Smart Papers: Dynamic Publications on the Blockchain

Michał R. Hoffman, Luis-Daniel Ibáñez, Huw Fryer, and Elena Simperl

University of Southampton, Southampton, UK, SO17 1BJ
[M.R.Hoffman|L.D.Ibanez|H.Fryer|E.Simperl]@southampton.ac.uk

Abstract. Distributed Ledgers (DLs), also known as blockchains, provide decentralised, tamper-free registries of transactions among partners that distrust each other. For the scientific community, DLs have been proposed to decentralise and make more transparent each step of the scientific workflow. For the particular case of dissemination and peer-reviewing, DLs can provide the cornerstone to realise open decentralised publishing systems where social interactions between peers are tamper-free, enabling trustworthy computation of bibliometrics. In this paper, we propose the use of DL-backed *smart contracts* to track a subset of social interactions for scholarly publications in a decentralised and reliable way, yielding *Smart Papers*. We show how our Smart Papers approach complements current models for decentralised publishing, and analyse cost implications.

Keywords: Smart Contracts, Blockchain, Dynamic Publications, Ethereum, Open Decentralised Publishing, Collaborative Processes, Trust

1 Introduction

With the advent of digitisation and Web technologies, dissemination of scientific research objects has become faster and less expensive. However, several authors (e.g. [7,11]) have pointed out that Web-based tools are currently mimicking the print-based format used in the past. The vast potential of the Web to separate dissemination, evaluation and retrieval aspects of publications is currently underused. More focus needs to be placed on the quality assessment aspect of both contributions and contributors, ensuring that proper credit is given to novel ideas and their proponents, and on avoiding the excessive concentration of power in the hands of the publishers and editors.

Conceptual models like Liquid Publications [7] and Dynamic Publication Formats [11] have been proposed to leverage Semantic Web technologies to transform research objects from static to *evolutionary* entities. In these models, authors collaborate on a *living* version of the research object that, upon the authors' agreement, has periodical snapshots or *releases* published on the Web. Releases can be open for comments and reviews from the members of the public, or submitted to Calls for Contributions of conferences or journals. Authoring tools like Dokie.li [6] go one step further and provide *decentralised* implementations of living research objects that allow authors to retain the ownership of, and sovereignty over their data. This supplies an alternative to the current state of play, where scholarly publication processes are centralised in publishing houses and large technology providers. However, an under-explored aspect in these models is how to manage the interactions between authors and contributors of a research object in a trusted way, which is of utmost importance for computing bibliometrics transparently. Examples of these interactions are (i) Agreement between authors on which snapshot of a working version should be released (ii) Agreement between authors on the attribution due to each of them for each release of a living research object (iii) Public comments and reviews of public releases, both as a mean to complement bibliometrics - often overlooked, yet crucial labour in academia. From the point of view of a single scholar that co-authors several papers with different teams, receives reviews and comments from peers, and reviews and comments research made by others, data produced by these interactions, used to measure their performance, is not only controlled by her, or a single third party, but also by many other scholars (or their trustees). Any accidental or malicious change in a data store that is out of her control might have catastrophic impact on her performance measures.

Our work advances several Semantic Web research areas, including trust management for the Semantic Web and decentralised scholarly publication. By proposing a system that uses distributed ledgers and smart contracts to manage trust in a scenario which has been long understood as a critical showcase of semantic technologies, we provide a timely contribution to an ongoing discourse on the role and future of the Web as a (re-decentralised) platform for progress and social good. We aim at answering two research questions in the context of open decentralised publishing systems: RQ1. *How to manage releases and their attribution agreements in a trusted way?*; and RQ2. *How to avoid malicious/accidental modifications in remote data stores affecting the computation of bibliometrics?*

Recently, Distributed Ledger Technologies, commonly known as *Blockchains* [15], have emerged as a novel tool that provides a decentralised solution to the problem of managing transactions of digital assets among parties that do not necessarily trust each other, while guaranteeing the immutability and verifiability of records. Their record-keeping capabilities have been extended to user-defined programs that specify rules governing transactions, a concept known as *smart contracts*. Smart contracts offer guarantees of *security*, *tamper-resistance* and *absence of central control*.

