MATRYX

Innovation Ecosystem Built on the Blockchain.
Token Ticker - MTX
Updated February 28, 2019

Steve McCloskey[1], Keita Funakawa[2], Vincent Brunet[3]
Scott Morgan[4], Edgardo Leija[5], Adam Simon[6], Kyle Lee[7], Max Howard[8]

***Abstract*— Innovation is often an iterative process; scientists have mastered the art of standing on the shoulders of giants. New discoveries are the result of collaboration between mathematicians, scientists, and engineers alike, each building on thousands of years of established knowledge. In this paper, we present Matryx: a platform that enables and incentivizes this type of collaboration. Matryx is composed of 1. a collaboration system to track innovation and 2. a bounty system to award solutions to scientific problems. The focus of this whitepaper is to provide a technical overview of Matryx Platform's design.**

I. INTRODUCTION

In 2000, the Henri Clay Institute of Mathematics selected seven difficult problems in science, technology, engineering, and mathematics (STEM) and offered a $1,000,000 prize for a solution to any of these problems. In 2003, Russian mathematician Grigori Perelman became the first person to solve one of these "Millennium Problems": the Poincaré conjecture.

The Poincaré Conjecture has baffled mathematicians since its formalization in 1903 by Henri Poincaré, the father of topology[1]. Richard Hamilton, Professor of Mathematics at Columbia University and one of the most brilliant mathematical minds in history, laid the foundations for Perelman's proof. Christina Sormani, Professor of Mathematics at the City University of New York, broadly describes the novel efforts [1] of Hamilton and Perelman:

> "In recent years Hamilton had been investigating an approach to solve this problem using the Ricci Flow, an equation which evolves and morphs a manifold into a more understandable shape. Then in late 2002, after many years of studying Hamilton's work and investigating the concept of entropy, Perelman posted an article which combined with Hamilton's work would provide a proof of Thurston's Geometrization Conjecture and, thus, the Poincaré Conjecture."

After seven years of peer review, Perelman was awarded the Millennium Prize. In an unexpected turn of events, he declined the prize money, arguing that the contributions of Hamilton and other mathematicians played a significant role in the development of his final solution. He declared his "disagreement with the organized mathematical community"[2], arguing that his peers deserved just as much of the award and recognition and that it would be wrong to claim the money and fame for himself.

Many mathematicians - like Perelman - consider these lump awards to be unjust. New ideas are usually collaborative in nature and are based on other people's existing ideas. Large awards incentivize competition rather than collaboration and fail to reward most contributors. As such, these lofty and unbalanced rewards may actually be counterproductive. Perelman is only one of many researchers who have rebelled against common incentives. Current incentive structures do not reflect the needs of collaborative fields.

Matryx provides a structure that reduces the friction of rewarding collaborators. Rather than attributing all the credit to one person or one group who proposes a solution that is built on other people's work, Matryx tracks the provenance of assets, enables collaboration, and divides rewards amongst all participants. In this way, Matryx can reward each unit of progress towards a given goal. Solitary research and siloed collaboration are discouraged, while open collaboration in pursuit of a shared reward is incentivized.

[1]Topology is the study of the properties that are preserved through deformations, twistings, and stretchings of objects. See Eric Weisstein's discussion on MathWorld: http://mathworld.wolfram.com/Topology.html

II. Problem

A. Distribution & Discovery

Research in STEM-related fields and academia is fragmented. Universities, corporations, institutions, and individuals host and share their resources in separate "siloed" databases, often with tightly controlled access. Even access to carefully curated private research repositories is not easily purchasable by those willing to pay. It is nearly impossible to find all current and past research on a given topic without navigating a maze of citations and licenses.

Innovation in STEM is hindered by this high friction of discovery. Researchers may be attacking the same problems with no knowledge of each other's respective progress. This wastes brainpower, time, and money. Organizations like SciHub have attacked this problem by circumventing technical and legal controls on information and research, but a solution within the bounds of the law is needed. In 2016, a gathering of ministers of science in the EU demanded all scientific research papers be made free and open by 2020 [3]. But this type of legal reform is time-consuming and has no guarantee of success, and doesn't provide a technical solution to the difficulty of discovery and dissemination of research.

