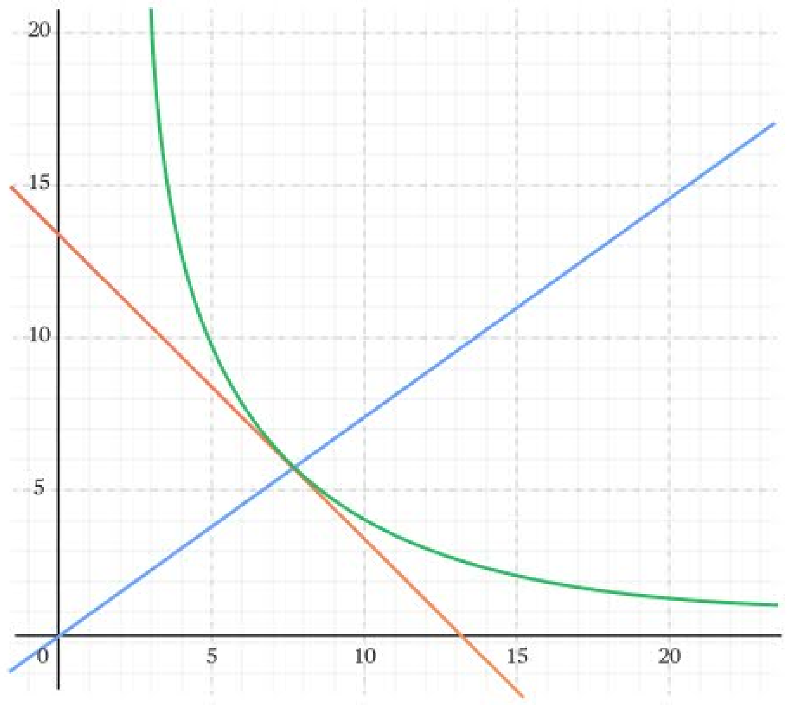


Figure 4: Maximization constraint with the iso-reputation curve (green) for N=2

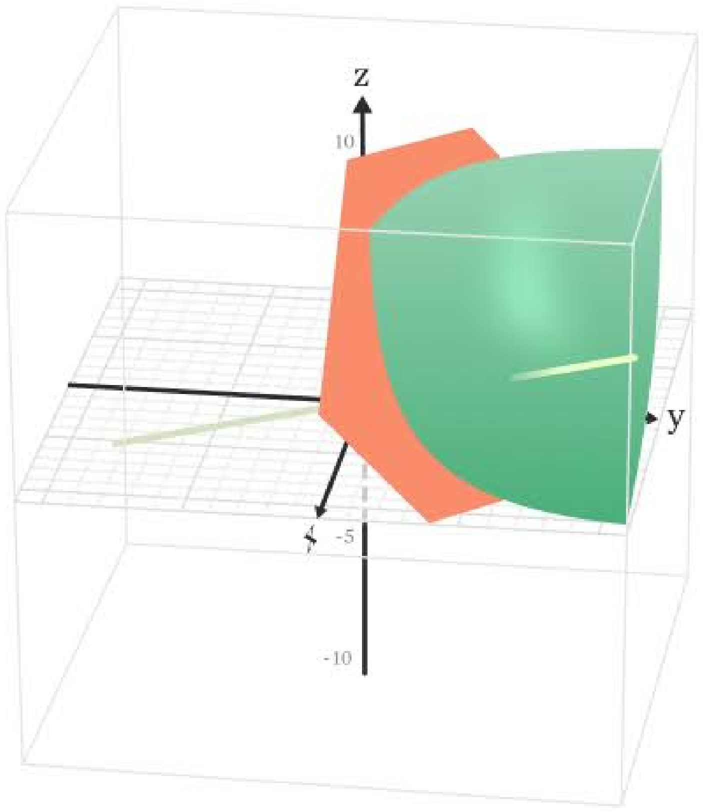


Figure 5: Maximization constraint with the iso-reputation curve (green) for N=3

The intersection between the domain W and some reputation function will always be a convex loop, for N=3 it can be visualized and it looks like this:

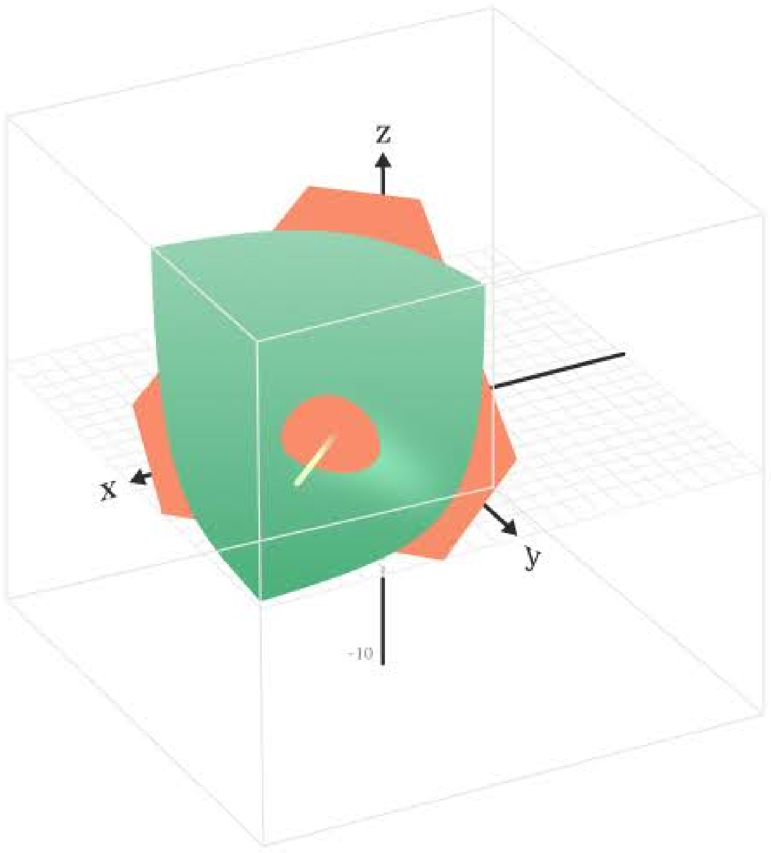


Figure 6: Convex loop at the intersection

The intersection of green and red surfaces, as displayed in Figure 6, determines all people that had imputed total work W and have the same reputation. As one increases the value of equi-Reputation, said loop will become smaller and smaller until eventually converging to the intersection point for the maximal Reputation users can have. It is now evident why the minimum of Reputation is set to **1**. This is a mathematical practicality that implies that the specific reputations are independent and if a user has a minimal reputation of one type that does not impact his overall score negatively.