In this paper, we introduce a system called *Smart Papers* to manage the attributions and annotations of scholar publications, filling the gap of existing open decentralised publishing models. In our approach, a suite of four smart contracts is deployed on top of the Ethereum platform, and reusability is achieved by by an unbounded number of research objects calling those contracts, and storing publication metadata in a distributed ledger. The smart contracts take the place of a trusted third party in keeping records, with the critical difference being that of data and execution not being controlled by a single entity, but rather inheriting all the guarantees of the host blockchain platform. In our Motivating Example (Section 2), we highlight some of the most critical problems with current models - issues that directly affect the quality and trustworthiness of the existing approaches. In Section 3, we survey the existing models and implementations concerned with scientific authoring. We subsequently propose the Smart Papers model (Section 4) and its implementation in Ethereum, paying particular attention to the issues of trust, identity, and platform technological considerations. Our discussion then progresses to cost analysis (Section 5), after which our conclusions and future work recommendations are presented, in the final section of this paper.

2 Motivating Example

Bob and Alice are scholars from two separate institutions, who agree to collaborate on a publication. They begin by employing their collaborative authoring tool of choice to start a working version of their paper. After a few weeks of work, they decide to release a public version to receive open comments and reviews. Charlie is a scholar from a third institution that finds Bob and Alice's release through an aggregator or a search engine. He reads the article and leaves some comments on it that are stored in his personal data store and linked to the release, for instance, using the Web Annotation ontology[1].

Bob and Alice integrate Charlie's comments in their working version. They continue their work and eventually publish a second release. This time round, they submit it to the Call for Contributions of a conference that uses open reviewing. Diane is one of the assigned reviewers. Her review is linked to the release which she read, as stored in the conference's data store

When it is finally time for Bob, Diane and Charlie's appraisal meeting, their employers ask them for the dynamic publications that they have been involved in. Bob shows the full sequence of releases of the publication, while Charlie shows the comment he made on Bob and Alice's paper, and Diane shows the review she made for the conference. Employers apply their preferred credit models to assign weights to each type of attribution described in the attribution metadata, and quantify their values.

However, when reputation, credit, and ultimately, jobs are involved, social interactions can go wrong, with people trying to game the system in their favour, or to disfavour others. Below, we outline some examples of when things can go awry:

Example 1. Alice trusts Bob for creating the releases and their attribution metadata, as Bob controls the data store. However, Bob can publish a release with the metadata giving more attribution to himself. If using a Trusty URI mechanism, once the release is picked up by other agents, it is very hard to overwrite it. In a decentralised authoring tool like Dokie.li, each author would

[1] https://www.w3.org/TR/annotation-vocab/

hold a copy of the working version, and they could independently generate the release, but if the attribution metadata differs between them, who solves this disagreement? How does an external agent know which copy to trust?

Example 2. Bob and Alice could collude to show different versions of the attribution metadata. For example, consider that employers use two different services to query dynamic publications linked to their faculty members. It is not hard to imagine a semantic store that returns a different version of the attribution metadata, depending on which agent is asking.

Example 3. Bob and Alice could collude to ignore Charlie's comment, in an attempt to not share part of the credit with him. In a decentralised model, a link to the comment and Charlie's identity should be stored in Bob and Alice's data store; however, if Bob and Alice control the data store, nothing prevents them from deleting the link. Charlie would have the copy of the comment and the link to the release, but he might have a hard time convincing a third party (his employer for example), that the comment was not forged.

Example 4. If Diane's review is considered unfair, the editors in control of the data store of the conference might be tempted to make it disappear. A third party agent querying the conference's data store would see nothing. An agent following links from Bob and Alice's data store would get a dereference failure (404). Even if Bob and Alice kept a copy of the review and a Trusty URI, how can they prove that they are not forging a review to damage Diane's reputation?

The common problem of these scenarios is that for all actors (Alice, Bob, Charlie, Diane and their employers), data that is crucial to show or measure performance is not entirely under their control, making it vulnerable to manipulation. Our approach addresses this problem by empowering all collaborators with the following:

- The notarisation of releases providing evidence that all the authors agreed to releasing a particular version of their paper.
- The notarisation of the attribution metadata linked to a release, ensuring that all authors have agreed on it, and guaranteeing to third parties that none of them can tamper with it.
- A mechanism that ensures that annotations made on releases by agents other than authors cannot be repudiated by annotators or their recipients, guaranteeing to both authors and third parties querying this data, that it was not tampered with.
- An index of links and data concerning a particular dynamic publication. This potentially facilitates the task of Web agents that compute bibliometrics, as there is no need to either trust the data store of the authors, or to crawl the Web in search of the comments and reviews to the publication.