Also, many academic researchers struggle with publishing quality research because of scarce funding and pressure to move up in the academic world. In 2014, Jeffrey Beall of the University of Colorado coined the term "predatory publishers", referring to publishers who encourage researchers to publish without proper peer review[4]. As a result, researchers must publish high volumes of low-quality papers due to demands to advance their careers in their respective institutions.

B. Attribution

In research and creative projects it is difficult to attribute value across contributions. Contributions are rarely tracked with any degree of accuracy, and there is not always a clear path from problem to solution. As such, owners are improperly (or not at all) compensated for subsequent usage and "remixing" of their works. Without clear attribution, incentives for innovation do not accurately reflect contribution. This creates disincentives for the creation and distribution of valuable works. This problem is common in STEM research, digital media creation, and a wealth of other fields. Some generalized solutions to attribution in communities have been proposed by projects like Backfeed and Mediachain, but no mature distributed attribution system has been deployed.

III. Matryx: A collaboration Platform

Matryx is a decentralized smart contract system for the creation, attribution and incentivization of scientific work. It provides a record of open units of scientific work ("Commits"), problems to be solved ("Tournaments"), and their proposed solutions ("Submissions"). This system of Commits, Tournaments, and Submissions is the core of the Matryx Platform. The Matryx platform is an upgradeable system and can support the introduction of new functionality. The ability to upgrade the platform can be used to introduce new features (such as the upcoming Matryx Marketplace) after having been migrated to the Ethereum Mainnet. The platform's upgrading mechanism will be discussed in **System Design IV**. Matryx can also interface with research tools such as Calcflow[2] and Nanome[3] to accelerate the rate of mathematical and nanotechnological innovation. We will now discuss the various components of Matryx, beginning with Matryx Commits.

A. Commits

A Matryx Commit is a unit of work on Matryx tracked by the Ethereum blockchain. The anatomy of Matryx Commits is as follows:

1) *owner* - The owner of the Commit
2) *timestamp* - The unix-epoch time in seconds at which the Commit was created on chain
3) *groupHash* - The keccak256 hash of the group working on the line of work this Commit is a part of
4) *commitHash* - The Commit's unique identifier; a keccak256 hash of: the creator of the Commit, the salt used to claim the Commit and the *content* of the Commit
5) *content* - A deterministic content Multihash (IPFS, Swarm, etc)
6) *value* - The value that has been assigned to the Commit
7) *ownerTotalValue* - The total value that the owner of this Commit has generated in this Commit's chain
8) *totalValue* - The total value of this Commit chain
9) *height* - The length of this Commit chain
10) *parentHash* - The parent Commit's keccak256 hash
11) *children* - keccak256 hashes of all child Commits whose content is derivative of this Commit

To create a Commit, a user must perform a commit-reveal process to ensure that their content is not front-run by malicious actors (see V-B). We have renamed commit-reveal to "claim-create" to avoid confusion from reusing the term "commit". This process requires the user to make two transactions to record their work:

1) claimCommit
2) createCommit *or* createSubmission

To claim a Commit, the user must provide the Commit's hash to the *claimCommit* function. As mentioned above, the *commitHash* is the keccak256 hash of the

[2]See https://nanome.ai/calcflow/
[3]See https://nanome.ai/

user's address, some secret or salt, and the Commit's *content*. Once the *claimCommit* transaction has been mined, the user is free to create their Commit by calling the *createCommit* function. Arguments to this function include *parentHash* (if this Commit was a continuation of prior work), the salt used in *claimCommit*, the *content* of the Commit, its *value*, and a special $isFork$ flag. When the user sets $isFork$ to true, or creates a Commit without a parent, a new group is created for their Commit. When a member of a Commit's group adds a Matryx user to that group, the new group member is then able to create Commits without the need to transfer MTX to that parent. This is, users in a group can freely commit off of one another's commits in the same chain. Conversely, passing true as $isFork$ will assign a new group to the user's Commit, allowing the user to work alone or with a new group of their choosing. This will require the user to transfer to the forked Commit its $totalValue$ in MTX. All ancestor Commit creators can then withdraw the share of MTX that corresponds to the sum of their contributions' values in proportion to the total value of the chain. After calling *claimCommit*, the user can alternatively call *createSubmission*, which will create a new Commit and a Submission. The Submission is then submitted to the Tournament designated by the user. We will define Tournaments and Submissions in precise terms in the two following sections.