Addressing Temporal Dynamics

Let us now address the time dimension of a hypothetical reputation function. Should the reputation function have an upper bound? If it does not, after sufficient time some active users will accumulate so much reputation that new users will have little chance to catch up - think about it as a common condition of maximal attainable hero level in MMORPGs. Had it not been bound, most of the games would have fallen out of favor after several years or would at a minimum resulted in severe inter-server fragmentation of active gamers.

This essentially is a problem of all merit-based systems, if **qualitative work** is additive then merit must be additive also. But if work is additive then early adopters benefit the most (which could be a desired property in some systems, but we shall concern ourselves with a more generalized case).

The only way to have a non-diverging meritocratic system is if we assume that the system has always been there - a condition that is impossible since we wish to create such a system. There is no general solution to this problem, but there may be a workaround. When designing the reputation function, we need to enable an option of diminishing returns, thus making reputation logarithmic in work input ensuring an easier entry to new users. Initially, a dynamical bound will exist and after sufficient time this constraint can be lifted by the community (e.g. using a 1P1V voting system).

Mathematics can get complex if one wishes to do this directly. A simpler approach can be to incorporate a decay directly in the reputation function. For example, consider how one reputation type R depends functionally on the amount of points X:

$$R_i = x_i^{\alpha_i} \quad for\ \alpha_i \in (0,1)$$

To visualize it:

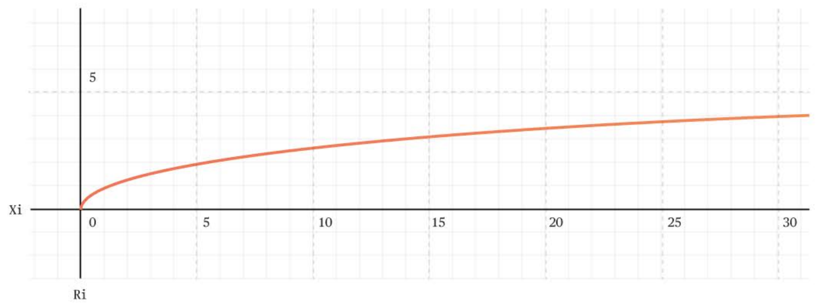


Figure 7: R_i dependence on x_i

The function does indeed possess diminishing returns, a fact one can see by taking a derivative of it:

$$\frac{\partial}{\partial x_i} R_i = \alpha_i x_i^{\alpha_i - 1} = \frac{\alpha_i}{x_i^{1-\alpha_i}}$$

This shows that as x_i increases $\frac{\partial}{\partial x_i} R_i$ continuously decreases thus for any Δ (maximal work any given user can input in any given time frame, for example a month) and every $\delta > 0$ there exists x_i such that $R_i(x_i + \delta) - R_i(x_i) < \delta$. This simple observation solves the problem of requiring an upper bound to reputation functions. If in a month a user loses more reputation to the decay than δ, the reputation will effectively have an upper bound. With this in mind let's define time decay.

Derivation via the Lagrange multiplier method must not be compromised when one incorporates time decay. Work variables need to be disconnected from time-dependent variables. When looking at the reputation function as a whole it is evident that the decay needs to act multiplicatively on all specific reputation functions respectfully. Mathematically, the specific reputation function with decay would look like this:

$$R_i(x_i, t) = x_i^{\alpha_i} * f_i(t)$$

As decay acts on the function of work it is evident that the decay will have the same impact for all different x_i thus it will result in a % decrease in the specific reputation. Since $f(t) \in (f_{min}, 1]$ and $f(t)$ is monotonic and decreasing, inactive users lose reputation weight over time. As the decay is relative (i.e. percentage based), the people with more reputation will lose more in absolute terms thus effectively decreasing the wedge between new and old users.

The simplest form would be a linear function:

$$R_i(x_i, t) = MAX\{1, x_i^{\alpha_i}(1 - k_i(t - t_{LatestAddition}))\}$$

The maximum function essentially does not impact the dynamics of the system but rather simply creates a lower bound that users can not cross, a box within the X_N space of specific reputations greater or equal to 1.

Let us now consider the problem that arises as a result of the functional form of decay. One must recognize that the total reputation of any type is essentially just a product of two functions, one that depends on qualitative work and the other that is only time-dependent. As arguably they are independent, the time component will always tend to reduce the amount of reputation linearly.

As discussed earlier, people will deploy their productive time into work but their returns will diminish with every incremental unit of work done. This implies that at some point people will move so far along the curve that no matter what they do they will not be able to increase their reputation further or even keep it from declining. After sufficient time all people would find themselves with their reputation equal to 1. This corollary creates evident challenges when considered from the vantage point of system incentives.

This problem can not be solved directly but rather with an algorithm:

1. Let us say that a user holds X amount of total work and that as of this moment, their reputation of type i is:

$$R_i(x_i, t) = X^{\alpha_i}$$

2. As time progresses a user loses reputation linearly:

$$R_i(x_i, t) = x_i^{\alpha_i}(1 - k_i(t - t_{LatestAddition}))$$

3. On the next increment of work, a user has:

$$R_i(x_i, t) = X_i(1 - k_i(t_{CurrentAddition} - t_{LatestAddition}))$$

4. Effective work would then equal to:

$$X_{eff.} = \{R_i(x_i, t)\}^{\frac{1}{\alpha_i}} = \{X^{\alpha_i}(1 - k_i(t_{CurrentAddition} - t_{LatestAddition}))^{\frac{1}{\alpha_i}}$$