3 Related Work

Several models have been proposed to take advantage of digital and Web tools to improve the way academic publications are produced and managed. Liquid

Publications [7] proposes evolutionary, collaborative, and composable scientific contributions, based on a parallel between scientific knowledge artefacts and software artefacts, and leveraging lessons learned in collaborative software development. Their model is based on the interaction between Social Knowledge Objects, *i.e.*, digital counterparts of the traditional paper unit; people and roles, *i.e.*, agents involved in the scientific knowledge processes, playing various cooperating and competing roles (from traditional ones, like author, reviewer or publisher, to new ones derived from the model itself, like classifiers, quality certifiers, credit certifiers); and processes to manage its lifecycle, namely: authoring collaboration, access control, IPR and legal aspects, quality control and credit attribution and computation. The Living Document model [9] aims at creating documents that 'live' on the Web by allowing them to interact with other papers and resources. It lets authors build social networks, with their interactions defined through the papers they write. Heller et al. [11] propose Dynamic Format Publications, where working versions are collaboratively edited by a small group of authors, that decide when a version or revision become widely available, following a formalised gate-keeping mechanism (e.g., consent among authors and/or peer-review). Only the Living Document approach provided a prototype (inactive at this time), and none of them discusses the security and trust implications of their models. Our work provides a foundation that can be used to track and manage credit attribution (and by extension, IPR and legal aspects) that can be easily plugged into a broader authoring model.

Concerning the decentralisation of scholarly communication, Dokie.li [6] is a fully decentralised, browser-based authoring and annotation platform with built-in support for social interactions, through which people retain the ownership of and sovereignty over their data. Dokie.li implements most of the functionalities described in the previously described conceptual models in a decoupled way. In a nutshell, a Dokie.li document is an HTML5 document enriched with RDFA, which is stored in the author's personal data store. The Linked Data Platform (LDP) protocol implementation enables the creation, update and deletion of documents. Interactions with documents are registered using the Web Annotations vocabulary. Documents are connected statically through links and dynamically through Linked Data Notifications [5], proving the viability of a decentralised authoring and annotation environment built according to Web standards. Authors consider that in a fully decentralised setting, each source is filterless and responsible for its own quality and reputation, whilst everyone is free to selectively distrust certain sources using any mechanism they desire. We argue that, although this assumption holds for trust in the *content* of the research object, stronger measures are needed for social interaction data on research objects that could be used to compute bibliometrics. Our approach also aims at solving some security issues that arise in decentralised environments, notably, the possibility of malicious deleting or updating of records to impact bibliometrics[14].

With respect to the application of blockchains for scholar processes, the Blockchain for Science association maintains a living document [3] that collects and proposes applications, use cases, visions and ventures that use blockchains for science and knowledge creation, providing an index of the potential impact of Distributed Ledger Technologies in all stages of the research workflow. For the particular case of publishing and archiving, timestamping and credit attribution of Dynamic Publications is mentioned as a promising use case. To the best of our knowledge, the open-source system that comes closest to ours is Manubot[2], a tool for writing scholarly manuscripts via GitHub. Manubot automates citations and references, versions manuscripts using git, and enables collaborative writing via GitHub. Data from Git related to commitment and authorship can be used to establish attribution. An innovation introduced by Manubot's authors [3] is the timestamping of manuscript versions on the Bitcoin blockchain, to prove the existence of the manuscript at a given point of time in a decentralised way. Our approach generalises Manubot's idea to further social interactions around publications.

Concerning the permanence and immutability of Web artefacts, Trusty URIs [13] propose to append to URIs the cryptographic hash of the Web artefact they represent, enabling the verification contain the content the URI is supposed to represent. Trusty URIs are immutable in the sense that any change in an artefact would change its URI as well, and permanent, under the assumption that Web archives and search engines that crawl them are permanent. Our approach implements functionality analogous to Trusty URIs, but also solves the further problem of conflicting metadata: if each author could publish metadata on the attribution about the research object, each one with its own Trusty URI, and both can be verified to not have been tampered with, then which one should an external agent use?