B. Tournaments

Tournaments are multi-round competition smart contracts. Tournament requirements are posted publicly and are hashed in the smart contract system. The Tournament owner determines a reward, which is locked into a smart contract for the duration of the Tournament. This is done to arbitrate the transfer of funds without the need for a third party, as would be the case in competitions hosted by centralized entities. Once the Tournament is public, users can create Submissions for that Tournament. Applications such as Nanome, Calcflow, and the Matryx web app will be the preliminary interfaces capable of publishing generated content to the Matryx Platform. However, Submission content can come from any external application.

When the user decides to make a Submission to an open Tournament, the underlying Commit content is put together with some additional content (title and description), signed, and made publicly available on the Tournament. At the end of a Tournament, the Tournament creator distributes the bounty. If there are multiple eligible winners, the Tournament creator can specify the distribution that will be used to split the bounty across the winning submissions. We will now discuss the structure of a Tournament in further detail.

Tournaments consist of one or more Rounds of content submission and evaluation. Formally, a Tournament is composed of the following data:

1) *version* - The Tournament's internal version
2) *owner* - Owner of the Tournament
3) *content* - A deterministic, decentralized storage address to a JSON object containing information about the Tournament. This information includes the Tournament's title, description and links to any additional files describing the nature of the competition
4) *bounty* - The bounty attached to the Tournament, as assigned by the Tournament creator
5) $entryFee$ - The cost to enter the Tournament as a participant, as determined by the Tournament creator
6) *rounds* - A list of all Rounds on the Tournament
7) $entryFeePaid$ - All entry fees paid by each user

To clarify, a Matryx user first creates a Tournament with *content*, *bounty*, $entryFee$, and the initial round details. At the time of the Tournament's creation, *bounty* MTX will be transferred from the Tournament owner to MatryxPlatform. This prevents Tournament creators from advertising many high value Tournaments that are not backed by their own balance of MTX. Subsequently, when a Matryx user enters a Tournament, $entryFeePaid$ is set for their address, and $entryFee$ MTX is transferred from their balance to MatryxPlatform's. This user is free to leave the Tournament at any time, at which point $entryFee$ MTX will be returned to their account. In the event that $entryFee$ changes after the user has entered the Tournament, upon leaving the Tournament, the user will still be sent the amount of MTX that they originally transferred in order to enter. *owner* is the sole address able to create Rounds, select winners and otherwise update the state of the Tournament, including any of its *content*. At any time, anyone can increase a Tournament's bounty by making a specific call to the Tournament. Additionally, each Tournament is said to be in one of several time-dependent states. We introduce the following notation to describe these Tournament states: Let S_i^t be the i^{th} state of a Tournament and R_j represent the j^{th} Round. These states are:

1) S_0^t - Not Yet Open
2) S_1^t - On Hold
3) S_2^t - Open
4) S_3^t - Closed
5) S_4^t - Abandoned

To clarify:

1) A Tournament will be in S_0^t if R_0 has yet to begin.
2) A Tournament will otherwise be in S_1^t if a Round R_j has yet to begin and R_{j-1} is Closed.
3) A Tournament will be in S_2^t if the current Round, R_j, of the Tournament is in its Open state.
4) A Tournament permanently enters into S_3^t when the Tournament owner decides to close the Tournament.
5) A Tournament permanently enters into S_4^t if the Tournament owner fails to select winners by the

end of the current Round R_j, or if there were no Submissions made to said Round.

Like Tournaments, Rounds exist in one of several time-dependent states. We introduce the following notation to describe these Round states: Let S^r_k be the k^{th} state of a Round. These states are:

1) S^r_0 - Not Yet Open
2) S^r_1 - Unfunded
3) S^r_2 - Open
4) S^r_3 - In Review
5) S^r_4 - Has Winners
6) S^r_5 - Closed
7) S^r_6 - Abandoned

Each Round of a Tournament, including the current Round R_j, is composed of:

1) *start* - the Round's start time
2) *duration* - the Round's duration in seconds
3) *review* - the Round's review duration in seconds
4) *bounty* - the Round reward