5. The reputation of type i would then be equal to:

$$R_i(x_i, t) = \{X_{eff.} + \Delta\}^{a_i}$$

6. Total work would then be equal to:

$$X_{tot.} = X + \Delta$$

7. As time progresses it evolves as:

$$R_i(x_i, t) = \{X_{eff.} + \Delta\}^{\alpha_i}(1 - k_i(t_{CurrentAddition} - t_{LatestAddition}))$$

This algorithm is not particularly complex; it simply ensures that due to the time decay, effectively, people move along the reputation curve. It is best to illustrate in a scheme in which a user can always increase the input work without any change to the "difficulty" (which arises from the fact that the first derivative of any reputation type is monotonically decreasing). This is illustrated by the example that the user will always need the same amount of work to position oneself at the position one once was if one does this periodically:

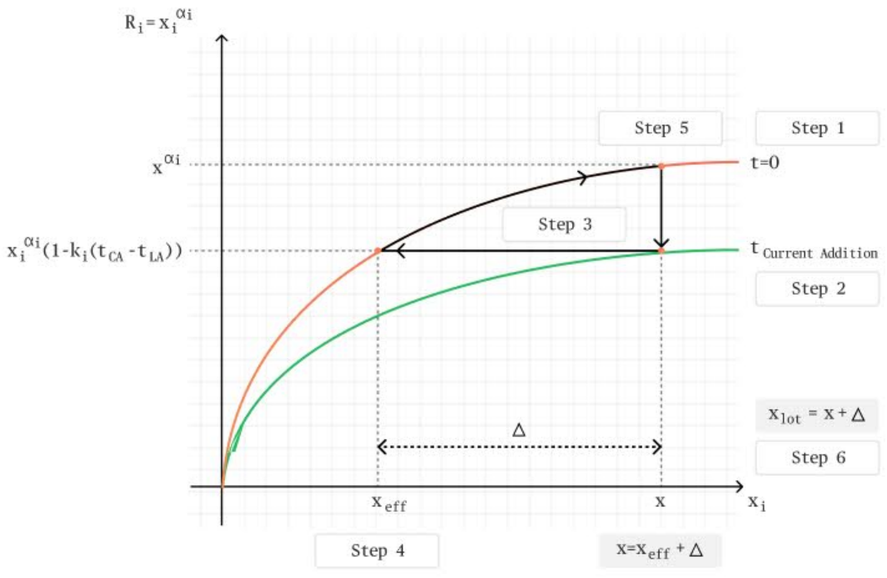


Figure 8: Time decay algorithm

Addressing Dynamic Incentives

The model that has been derived so far possesses a unique incentive line that is fixed throughout time, see Figure 5. This is great, but cannot accommodate a simple and ubiquitously desired property of dynamic weights with smooth transitions. That is to say that if we want to create reputation functions that can dynamically adjust in the face of changing circumstances while ensuring smooth interstate transition, our math has to allow for that.

As a protocol progresses the desired incentives may change which shall result in a redistribution of weights resulting in a change in the slope of the incentive line to any of the axes. If one defines the reputation in absolute terms, the consequence of a dynamic incentive line would be an abrupt change in realized user reputation - all of a sudden some will have more, and others will have less. Aside from an evident user experience problem, there is a more subtle one. Assuming agents strive to maximize their reputation, it becomes a game of prediction of future weights rather than a game of maximizing reputation under current conditions, thus incentives become distorted.

There is also an ideological problem: the system is not meritocratic if all reputation that someone has accumulated becomes negligible at some point in time. The way to mitigate this problem, and ensure that the people that were following the incentives all of the time are the ones that are rewarded the most, even in cases of dynamic weights, is to make reputation continuously accruable - i.e. dynamic.

The way to achieve this is to define reputation in dynamic terms - the rate of reputation accrual per block:

$$\Delta R_i(x_i, t) = x_i^{\alpha_i}(1 - k_i(t - t_{LatestAddition}))$$

$$\Delta R = R_1 R_2 ... R_N$$

$$\Delta R(x_1, x_2, x_3, ..., x_N) = \prod_{I=1}^{n} x_i^{\alpha_i}(1 - k_i(t - t_{LatestAddition}))$$

The decay mechanism is kept in this transition. If at a given point in time, users behave optimally (they behave like the function incentivizes them to) they will be rewarded the most then. Ones that are at an optimal position (in the sense of the distribution of work) all of the time will certainly have the most.

A careful reader would note a small detail. Since everything is defined up to an increment that can be non-zero at all times the global reputation function now is not bounded. But yet again it has diminishing returns for larger values of work. This area is that of ongoing research at the time of publishing of this paper.

There are some candidate solutions: one can incorporate a global time decay or non-additive components into the reputation function. One of those can be a simple parameter solution:

Let us denote increments of reputation (the continuously accruing component) at time t_i as $\Delta R_i = \Delta R(t_i)$. The protocol accrues information about these increments and denotes their sum:

$$\sum_i \Delta R_i$$

The total reputation score is a function of this sum $R = R(\sigma_i \Delta R_i)$. This ensures that users, effectively, move along the reputation curve while accruing reputation increments. As this restriction is global it should not impact the optimal behavior of users within the system since their optimality condition stays intact.

While addressing the issues of intertemporal congruence of incentives, this way of calculating reputation is completely exogenous to the dynamics of the system.

A Note on Negative Reputation

An important note to add here is that although the reputation function is defined in positive-only terms, it does not imply that this functional form cannot be used for enabling some sort of negative reputation scores. In our view, from a game theoretical viewpoint, there must be disincentives to malicious actions, although there must also be a right to be forgotten that can be enacted under certain circumstances.

A wider discussion on the philosophical aspects of negative reputation and the entire problem space of privacy matters evolving around it is far beyond the scope of this article, yet we still would like to address this issue from a vantage point of the mathematical model presented above.

In general, there are two ways to incorporate negative reputation into the model - either by reducing the total work that has been performed by the user (which will reduce his/her reputation as a consequence) or by exogenously decreasing the user's reputation score directly. The first solution reduces the user's work but does not impact his/her "difficulty" in obtaining more reputation points. The second does not impact the amount of work a user has done but increases the "difficulty" of accruing more reputation since the model gives the reputation as an output and is not impacted by its end values. It is a hard choice between these two options since the solution needs to be implementable in code but also fair. Further research is being performed in this area.