4 The Smart Papers Model

The Motivating Example (Section 2) illustrated the importance of trust management throughout the collaborative process. When reviewing related work (Section 3), we highlighted a strong need for making agreements and setting their outcomes in stone so that they cannot be later repudiated. Furthermore, all the essential artefacts associated with those agreements must be timestamped and securely stored in a truly permanent way. Currently available collaborative tools solve some trust issues, for example Dokie.li removes centralisation so that the authoring parties do not have to rely on an intermediary to publish and annotate their documents. This is a very welcome step towards removing the overhead associated with middleman activities (publishing house), albeit it merely shifts the trust towards the authoring parties (author, reviewer). It is easy to imagine a situation in which the authors destroy their data, the reviewers could do the same, and any track of their writing will be lost forever.

The purpose of our model is to provide trust where it has not existed before. Smart Papers provide a collaborative platform that preserves a single version of the truth throughout the collaborative process. This is somehow similar to

[2] https://github.com/greenelab/manubot-rootstock

[3] https://github.com/greenelab/deep-review/pull/274

employing a trusted third party (e.g. a notary public) to keep track of contracts signed by multiple parties, alongside with all the certified photocopies of all the evidence attached to the contracts as relevant appendices. An example of such notarised contract would be Alice and Bob signing an agreement specifying the ordering of their names on a paper (e.g. 'Bob, Alice') and then attaching a certified photocopy of their paper in its current version as an appendix. We use smart contracts for maintaining all such signed agreements in order to implement Smart Papers. Table 1 summarises how smart contracts can provide the functionality analogous to that of a traditional trusted third party.

Table 1. Blockchain smart contracts as compared to a traditional trusted third party

Notary public function	Blockchain function
Authenticate parties using their legal identification	Identify parties cryptographically
Take statutory declarations, store them and certify photocopies	Store data permanently and securely and provide real time access
Prepare and certify contractual instruments	Store and execute smart contracts
Provide a trusted record for the above	Provide a trusted record for the above

4.1 Design

To design the Smart Papers model, we shall assume that all authors successfully identify through their ORCID (Open Researcher and Contributor ID [10]) which is a non-proprietary alphanumeric code to uniquely identify scientific and other academic authors and contributors. ORCIDs are mapped to authors' signing and encryption keys using a smart contract. The main functionality for our model is then designed using the separation of concerns (SoC) design principle [12], such that each contract file addresses a different concern, i.e. a different set of information that jointly affects the global state for the Smart Papers use case. We use UML to model the main classes corresponding to our smart contracts. It is important to note that smart contracts and OOP classes (as modelled by the UML) are not quite the same. The semantics are very similar in many cases, but some fundamental differences arise from the fact that smart contracts can store and send value and have a public address once deployed [16]. The UML diagram in Figure 1 shows how we group these concerns into the following four categories: Paper, Version, Annotation and Contributor.

To begin with, an article and its metadata (e.g., attribution encoded in the ScoRO ontology[4]) is submitted by a writer (we shall refer to her as Alice, from