Until time *start*, the Round remains in the initial state, S^r_0. At *start*, the Round transitions to state S^r_2. Contributors may register new Submissions to R_j at this time. Once time *duration* has passed, R_j will enter state S^r_3, at which point no more submissions may be made to R_j. The Tournament owner may then choose a set of winning submissions until time *start+duration+review*. This set may consist of one or multiple Submission hashes. Upon the Tournament owner selecting a set of winning submissions from R_j, *bounty* MTX is allocated among the winning Submissions. During this time, the Tournament owner must also choose one of the following courses of action for their Tournament:

1) *DoNothing* - Keeps R_j open.
2) *StartNextRound* - Closes R_j and opens R_{j+1}
3) *CloseTournament* - Closes R_j and the Tournament

In the case of the *DoNothing* option, R_j will transition to S^r_4 until time *start+duration+review*. At *start+duration+review*, R_{j+1} will be become the active Round and R_j will again transition into S^r_5. With this option, R_{j+1} will begin at *start+duration+review*, end at *start+duration+review+duration*, and otherwise inherit its parameters from R_j.

In full, R_{j+1} has the following parameters:

$$\begin{aligned} start_{c+1} &= start + duration + review \\ duration_{c+1} &= duration \\ review_{c+1} &= review \\ bounty_{c+1} &= bounty \end{aligned}$$

If the Tournament contains less than *bounty* MTX, the remainder of the Tournament's MTX will instead be used to fund R_{j+1}. If the Tournament has no MTX left, round R_{j+1} will transition to state S^r_1 at time *start+duration+review* until the Tournament owner adds more MTX to the Tournament and allocates some to R_{j+1}. Then, as long as the time is still less than *start+duration+review+duration*, R_{j+1} will transition to state S^r_2 and become an active round.

If the *StartNextRound* option is chosen, R_j will transition to S^r_5 and Round R_{j+1} will be created with the new Round parameters passed by the Tournament owner in tandem with their winning Submission set. In this case, the Tournament must have enough MTX in its balance to allocate to R_{j+1} the *bounty* that was specified.

Finally, the *CloseTournament* option places both R_j and the Tournament into their Closed states (S^r_5 and S^t_3 respectively) and distributes all remaining Tournament funds to the winners of R_j.

Without any mechanism to stop them, Tournament creators may be motivated to use a Tournament's bounty to collect valid solutions, yet never pay any Submission creators for their work. As a result of S^r_6, this behavior is made impossible; in the event that the Tournament owner elects to attack the system by refusing to properly select a set of winning submissions or otherwise fails to select this set by *start+duration+review*, R_j will enter S^r_6, wherein all participants who have submitted to the round will be able to withdraw an evenly-divided portion of the Tournament's bounty. That is: regardless of the Tournament owner's actions, once the Tournament has been created, the associated bounty *will be distributed* to at least some if not all participants. In the event that there are no participants, the Tournament owner will be allowed to recover their funds. In this way, Matryx attempts to counteract the risk of malicious Tournament owners refusing to reward Submission creators for their work. The reputation system also plays a role in mitigating this attack (see section V-B).

C. Submissions

After entering a Tournament, a user can create one or multiple Submissions only when the Tournament's current Round R_j is in S^r_2. A Submission is defined as:

1) *tournament* - The tournament address the submission was made to
2) *roundIndex* - The index of the round to which the submission was made
3) *commitHash* - The address of the Commit the submission references
4) *content* - A deterministic storage address to the content of the submission, including its title and description
5) *reward* - The MTX reward this submission has won
6) *timestamp* - A unix-epoch timestamp in seconds for the creation of the submission

As described above, each Submission is submitted to a particular *roundIndex* of a *tournament*. A particular *commitHash* can be included in multiple Submissions, but said Submissions cannot be entered to the same *roundIndex* of the same *tournament*. We will now describe the Matryx System design.

IV. System Design

Sitting above each deployed Matryx contract is MatryxSystem: a smart contract responsible for delegating work to the intended library based on information from the incoming call. MatryxSystem is the backbone of Matryx, and underlies the proper functioning of each call made to contracts on the Platform. A UML representation of the MatryxPlatform and MatryxSystem can be seen in **Figure 1**.