A Note on One-off and Continuous Reputation

Work performed is transcribed via the table to the number of points. From the protocol's perspective only points matter thus we are treating these points as work in the sense that they are additive. Tables are composed of tasks (actions) and their point rewards.

There are 2 kinds of tasks: One-off tasks and Continuous tasks.

Tasks	One-off	Continuous
Reputation	$R_o = \prod_{i=1}^{n} = x_i^{\alpha_i}$	$R_c(\sum_i \Delta R_i)$
Time decay	No	Yes
Continuous accrual	No	Yes
Total reputation for one user	$R_{total} = R_o + R_c = \prod_{i=1}^{n} x_i \alpha_i + R_c(\sum_i \Delta R_i)$	

Table 1: Difference between one-off and continuous tasks

One-off tasks can be done only once thus they must not be submitted to the decay function. If they were then after sufficient time they would have no impact on users' reputation and the final result would not be meritocratic. On the other hand, if they are not subject to time decay they would farm reputation indefinitely and create progressively bigger differences between the users.

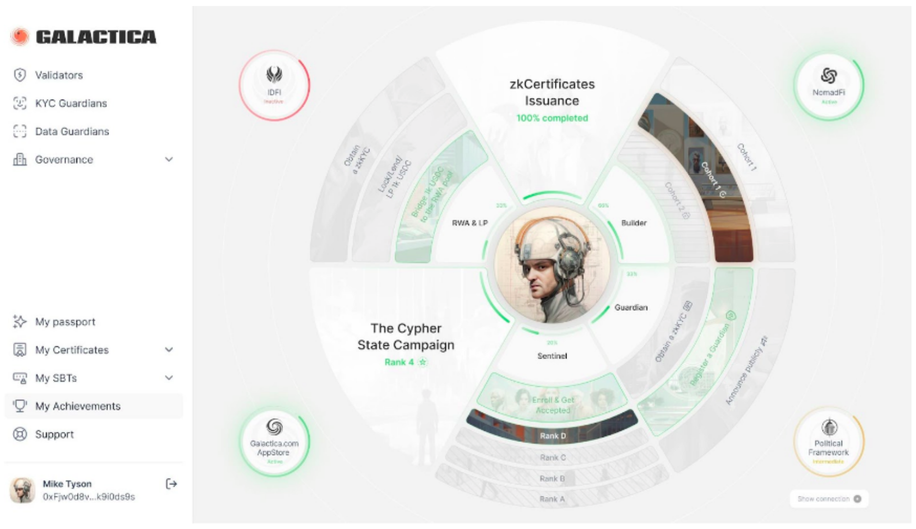
The way to solve this problem is to split reputation into two parts: a one-off part and a continuous part. The one-off part has no time decay but it does not farm reputation - it is a scalar that effectively translates the reputation function in the $\mathbb{R}^+$ space.

The continuous part has time decay and it has all the properties that had been derived until now. Thus, the functional form of the one-off reputation part is the same as ΔR just without the decay. Table 1 as seen below, illustrates the differences between these two types of reputations:

Addendum to Part II: the Skill-wheel Example

What is the reputation wheel and what things are inside of it, to explain.

Let us now go through a specific example of Galactica. Skill wheel represents a collection of many possible reputation types in one diagram.



Different sections of the skill wheel represent different types of reputation. Within all these types different subtasks define the amount of points a user will have in a given task. All these points we normalize to 100 and thus come up with the table of numerical scores:

Rep. Type	**Max weight**	**Relative weight**	**Point for tier 1**	**Point for tier 2**	**Point for tier 3**	**Point for tier 4**	**Point for tier 5**
Sentinels	2.50	11.11%	15	30	45	70	100
CypherState	2.50	11.11%	10	25	50	75	100
LP Dimension	4.00	17.78%	20	40	60	80	100
Data Dimension	1.50	6.67%	25	50	75	100	
Referrer	8.00	35.56%	30	48	65	83	100
Referrer	4.00	17.78%	20	40	60	80	100

Points correspond to the total amount of points awarded if a user is in a given area. They are not additive but absolute. As it is evident the maximum amount of points any given user can have within any given reputation type is 100. Weights are the relative ratios of reputations users will have if they have 100 points in their respective categories. It is instructive to think about the points as percentage points of completion of a given reputation type. For example, if User A has 100 points for the Data dimension reputation type and User B has 100 points in the Sentinels reputation type then User B will have 4/2.5=1.6 times more reputation than User A. This is the information the table holds, now the model needs to be made in line with the theory presented above. The coefficients α_i need to be positive and lower than 1. Let us now demonstrate how these coefficients are found in the table.

Reputation coefficients

All tasks are one-off tasks thus no time decay is incorporated in the model. As it was mentioned the total reputation is calculated as:

$$R_o = \prod_{i=1}^{n} R_i = \prod_{i=1}^{n} x_i^{\alpha_i}$$

To have diminishing returns $\alpha_i \in (0, 1)$. One condition that will ensure this is the normalization of coefficients:

$$\sum_i \alpha_i = 1$$

As weights are given in relative ratios we can choose one of them as a basis to find the absolute value of coefficients from it. Let's pick the Data dimension reputation for example (since it has the lowest weight). Let us construct the table with relative weights w_i; $i \in [1, 6]$:

Rep. Type	**Max weight**	**Relative weights**
Sentinels	2.50	1
CypherState	2.50	1
LP Dimension	4.0	1.6
Data Dimension	1.50	0.6
Referrer	8.0	3.2
Partner	4.0	1.6

Sentinel weight is by definition $w_1 = 1$. Others can be found using the relationship:

$$w_i = \frac{R_i(100)}{R_1(100)} = 100^{\alpha_i - \alpha_1}$$

We end at the system of equations:

$$i \in [2, 6]; \quad w_i = \frac{R_i(100)}{R_1(100)} = 100^{\alpha_i - \alpha_1}$$

$$\sum_i \alpha_i = 1$$

The task is now to find α_i's under these conditions. If one multiplies all equations given above one obtains:

$$\prod_{i=1}^{6} w_i = 100^{\sum_{i=1}^{6}\alpha_i - 6\alpha_1} = 100^{1-6\alpha_1}$$