[4] http://www.sparontologies.net/ontologies/scoro

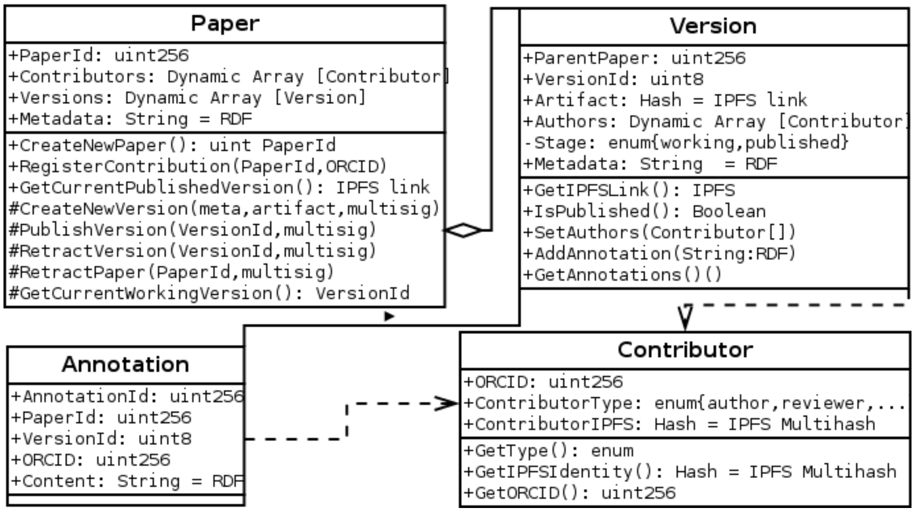
Paper
+PaperId: uint256
+Contributors: Dynamic Array [Contributor]
+Versions: Dynamic Array [Version]
+Metadata: String = RDF
+CreateNewPaper(): uint PaperId
+RegisterContribution(PaperId,ORCID)
+GetCurrentPublishedVersion(): IPFS link
#CreateNewVersion(meta,artifact,multisig)
#PublishVersion(VersionId,multisig)
#RetractVersion(VersionId,multisig)
#RetractPaper(PaperId,multisig)
#GetCurrentWorkingVersion(): VersionId
Version
+ParentPaper: uint256
+VersionId: uint8
+Artifact: Hash = IPFS link
+Authors: Dynamic Array [Contributor]
-Stage: enum{working,published}
+Metadata: String = RDF
+GetIPFSLink(): IPFS
+IsPublished(): Boolean
+SetAuthors(Contributor[])
+AddAnnotation(String:RDF)
+GetAnnotations()()
Annotation
+AnnotationId: uint256
+PaperId: uint256
+VersionId: uint8
+ORCID: uint256
+Content: String = RDF
Contributor
+ORCID: uint256
+ContributorType: enum{author,reviewer,...
+ContributorIPFS: Hash = IPFS Multihash
+GetType(): enum
+GetIPFSIdentity(): Hash = IPFS Multihash
+GetORCID(): uint256

Fig. 1. The Smart Papers distributed application design

our motivating example earlier), and stored in a distributed file store, all of which is recorded on the blockchain. Alice will have been set up in the system through the use of the *Contributor* smart contract. In our implementation, the *Contributor* contract requires Alice to have a valid ORCID as well as an IPFS node identity belonging to her. The default type for Alice is 'author'. Bob is also set up as an 'author', but Diane uses a different argument for the *Contributor* contract, and so she becomes registered as a 'reviewer'.

Smart contracts often act as state machines, meaning that they have certain stages making them act differently, and in which different functions can be invoked. A function invocation often transitions the contract into the next stage which can be used to model work flows. We use this feature of smart contracts to model the Smart Paper workflow, as seen is Figure 2, which allows the participants to release new versions of their paper and to publish versions when enough authors agree to do so.

Papers can also be retracted. As illustrated in Figure 2, once instantiated, a Smart Paper becomes a dynamic list of versions, each of which can exist in a working state or become published. The number of contributors and their formal ordering is allowed to change on a per-version basis. Annotations can be left by reviewers on published versions.

To create a new Smart Paper, either Alice or Bob call *createNewPaper* in the *Paper* Contract which will return a valid *PaperId* that uniquely identifies their new publication. This also instantiates the workflow with an initial, blank, working version of this paper manufactured by the *Version* contract. Bob and Alice work on their preferred authoring tool to produce a first draft (e.g., to show to a trusted colleague), to register it in the Smart Paper, Bob calls *addNewVersion*

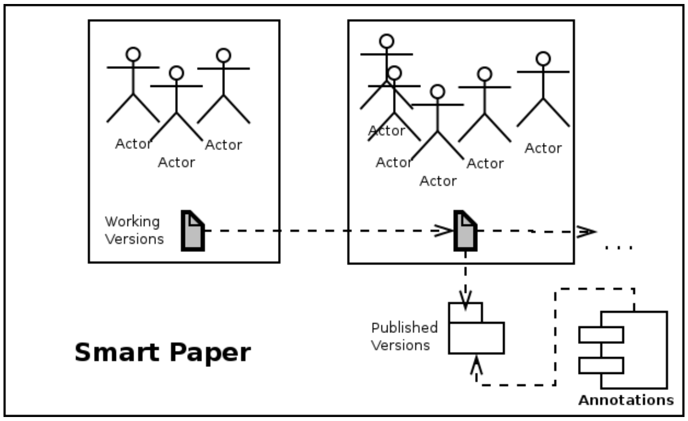

Fig. 2. The workflow of a Smart Paper involves multiple working versions with dynamic collaborators. Versions can become published and made available for annotating.

in the Version contract, including the artefact, its metadata and his signature. Before committing the transaction, the Smart Paper will wait for Alice (marked as contributor of the paper) to also perform a call to *addNewVersion* using the same artefact, metadata and her signature.