Calls to MatryxPlatform happen in several steps:

1) Caller type lookup on MatryxSystem
2) Library name lookup on MatryxSystem
3) Library address lookup on MatryxSystem
4) Calldata transformation lookup on MatryxSystem
5) Call to library function

In further detail:

1) *Caller type lookup:*
   When a call is made to the Platform, the Platform's fallback function is invoked. The fallback first makes a call to System to determine what type of Matryx entity (Platform, Commit, Tournament, or Unknown), is making the call.
2) *Library name lookup:*
   Using the type from 1), Platform makes a call to System to look up the name of the library associated with that type.
3) *Library address lookup:*
   Using the library name from 2), Platform makes a call to System to look up the current library address associated with that name.
4) *Calldata transformation lookup:*
   Using the library name and function signature from the incoming calldata, Platform makes one final call to System to determine how to transform the incoming calldata for the respective library call.
5) *Call to library function:*
   With the transformed calldata, Platform makes the call to the library function and returns the result.

While most of these steps are quite straightforward, we believe step 4) warrants an explanation. Calldata transformation information contains the following data:

1) $modifiedSelector$ - The first four bytes of the hash of the library function responsible for performing the operations expected by the incoming call.
2) $injectedParams$ - A list of values to be inserted into the incoming call's calldata before the pre-provided parameters.
3) $dynamicParams$ - A list of parameter indices indicating which parameters from the incoming call are dynamic.

After the Platform receives this data from MatryxSystem, it will synthesize a call to the library at the address received from System by modifying the incoming calldata in the following way:

1) Platform will replace the function selector of the incoming call with $modifiedSelector$ returned by System.
2) Platform will inject $injectedParams$ directly after $modifiedSelector$. $injectedParams$ includes the address of the original caller to the platform, so that $msg.sender$ is preserved in the call to the library, and the address of MatryxPlatform, so that the library has access to the data stored by the Platform.
3) Platform will offset each parameter whose index is contained within $dynamicParams$ by the length of $injectedParams$. This allows MatryxPlatform to add parameters to an incoming call while maintaining the integrity of the call's existing parameters.

MatryxPlatform will then make a delegatecall to the library at the address received from System with this transformed calldata.

A. Upgradeability

In order to further iterate on the functionality of the platform, the Matryx team took considerable steps to ensure that MatryxPlatform contained a system to safely and effectively upgrade its behaviors and storage model. This is done via the aforementioned MatryxSystem.

There are two types of upgrades possible for Ethereum smart contract systems:

1) Functionality upgrades
2) Storage upgrades

For purely functional upgrades, MatryxSystem can register a library containing new functionality at any time. All versions of MatryxPlatform are capable of undergoing functionality changes and bug fixes in this manner without the need to recreate the Platform's storage within another contract.

Unlike a purely functional upgrade, storage upgrades require the creation of a new Platform version on MatryxSystem. Each new Platform library under must then be registered to MatryxSystem under that version. MatryxSystem's $currentVersion$ field must also be updated, so that any calls made to this "new" Platform (or any new contracts created by it) will be directed to their respective upgraded libraries. The upgraded libraries for a Tournament or Commit can then safely change the structure of their data so that new Tournaments and Commits can support new features.

In the interest of security, upgrades to Matryx can only be performed by the Matryx team at Nanome.

V. Platform Security

A. Trust

Unfortunately, game theory alone cannot completely eliminate all malicious behavior on a distributed system. As a result, a reputation system is a necessary component of Matryx. There has been prior work done on distributed reputation systems, such as Eigentrust[5], Eigentrust++[6] and PeerTrust[7]; however, after critical evaluation and partial implementation, none appeared to be fit for an application running directly on the Ethereum blockchain. The Matryx Team has thus opted to store and compute reputation values within a centralized context. As a result, we've removed reputation's effect on the mechanics of transactions made to the platform. Aside from reputation and sound game theory, future iterations of the platform will also attempt to place additional checks on Tournament owners by, for example, requiring the use of an identity system for all Tournament creators.