Subjecting both sides to ln function:

$$\alpha_1 = \frac{1}{6}\left\{1 - \frac{ln\left(\prod_{i=1}^{6} w_i\right)}{ln(100)}\right\}$$

And other coefficients:

$$i \in [2,6]; \quad \alpha_i = \alpha_1 + \frac{ln(w_i)}{ln(100)}$$

If some coefficients are negative the normalization can be done to some larger number, for example, they can add to 2. The table would look like:

Rep. Type	**Max weight**	**Relative weights**	**Alpha coeff.**
Data Dimension	1.50	1	0.1647805257
CypherState	2.50	1.666666667	0.2757049005
LP Dimension	4.0	2.666666667	0.3777648918
Sentinels	2.50	1.666666667	0.2757049005
Referrer	8.0	5.333333333	0.5282798897
Partner	4.0	2.666666667	0.3777648918

The reputation function looks like this (with reputations indexed in the order in which they are in the table):

$$R = x_1^{0.1647805257} x_2^{0.2757049005} x_3^{0.3777648918} x_4^{0.2757049005} x_5^{0.5282798897} x_6^{0.3777648918}$$

Let us now look into specific user examples.

Example 1

Rep. Type	**Points**	**Reputation**
Data Dimension	100	2.135802307
Data Dimension	50	1.905272813

Example 2

Rep. Type	Points	Reputation
Data Dimension	100	2.135802307
Referrer	100	11.39094564

Which are in the 1: 5.333 ratio.

The coefficients are found and maximal reputations are in ratios of the weights. It is now evident that diversifying work in different reputations will produce a compounding effect on the reputation score. One can rescale the final result by multiplying the reputation function by a constant.

Part III: Reputation System Technical Design

In the Parts I and II of this series we have addressed the key ideas underpinning the notion of on-chain reputation. We have elaborated on why, given the nascency of these concepts, we need a new vocabulary to even agree that we are talking about the same subject matter. Likewise, upon defining the vocabulary and agreeing on the terms, we have laid down the desired conceptual properties of on-chain reputation functions.

Our core message is that on-chain reputation is the ultimate use-case for the web3 identity stack, but only so if the desired properties are achieved.

The second part of this series delved deeper into the mathematical properties of optimal reputation functions. We have derived a basic model for game theoretically coherent behavior of effort/reward optimizing agents. We have then shown that there exists such an expressive functional form that enables all core use-cases for persistent reputation.

We are now proceeding to arguably the most challenging part of our exploration - that of defining which technological stack is most optimal for achieving all the properties - mathematical and conceptual - that we have addressed in this series. Cryptographic properties as well as computational feasibility would be the focus of this part.

Brace yourselves for quite a ride, dear reader, as the following text will bring you deep down the rabbit hole of some of the most exotic cryptographic primitives - those that will change the surface and the depth of the internet as we know it.
Let's get started.

The Setup

Implementing a reputation system that is both private and supports penalties for unwanted behavior (called *reputation slashing*) is a stiff challenge.

1. Privacy is essential for Galactica because it deals with encrypted personal data on a public blockchain with publicly viewable transactions. Thus, the

design should eliminate any even purely theoretical attack vectors around the system privacy stack.

2. Reputation slashing is desirable because it can penalize unwanted behavior and enable use cases, such as undercollateralized lending.

As one shall see below, the major unexplored area in designing a holistic web3 reputation system lies in combining privacy and verifiability. When solved for, these two properties combined can enable use-cases that everyone has been speaking of since the very emergence of blockchains as subjects of public discourse, but yet none of which have emerged to-date as production ready systems. To be precise, Galactica is not the first project to attempt to tackle issues such as enabling undercollateralized lending, holistic identity stack or functional on-chain reputation. All these concepts are known, well explored and evidently valuable. But why have none of them really emerged to-date?

Let us offer you a very simple explanation.

Zero-knowledge cryptography (ZKC) and Fully Homomorphic Encryption (FHE) offer great solutions for privacy. They allow proving statements and processing data without disclosing underlying confidential information. But in a practical setup, both of these sets of cryptographic primitives still leak data and these leaks can accumulate over time, thus preventing us from achieving the desired level of privacy. Even if the personal data stays confidential, an outside observer can track the transactions an account has made over time. Selective disclosures through ZK and the contracts an account is frequently interacting with shows which requirements the person owning the account fulfills. In the case of sufficiently diverse requirements fulfilled, many properties of a user's private profile can reliably be inferred by a reasonably equipped outside observer. For example, an account uses a lending platform for Canadian residents only, participates in a DAO for individuals under 20 years, holds a particular NFT, and has sent tokens to an account selling tickets for a specific conference.

All these interactions expose little bits and pieces of a Persistent Identity that can easily be combined to 'reverse engineer' the seemingly encrypted digital footprint of a real person behind it.

A promising solution for preventing this kind of unwanted inference is to split on-chain activity over multiple accounts. This ensures that on-chain disclosures and activity do not accumulate into an identifiable profile. However it requires caution on the side of the user - not to link accounts accidentally, e.g., by sending funds between accounts without mixers. These can be easily ameliorated with modern open source libraries and the most basic OpSec practices.

For reputation slashing and the use cases enabled by it, multiple private accounts are a challenge as the data from all accounts may be relevant. Reputation slashing here is a game-theoretical rather than mathematical concept - it would include any use-case where a user has an incentive to conceal a part of one's Web3 Footprint, such as non-compliance or adverse credit history. For example, to solve for regulatory compliance, a reputation function such as "KYCed user never sent tokens to a sanctioned address" only works if it is enforced to include all accounts. This can be done by maintaining a list of all accounts that share the personal KYC data, identified by the humanID hash, a unique identifier derived from personal data such as name, birthday and nationality. The list needs to consist of encrypted addresses or commitments that the user has to fill out in a ZK proof.

However, there are a few challenges in relation to this approach, when trying to achieve all desired reputation properties.