The procedure is repeated each time Bob and Alice want to register a new version. For marking a version as public, Bob calls *publishVersion* in the *Paper* contract, providing the *versionID* and his signature. Similar to *addNewVersion*, Alice needs to issue her signature through a function call to *publishVersion* before the Smart Paper commits the transaction. The *getCurrentPublishedVersion* and *getCurrentWorkingVersion* return a versionID that can then become the input to the *getIPFSLink*. Up to this point, we have provided a solution for the issues between authors described in Example 1, the Smart Paper only commits a version (including metadata) if all authors sign their agreement to it. An external agent that gets a version from a Smart Paper instance has the assurance that it was approved by all authors, and that the Smart Contract consistently returns the correct version and metadata, solving the issue described in Example 2.

Interactions with external actors like reviewers or annotators, are abstracted as *Annotations*. When Charlie or Diane want to leave their comment or review, they call *addAnnotation* using the *versionID* of the version they want to comment on, and their signatures. Contrary to the *Version* functions, no approval from authors is needed. The annotation is registered in Ethereum's Blockchain and can be retrieved by calling *getAnnotation*. Looking back at Example 3, Charlie can now point to the Smart Paper to show that he made that comment. For the case of Example 4, the Smart Paper holds a register of the reviews. Alice and Bob can now prove that the annotation held by the Smart Paper was signed by Diane.

4.2 Implementation

Although in theory, the Smart Papers model could be implemented on any smart contract-enabled platform, the choice of the implementation framework dramatically impacts development time and costs. Whilst there are multiple distributed ledger technologies, such as Corda[5] or HyperLedger[6], that could be utilised to develop trusted smart contract code that runs on top of the blockchain, for this paper, we elect to develop on top of the Ethereum platform [16] which is the most commonly used technology of its kind [2]. We defer the feasibility and cost of development in other platforms for future analysis.

Background: Ethereum and IPFS Ethereum is an open-source, public, blockchain-based distributed computing platform featuring smart contract functionality [16]. It plays the role of the trusted third party for all Smart Papers agreements in our model. Ethereum blockchain was designed to be deterministic. This means, that everyone should always end up with the same, correct state, if they try to replay the history of Ethereum transactions. In Ethereum, the code execution layer is provided by the Ethereum Virtual Machine (EVM), a Turing complete 256bit VM that allows anyone to execute code that references and stores blockchain data in a trust-less environment. Every contract on the Ethereum blockchain has its own storage which only it can write to; this is known as the contracts state and it can be seen as a flexible database albeit at a high cost. When deployed, Ethereum contracts get an *address*, that can be considered similar to an URI in Ethereum's namespace. Using this address, a client can call functions defined in a smart contract, in a similar fashion to a web service.

When implementing our model, we chose to store all the artefacts using IPFS [4]. The InterPlanetary File System is used for efficiently distributing and referencing hash-linked data in a way that is not centralised and does not necessarily involve blockchain transactions, thus avoiding the economic penalties associated with on-chain storage. In many ways, IPFS is similar to the World Wide Web, but it could be also seen as a single BitTorrent swarm for exchanging objects. Furthermore, the IPFS specification contains a special *commit* object which represents a particular snapshot in the version history of a file. This allows us to reference resources in an immutable way, akin to Trusty URI functionality. Using IPFS we can, therefore, limit the role of Ethereum, so that it only deals with the application logic; the data layer is provided by the InterPlanetary (IPFS) stack, and the two layers are integrated via hash references.

Reaching agreements One of the core requirements of the SmartPaper model is the ability to provide a tool for all collaborators to agree with the result of a certain interaction. Decision making can be implemented in different ways. In our implementation, the number of collaborators can be unbounded, but