B. Attacks

In designing Matryx, several potential attack vectors were considered. The following are some of those attacks and the strategies we employed to mitigate them:

1) Tournament bounties give Submission creators a strong incentive to bias the results of a Tournament in their favor. One way a Submission creator might attempt to achieve this is by entering similar Submissions multiple times, in the hopes of increasing the likelihood of being evaluated in Tournaments with many Submissions. Because Tournaments do not allow users to create multiple Submissions in a Round using the same Commit hash, the only feasible way to create multiple similar Submissions is to claim and create multiple Commits with slightly modified content. This attack is therefore somewhat mitigated, as there is a significant time and gas overhead for each new Submission the user wishes to generate. Additionally, because creating a Submission generates an event, Tourmament owners can efficiently filter Submissions by sender, allowing them to see easily detect duplicate Submissions by the same user. However, an anonymous Submitter may still perform a Sybil attack on a Tournament by making Submissions with multiple Ethereum public keys. We have attempted to minimize the cost of this attack by introducing entry fees to Tournaments, as described in Section III-B. With an appropriately chosen Tournament entry fee, the potential benefit from making multiple Submissions can be balanced against the opportunity cost of having to pay an entry fee for each new anonymous public key used to submit a duplicate Submission. Additionally, user reputation may aid in mitigating this attack to a small degree by allowing honest Matryx users to negatively influence the reputations of users who attempt to perform this attack.
2) One commonly exploited blockchain attack is the phenomenon of frontrunning. Because Matryx is deployed on a public blockchain, pending transactions are visible to anyone. This was of particular concern for Matryx Commits. A malicious actor could monitor pending transactions that create new Commits, replicate their content, and send their own transaction with a higher gas price, increasing the probability that the fraudulent, replicated Commit transaction will be mined first. Without a Commitment scheme ("commit-reveal"), the contents of the Commit would be attributed to the malicious user instead of the honest one. The current implementation of the Commit creation process in Matryx uses a Commitment scheme to completely eliminate the possibility of this attack. Commits are created in two separate transactions: The first transaction claims a Commit hash for future use, and the second one reveals the contents of the Commit. Matryx requires that the second transaction occur at least one block after the user has already claimed the Commit. This ensures that malicious actors monitoring the system cannot frontrun Commit creators, since the Commit contents are only revealed on the second transaction and are reserved for the user address that made the original claim.
3) A malicious Commit creator could slightly modify the content of an existing Commit and create a new Commit without referencing the original Commit as a parent. In such a case, the creator of the original Commit would be left without credit, despite the fact that a derivative Commit was created from it. We plan to mitigate this attack by running similarity checks against existing content when new content is uploaded. Still, this solution will only mitigate attacks made using Nanome's interfaces and will fail against command-line attacks. In the command-line case, Commit timestamps offer definitive proof of authorship and can be used to settle legal disputes outside of Matryx in the event of this particular attack.
4) Tournament owners may also attempt to attack the Platform. The most obvious attack available to a malicious Tournament owner is to refuse to reward their Tournament's bounty to any of its participants. The Tournament owner could then benefit by seeing all work performed on a problem without ever having to credit those who did the work to solve it. As stated earlier, The Matryx Platform eliminates this attack with state S_6^r, wherein all users who have submitted to the round are able to withdraw an equal share of the Tournament's bounty in the event that the Tournament owner does not select at least one winning Submission.

VI. Future Work

A. Reputation and Peer Review

Reputation is currently an auxiliary component to give contributors and Tournament posters a public view of how their contributions are valued. Determining the value of a contribution given a wide range of bounties will likely take human interaction. Individuals who have a monetary stake in a reward mechanism will initially have to place some trust in the Tournament creator and assume that the creator will reward the Submissions fairly based on the quality of their content. Contributors will have a public record of where awards were sent and may judge their Tournament creators accordingly.

Future implementations may use a voting system by the crowd to help determine contribution value. These votes may be Sybil attacked (though with on-chain voting, the gas cost of voting will help mitigate this). Additionally, the validity of these votes must be taken into account as voters may not have the expertise needed to formulate an accurate assessment of a contribution. This leads the system into a model where curators who have gained a higher reputation in certain context-areas may facilitate the value assignments. We will explore the possibility of combining financial incentives and willing collaboration to generate a higher reputation.

B. Marketplace

The Matryx Platform will also serve as a medium for the design and open exchange of digital assets through a marketplace system; any user with MTX tokens will be able to buy and sell assets under the licensing agreements of the asset owner. The metadata for these objects will be stored on the blockchain while the objects themselves will be stored off-chain (similarly to how Tournaments and Submissions store their contents).