The Challenges

Combining Reputation Scores Across Accounts

When the user has multiple accounts and the system provides a reputation score for each account, the system should be able to combine those scores into an overall reputation for the owner. Depending on which mathematical primitives the reputation function is using, this might be complicated.

For some reputation functions, it is simple to combine reputation scores of accounts into a total score. One simple example is the reputation function counting transactions to a particular smart contract. The sum of the transaction counts of each account of a user would be the total reputation score. It is always the same, regardless of how the user distributes his activity.

Other reputation functions can not be combined that simply, for example a reputation function for the average swap volume. The same activity can be arranged differently over multiple accounts yielding different reputation results when trying to combine it in a trivial way.

Those more complex reputation functions cannot be combined over multiple accounts consistently. They are more challenging to implement. They require careful consideration to avoid unintended incentives. An approach to solve this is making more of the source data available for the algorithm combining reputation scores across accounts. For the example of the average swap volume, it would be enough to provide the number of transactions and sum of the transaction value, so that both can be combined separately before taking the average. Depending on the complexity of the reputation function, it can be difficult to provide enough source data without reaching the limits of data availability, privacy or computational effort.

Private Reputation Proof Generation

When a user splits the on-chain activity over multiple accounts for privacy reasons, it is desirable to keep the list of accounts that belong to the same user confidential. The reputation system should be able to calculate a total reputation score for a user by combining reputation scores of all accounts. A ZK proof can do this without revealing the list of accounts owned by the user. It can verify inputs, the reputation scores for accounts, with a *Merkle proof*. This way, only the Merkle root is disclosed and not the account's address. Then the proof can combine multiple Merkle proofs to query the reputation score for each of the user's accounts and combine it into an overall reputation score for the user.

This approach works well in theory, however has significant limitations in practice.

1. The tree data, representing a mapping from account to reputation score, needs to be available to generate proofs.

2. The generation of the ZK proof is computationally heavy and increases with the amount of data that is processed and the complexity of the ZK circuit. One of the disadvantages of ZKC is that circuits have a fixed size. This means that, if a circuit has the capacity to combine reputation from 4 accounts, all generated proofs take similar computation effort, no matter if the user has 1, 2, 3 or 4 private accounts.

To keep personal data private, Galactica relies on self custody by users. They store zkCertificates data and keys on their own device(s) and generate ZK proofs locally. Therefore, the reputation proof computation should work on consumer-level hardware with an acceptable user experience regarding simplicity and responsiveness.

Benchmarks with early prototypes for reputation proofs showed that existing solutions are far from feasible. We attempted both - storing reputation in Merkle trees and using Patricia trees, an efficient specialization of Merkle trees for key-value mappings as used by Ethereum.

1. **Patricia trees** are suitable for representing the address to reputation score mapping. But the fixed size property of ZK circuits and the path encoding overhead result in very large circuits. A prototype simplified from an existing Patricia tree circuit has a size of about 100k non-linear constraints for a single account. The prover binary is about 0.4 GB large and 64 GB of RAM is insufficient to compile it.

2. **Merkle tree proofs** are more manageable with about 7k non-linear constraints per proof if the tree holds 2^{32} entries. However, because they do not support a clear address mapping, entries need to be updated in a UTXO-based model. This leads to minutes wait times (depending on the amount of logs and the speed of the endpoint) on the user side when querying and processing blockchain logs to find the user's latest UTXO entries. A specialized indexing service could assist users in this step. By building a database of previous UTXOs, it can preprocess the data and serve user queries much faster. However this comes with a privacy cost of disclosing to an external party what accounts are searched for. When a user queries UTXOs for a specific account, the indexer providing the service can track it in association with the user's IP and track that account and IP. This means the indexing server provider could collect information on a user based on IP tracking and correlating query timings.

There are two promising alternatives for solving the problem of storing and combining reputation privately:

1. Fully homomorphic encryption is a form of encryption that allows general computation on encrypted data without having to decrypt the data first.

$$f(Enc[x]) = Enc[f(x)]$$

Zama.ai integrates this advanced type of encryption into an EVM extension called fhEVM. It enables smart contracts to process encrypted data confidentially and to programmatically define rules for sharing and decrypting the data. This system provides an approach for compliant privacy that is different from Galactica's approach utilizing ZKC. fhEVM allows the computation to take place on the public blockchain, whereas with ZKC the computation takes place on the prover's side and the blockchain only verifies that the result is valid using the ZKP. This means that fhEVM requires far less computational effort on the user side and that ZKC is more efficient to verify. An advantage of ZKC compared to FHE is that it offers more existing implementations for blockchain use-cases such as Merkle trees.

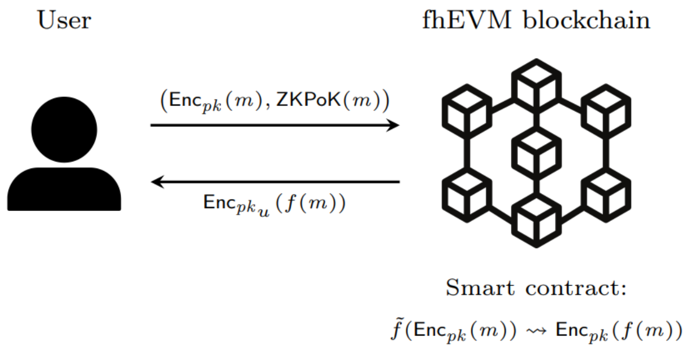

User
fhEVM blockchain
$(\mathsf{Enc}_{pk}(m), \mathsf{ZKPoK}(m))$
$\mathsf{Enc}_{pk_u}(f(m))$
Smart contract:
$\tilde{f}(\mathsf{Enc}_{pk}(m)) \rightsquigarrow \mathsf{Enc}_{pk}(f(m))$

Figure 9: The user can send personal data in an encrypted form to the fhEVM blockchain accompanied with a ZK proof of knowing the content. It is processed confidentially according to logic defined in smart contracts. To output data, a read transaction must pass smart contract checks before the validators reencrypt it with the user's key using Multi Party Computation. [fhEVM whitepaper]

Combining the ZKC and fhEVM unlocks many new capabilities for Galactica. The combination can make the usage of compliant privacy simpler by moving computation effort to the *validators*. Specifically for private reputation proofs, it would require the user to only generate small ZK proofs for the reputation score of each account. The combination into an overall reputation score for the user can be done in an encrypted form in the smart contract. This is not possible without fhEVM, as disclosing the reputation score of a single account can compromise privacy. A third party would be able to recompute reputation scores for all accounts to check which ones have the same reputation score as the disclosed one. Hurdles for the integration of fhEVM into Galactica are: (a) the unknown impact on Galactica validator performance, and (b) that fhEVM is not production-ready for MainNet.