[5] https://github.com/corda/corda

[6] https://www.hyperledger.org/

to make a decision, an agreement needs to be reached by all authors. We use the multiple signature scheme to enable the authors to jointly perform specific actions on a Smart Paper. The following Ethereum code snippet illustrates how multiple authors' signatures are verified by the *PublishVersion()* functionality of the Paper contract before the paper can be published, ensuring they've all agreed before committing to a change in their paper.

```
1   uint public threshold; //quorum needed to decide
2   mapping (address => bool) isCollaborator;
3   function PublishVersion(uint paperId,signature[] sigs){
        require(checkSignatures(sigs));
4       //Publishing code follows:...
5       }
6   function checkSignatures(signature[] signatures){
7      if (signatures.length < threshold) throw;
8      for (uint i = 0; i < signatures.length; i++) {
9           r = signatures[i].slice(0, 32)
10          s = signatures[i].slice(32, 64)
11          v = signatures[i].recovery + 27
12          checkSig(v, r, s);
13        }
14   }
15   function checkSig(uint8 sigV, bytes32 sigR, bytes32 sigS) {
16     //ERC191 signature: github.com/ethereum/EIPs/issues/191
17     bytes32 txHash = sha3(byte(0x19), byte(0), this);
18     address recovered = ecrecover(txHash, sigV, sigR, sigS);
19     if (!isCollaborator[recovered]) throw;
20   }
21  //The following code is called upon instantiating new paper
22  function SetUpCollabs(uint threshold_, address[] collabs_) {
23    if (threshold_ > collabs_.length || threshold_==0) throw;
24    for (uint i=0; i<collabs_.length; i++) {
25      isCollaborator[collabs_[i]] = true;
26    }
27    threshold = threshold_;
28  }
```

The *signatures* array acts as an accumulator waiting for enough signatures to be collected according to the threshold. The *PublishVersion* function on line 3 finally becomes triggered by an event (out of the scope of this snippet). The code example also shows how an elliptic curve signature can be parsed for every participant (*sigV, sigR and sigS* arguments) on line 15. To improve on the security of this code, a nonce should be used that is always incremented to prevent replay attacks. We make this pattern reusable, and reference it by all functions that require a quorum for a binding decision to be agreed upon. These functions include *RetractPaper(), PublishVersion(), RetractVersion() and SetContributions().*

5 Discussion

5.1 Identity

Whereas most blockchain applications generally guarantee user anonymity, our use case calls for verifying collaborators' identities. Whilst different digital identity schemes exist, the most popular form seems to be digital certificates used to prove ownership of a public key associated with someone's private key. Even though public-private cryptography can exist in a decentralised environment, digital certificates are always issued by authorised entities.

There exist multiple such authorities which makes it difficult to implement a universal solution. Due to the complexity of this issue, the logic for liaising with different types of digital certificates to verify parties' identities is normally moved to the client's user interface, as it would be too costly to include in smart contracts.

5.2 Cost

Ethereum contracts are not free to execute. Currently, because of the complex nature of the Proof of Work consensus algorithm used by most blockchains including Ethereum, computations performed by blockchain based smart contracts are expensive compared to the same computations performed by a centralised entity.

Execution of a smart contract begins with a transaction that is sent to the blockchain. This transaction specifies the address for the contract, the arguments, and an amount of Ethereum's currency to pay for the execution. It is commonly observed in small-to-medium size contracts that most of the cost is taken up by a fixed 'base fee'. This base fee of 21,000 is expressed in 'gas' which is an abstract unit. While gas is fixed per each transaction, it's additionally fixed for every operation called from within the smart contract. Each low level operation available in the EVM is called an OPCODE. These include operations such as *ADD* - adding two integers together, *BALANCE* - getting the balance of an account, and *CREATE* - creating a new contract with supplied code. Each of these OPCODEs has a fixed amount of gas that it costs to execute. The fixed amount of gas has been chosen by the designers of Ethereum for each OPCODE in a way that reflects the relative complexity of that OPCODE.

Whereas gas is fixed and predictable, the amount a user pays per gas, the *gas price*, is dynamic and dictated by market conditions. The price is usually given in units of *ether*, Ethereum's default currency. Miners receive ether fees based on the amount of gas multiplied by the gas price, which incentivises them to prioritise those transactions that attract higher fees. It also follows that the higher gas price you are willing to pay, the faster your transaction will be processed, and the sooner your contract will be allowed to execute. While offering a high gas price can speed things up, there is a limit to the acceleration. Finally, when discussing cost, it must be mentioned that Ethereum designers have planned mechanisms that will allow the owner of the contract to take all costs upon themselves, thus further incentivising the users of the contract to participate in it.[7]

6 Evaluation

We evaluate the cost of the Smart Papers system by simulating smart contract transactions in a local blockchain environment (Step 1) and then applying the live gas price (Step 2). We focus on the cost of the *Paper* contract functionality[8].