C. Judging Boards

Rather than trusting the Tournament owner to judge a Tournament, it may be advantageous to select a group of third-party judges. This group should consist of experts in the Tournament's field. Many structures could be implemented, including direct or weighted voting and an oversight board with veto power. It would be possible to reward these reviewers. Determining appropriate structures for this would require significant time and incentive analysis. As such this capability will not be implemented until later versions of Matryx.

D. Private Tournaments

A system can be conceived where the results of a Tournament are made private, by encrypting all Submissions with a public key of a key pair created by the Tournament owner. This would ensure that no one but the Tournament owner could access the Submissions. The Tournament could proceed as normal, with the winning Submission from each round revealed publicly.

E. Alternative Incentives

It may be that monetary incentives are not applicable to scientists. Often it is fame or recognition for achieving something that is sought after. Bounties are not limited to a financial reward like the Millennium Prize. Title-based rewards registered by trusted authorities could potentially be placed as bounties.

References

[1] Christina Sormani. "Hamilton, Perelman and the Poincare Conjecture". In: (). URL: `http://comet.lehman.cuny.edu/sormani/others/perelman/introperelman.html`.

[2] Jeffrey Ritter. "Russian mathematician rejects $1 million prize". In: (2010). URL: `https://phys.org/news/2010-07-russian-mathematician-million-prize.html`.

[3] Nadia Khomami. "All scientific papers to be free by 2020 under EU proposals". In: (2016). URL: `https://www.theguardian.com/science/2016/may/28/eu-ministers-2020-target-free-access-scientific-papers`.

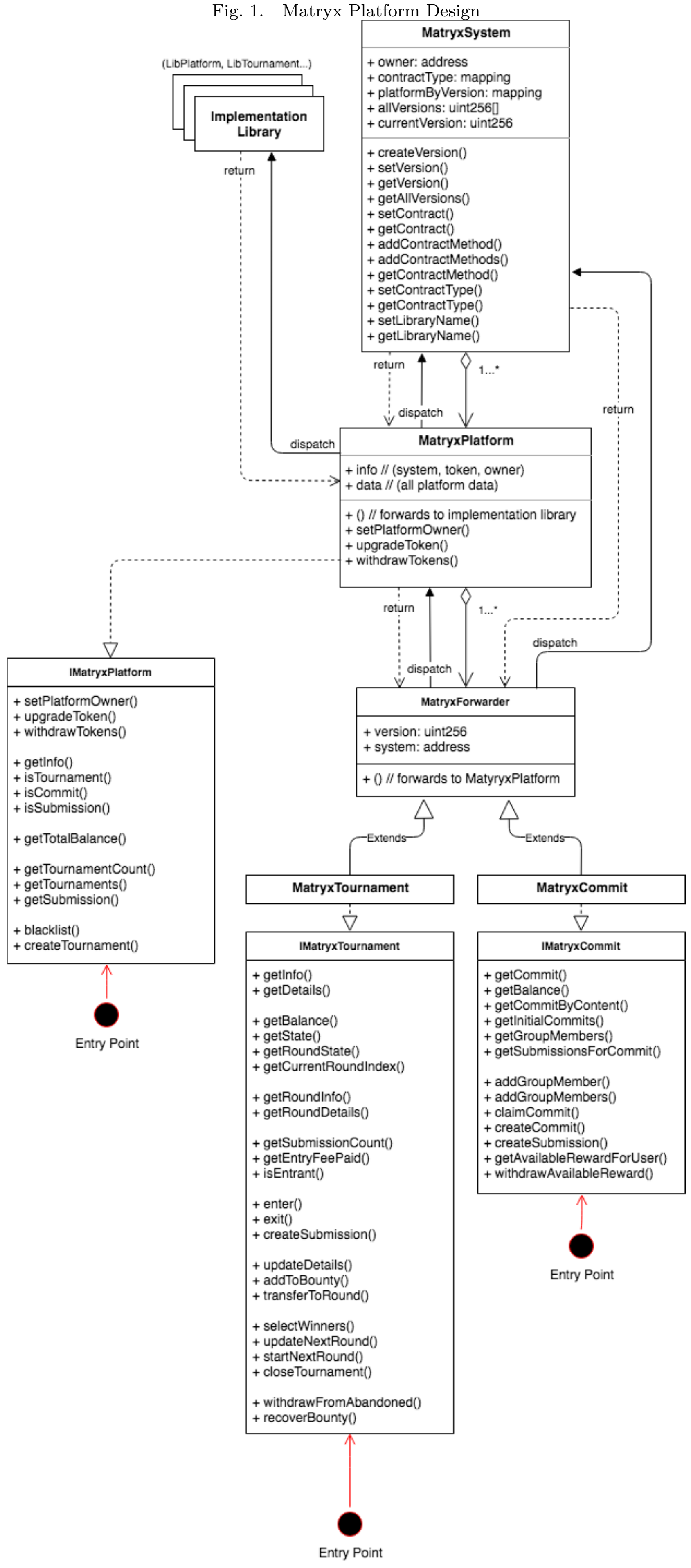
[4] Elizabeth Wager. "Why we should worry less about predatory publishers and more about the quality of research and training at our academic institutions". In: 27.3 (Mar. 2017), pp. 87–88. URL: `http://www.sciencedirect.com/science/article/pii/S0917504017300217`.