2. Verkle trees are an advanced alternative (utilized by Ethereum*) to Merkle trees serving the same function - to represent large amounts of data in a single root and allow efficient proofs for inclusion of specific data points. The difference is that Verkle trees use vector commitments to achieve smaller proofs and to combine multiple data points into one proof. These properties are ideal for efficiently and verifiably querying reputation scores for multiple accounts in a ZKP so that they can be combined into an overall score on low spec hardware.

   The disadvantage is that Verkle trees are a rather new technology that has less mature implementations and requires more effort for integration in the Galactica technology stack.

There is a benefit in combining both approaches because Verkle trees alone do not solve all issues of reputation proofs. Verkle trees are probably so much more efficient, that reputation proofs are manageable even for many accounts in a single step. However they still leave all the compute gathering required data on the user's side, leading to usability problems.

FHE solves this part efficiently as all the "hard work" is done by professional validators and users only need to do a simple task. Also voluntary disclosure is more elegant on FHE as it can be defined in smart contract logic instead of sending shamir shards to a list of fraud investigation institutions.

Reputation Storage Model

Because reputation functions are user-defined and based on the whole web3 footprint, there are challenges in data availability, storage, and computation.

For example, if a user defines a reputation function counting previous transactions of an account, the calculation requires accessing and processing all past transactions. To query reputation scores efficiently, the source data is aggregated into metrics, such as the transaction count, that can be continuously updated with low effort. As the initial computation of a newly defined metric has to process the existing transaction history, it might be too heavy to fit into the update time of a single block and therefore requires an asynchronous initialization of reputation metrics. This brings challenges for the blockchain consensus because some validators might take longer to process the data.

The storage of those metrics for each relevant account as well as the initial setup and continuous update lead to hardware and compute cost. The economic model to pay for these costs of reputation in the Galactica protocol is an open question.

A potential approach to simplify storage and compute requirements is limiting the reputation functions to a set of predetermined metrics and operations. This would allow precomputing and sharing metrics for all reputation functions so that no costly initialization process is required for new reputation functions. Naturally, this would limit the expressiveness of reputation functions.

Another question is *how to store reputation scores* so that they are accessible to smart contracts, while supporting privacy. Merkle trees and their variants (Patricia trees and Verkle trees) are a promising choice because they support private data lookups through ZK Merkle proofs. However, generating those proofs requires access to the data in the Merkle tree and computational effort for the ZK proof. This can impact the user experience as both downloading Merkle tree data and generating the proof may take significant time (minutes).

An alternative would be a mapping from account addresses and reputation function to a reputation score. This is faster to process, but not able to conceal for which accounts the reputation was used.

Storing reputation scores on-chain in an encrypted way using FHE is an interesting option too. It would be great for privately using reputation, so that others do not know the value. It can also be computed and processed without requiring effort on the user side, but reading and evaluating require user approval. However it would be computationally costly to keep reputation data in encrypted form updated because FHE operations are expensive and transactions impacting reputation are expected to be frequent. Furthermore it is hard to confidentially combine reputation across accounts in FHE because the traces of accounts and transactions impacting encrypted reputation values would be public.

Due to those disadvantages, Galactica focuses on calculating and storing reputation outside of FHE. Many of FHE's advantages are still available because it is possible to encrypt reputation data for onboarding in FHE whenever needed.

Real-Time Reputation

Time-consuming generation of reputation proofs is also a problem for ensuring up-to-date information. A possible attack vector on reputation are flash attacks bundling malicious behavior, so that subsequent actions might be based on outdated reputation information. This problem is only relevant for functions supporting reputation slashing because monotonically increasing reputation functions do not decrease.

To solve this challenge, reputation either needs to be updated during the transaction processing, which is infeasible for privacy through ZK Merkle proofs, or each transaction needs to record reputation requirements so that the post-processing of a transaction in the validator can check if the requirements still hold after all actions performed in the transaction.

Where to Calculate Reputation Scores?

Since calculating reputation functions for an account involves processing lots of data from the transaction history, it is infeasible to pack the computation into a ZK proof.

As reputation should be verifiable and permissionless, it would make sense to calculate reputation directly as part of the blockchain protocol or alternatively use off-chain oracles that make reputation values available on-chain. Integrating it into the protocol as part of the validator node simplifies the data availability, however puts computational load on the validator and the additional complexity is a risk for the stability of the network.

The solution through oracles is more modular and simple, but requires a trusted operator. An option to reduce the required trust is to depend on a network of reputation nodes instead. They could specialize on reputation calculation to be more efficient than a direct integration in the Galactica node, while still providing more decentralization than a single oracle. However, providing real-time reputation would only be feasible with direct node integration to process transactions without significant delays.

Puzzle-Pieces for the Technical Design

Reputation on Private Data

Galactica handles personal data, such as KYC, that is verifiable on-chain through confidential zkCertificates and ZK proofs. It would be nice to utilize this data in reputation functions. For example it would be useful to have a reputation function that considers if a person's name is on a sanction list. However this personal data cannot be used directly for reputation because the raw data is not available to third parties calculating or verifying reputation.

There are two ways to solve this challenge:

1. **Intermediate Disclosures:** To provide data for a reputation function, a user can publish a selective disclosure on-chain. For example, this can be a ZK proof about not being on a sanction list. After submitting this proof in a transaction and verification on-chain, an SBT is issued for the sending address. This SBT can be used for calculating the reputation.

2. **Fully homomorphic encryption:** Using FHE, personal data can be stored in encrypted form on-chain. It can be processed in a confidential manner by anyone trying to calculate the reputation of an address. The result would be an encrypted reputation score that can be used in further FHE operations, such as receiving an encrypted amount of tokens for an airdrop (example implementation of encrypted tokens by Zama). Compared to intermediate disclosures with ZK proofs, this approach works without requiring the user's interaction. Depending on the implementation, the user only has to approve providing the encrypted data to the reputation calculation beforehand. The disadvantage of this approach is

that the calculation takes significantly longer to compute and that the set of available operations is limited and not turing complete.

How to Divide On-chain Activity?

To prevent selective disclosures from accumulating into a trackable profile of a user, activity on-chain can be split to different accounts. For example, a user might use one account for long term investments on an account secured by a hardware wallet, another account for regular governance duties on a hot wallet and a third account for meme coins and games. The user can just generate or derive new accounts on his own as on any other EVM-compatible chains.

When activity is split for privacy reasons, there are a few challenges of operating this way without accidentally linking the accounts in the public transaction history. Funds should be moved through privacy mixing protocols and selective disclosures should not show that both accounts share the same humanID for a DApp.

Linking Accounts for Reputation

As discussed above, it is useful and we believe necessary for the reputation system to enable the combination of reputation data from multiple accounts, for example to prove that the user has never interacted with sanctioned addresses, regardless of the account used.

To solve this challenge, the reputation system needs a way to link a user's account without disclosing that link publicly. This can be done using ZK circuits and a list of encrypted addresses of the user. Doing everything in a ZK proof would be too expensive computationally to run it on the user's machine in self custody. Therefore it makes sense to limit the ZK proof to the minimum by only providing intermediate reputation scores for accounts. The rest is pushed to other computation layers, the reputation calculation in the node and the FHE smart contract system for combining account scores into a total user score. This way a lot of the heavy computation work can be outsourced to professional validators, so that the remaining work is as user friendly as possible.

The ZK proof on the user side is still required because the node and FHE can not keep the confidentiality of which accounts belong to the user. If the minimal ZK proof is still too heavy for user hardware, Verkle trees can be a possibility to make them efficient enough.

Whenever a new account registers using a zkKYC, the ZK proof can add the account to the list of accounts associated with the humanID behind the KYC. This list can be used in ZK proofs combining the reputation of all addresses as long as the ZK proof can hide which accounts the reputation was queried for.

The limitation of this approach is that the reputation proof only covers accounts that have registered a zkKYC before.

Sharing zkKYC Between Accounts

A zkKYC can be shared between multiple accounts because Galactica's zkKYC system allows a KYC to authorize an account using a signature of the account holding the zkkYC. The ZK circuit ensures that the signature is valid and matches the holder of the zkKYC.

Technical Design Conclusion

As discussed above, a number of questions remain, about the technical design of Galactica's reputation system. The *main challenges* for achieving all desired reputation goals are:

1. Combining user privacy and protection against tracking, with support for reputation slashing requires to force inclusion of all accounts.
2. Finding solutions with sufficiently good user experience as many approaches increase the system complexity and thus cost the user more time to access data, generate proofs or, in the fhEVM case, impact validator performance in a way that decreases the blockchain throughput.
3. Finding a solution for protecting against reputation flash attacks, such as updating reputation in "real-time" during transaction processing without overloading the validator nodes.
4. Storing reputation data in a way that is accessible, efficient, and private enough.

Finding feasible solutions to those challenges can involve adopting new technologies, such as fhEVM, making simplifications or reducing the requirements. For example removing the support for reputation slashing would simplify the reputation system significantly. Users would not need to be forced to include all accounts in reputation calculation and there would not be reputation flash attacks requiring real-time reputation. However those simplifications also limit the amount of use cases, such as undercollateralized lending. It is up to future discussions which requirements are essential for a successful reputation system and what trade-offs exist.

The early versions of Galactica's reputation system will focus on a subset of features forming its MVP. Based on the community and builders' feedback we will learn more about its performance and importance of the various components of the reputation system.

*Verkle trees are used to improve the efficiency and scalability of data storage and retrieval on the Ethereum blockchain. They provide a more efficient way to organize and access data, particularly for state data, which includes account balances, smart contract storage, and other information critical to the operation of decentralized applications (DApps) on the Ethereum network. By using Verkle trees, Ethereum aims to reduce storage costs and increase the speed of accessing this data, which is necessary for enhancing the overall performance and scalability of the platform.

Appendix:

Reputation slashing

This section provides a definition of reputation slashing and discusses its implications. *Reputation slashing* is defined as the decrease of someone's reputation due to an undesired action. An example for a reputation function supporting reputation slashing would be modeling compliance with a sanction list. When a user sends funds to a sanctioned blockchain address, this transaction would decrease this user's reputation score.

Reputation slashing is different from *negative reputation*, which is defined by reputation functions that can yield reputation scores below zero. It is enough if the reputation difference caused by an action can be below zero. Therefore functions with reputation slashing support can still have lower bounds, such as 0.

The opposite of reputation slashing would be a reputation function that can only yield monotonically increasing reputation scores.

Reputation slashing is desirable because it enables use cases that penalize malicious behavior. Furthermore it allows reputation to be used as collateral that can be lost if some predefined condition is met.

Reputation slashing also brings game theoretical implications that challenge the technical design of the reputation system. In a system with monotonically increasing reputation, a user is incentivized to include all actions in the input of the reputation calculation. As soon as reputation slashing could decrease a user's reputation, the user has an incentive to hide activity from the reputation calculation, for example by sending it from another account or hiding it in the privacy system provided by Galactica through ZKC, Merkle trees and FHE. Therefore the reputation system needs to be designed in a special way that ensures that all user actions impacting reputation are considered. This introduces complexity and limitations.

Reputation slashing also brings game theoretical implications that challenge the technical design of the reputation system. In a system with monotonically increasing reputation, a user is incentivized to include all actions in the input of the reputation calculation. As soon as reputation slashing could decrease a user's reputation, the user has an incentive to hide activity from the reputation calculation, for example by sending it from another account or hiding it in the privacy system provided by Galactica through ZKC, Merkle trees and FHE. Therefore the reputation system needs to be designed in a special way that ensures that all user actions impacting reputation are considered. This introduces complexity and limitations.