[5] "The EigenTrust Algorithm for Reputation Management in P2P Networks". In: (Nov. 2002). URL: `http://ilpubs.stanford.edu:8090/562/1/2002-56.pdf`.

[6] "EigenTrust++: Attack Resilient Trust Management". In: (Jan. 2012). URL: `https://www.researchgate.net/publication/261093756_EigenTrust_Attack_Resilient_Trust_Management`.

[7] "PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities". In: (Aug. 2004). URL: `https://www.researchgate.net/publication/3297318_PeerTrust_Supporting_Reputation-Based_Trust_for_Peer-to-Peer_Electronic_Communities`.

Fig. 1. Matryx Platform Design


MatryxSystem
+ owner: address
+ contractType: mapping
+ platformByVersion: mapping
+ allVersions: uint256[]
+ currentVersion: uint256
+ createVersion()
+ setVersion()
+ getVersion()
+ getAllVersions()
+ setContract()
+ getContract()
+ addContractMethod()
+ addContractMethods()
+ getContractMethod()
+ setContractType()
+ getContractType()
+ setLibraryName()
+ getLibraryName()
(LibPlatform, LibTournament...)
Implementation Library
return
dispatch
return
dispatch
1...*
return
MatryxPlatform
+ info // (system, token, owner)
+ data // (all platform data)
+ () // forwards to implementation library
+ setPlatformOwner()
+ upgradeToken()
+ withdrawTokens()
return
dispatch
1...*
dispatch
IMatryxPlatform
+ setPlatformOwner()
+ upgradeToken()
+ withdrawTokens()
+ getInfo()
+ isTournament()
+ isCommit()
+ isSubmission()
+ getTotalBalance()
+ getTournamentCount()
+ getTournaments()
+ getSubmission()
+ blacklist()
+ createTournament()
Entry Point
MatryxForwarder
+ version: uint256
+ system: address
+ () // forwards to MatyryxPlatform
Extends
Extends
MatryxTournament
MatryxCommit
IMatryxTournament
+ getInfo()
+ getDetails()
+ getBalance()
+ getState()
+ getRoundState()
+ getCurrentRoundIndex()
+ getRoundInfo()
+ getRoundDetails()
+ getSubmissionCount()
+ getEntryFeePaid()
+ isEntrant()
+ enter()
+ exit()
+ createSubmission()
+ updateDetails()
+ addToBounty()
+ transferToRound()
+ selectWinners()
+ updateNextRound()
+ startNextRound()
+ closeTournament()
+ withdrawFromAbandoned()
+ recoverBounty()
Entry Point
IMatryxCommit
+ getCommit()
+ getBalance()
+ getCommitByContent()
+ getInitialCommits()
+ getGroupMembers()
+ getSubmissionsForCommit()
+ addGroupMember()
+ addGroupMembers()
+ claimCommit()
+ createCommit()
+ createSubmission()
+ getAvailableRewardForUser()
+ withdrawAvailableReward()
Entry Point