For Step 1, the Ethereum simulator **testrpc**[9] has been used, as it does not require payments for used gas when deploying or testing smart contracts locally. The testrpc utilty is a Node.js client that uses the **ethereumjs**[10] JavaScript library to simulate the blockchain ecosystem behavior and make developing Ethereum dapps (distributed applications) faster. For estimating gas consumption, we use the Web3.js library[11] is the Ethereum compatible JavaScript API that implements JSON remote procedure calls. After contract creation, we use the **estimateGas** call provided by Web3 to estimate the gas amount required to pay for our smart contracts' functions. We arrived at ˜75,000 gas per typical Smart Paper transaction.

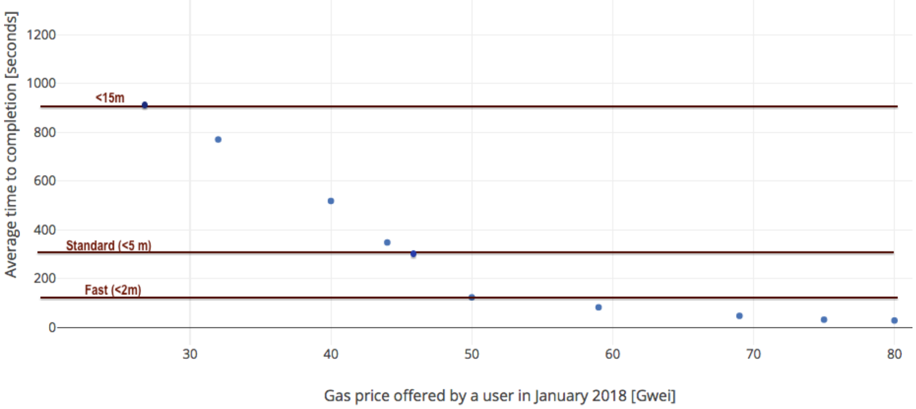

Fig. 3. Average wait times for Ethereum code execution on 11 January 2018

For Step 2, the gas price can be found with ETH Gas Station [1], the defacto reference for understanding the current gas market conditions and miners' current policies. The "Recommended User Gas Prices" section of ETH Gas Station shows the range of gas prices you might pay for an expected transaction

[7] https://blog.ethereum.org/2015/12/24/understanding-serenity-part-i-abstraction/
[8] https://github.com/mikehoff/SmartPapers
[9] https://github.com/trufflesuite/ganache-cli
[10] https://github.com/ethereumjs
[11] https://github.com/ethereum/web3.js/

commitment time. Typical time ranges are known as *SafeLow* (<30 minutes), *Standard* (<5 min) and *Fast* (<2m). Figure 3 illustrates this relationship for the 11th January 2018. Assuming the *Fast* (<2m) confirmation time, our graph suggests this would cost us 50 gwei per gas on this day (*gwei*, also known as *shannon*, is one billionth of one *ether*). On the same day, if we were happy to wait up to 15 minutes for a confirmation, gas price would have gone down to 26 gwei.

To put Steps 1 and 2 together, we use the following formula:

$$contractCost := baseFee + (gasUsed \times gasPrice)$$

which yields transaction cost in the *Fast* range to be around 3,750,000 *gwei*, i.e. *0.00375* ether per Smart Paper transaction. If we are happy to wait a bit longer (15 minutes), this goes down to 1,950,000 gwei.

This translates to a sterling cost of *£1.7 ($2.3)* per Smart Paper transaction such as publishing or retracting a paper if we want this transaction to be accepted in 15 minutes. Assuming that contributors have access to IPFS nodes, there is no extra cost, in terms of gas, associated with storing of the binary artefacts.

7 Conclusions and Future Work

There is an incentive to use blockchain technology for collaborative processes because it is inherently trustworthy. In Smart Papers, we used Ethereum to provide the framework for collaborative authoring, and IPFS for the storage of the papers.

We analysed a use case demonstrating how the nature of scientific publishing would benefit from storing agreements and artefacts in a verifiable distributed database that does not reside within the confines of a single point of failure, and also does not rely on a centralised party to provide proofs. We found that Distributed Ledger Technologies, by their design, are appropriate for this use case.

We have conducted initial testing to run simulations using a suite of Ethereum smart contracts that we have developed based on our Smart Paper model and workflow. Future development should be focused on implementing a robust web client, a working version contract, and the annotation functionality.

Further research work is needed to explore how the market conditions for transaction execution may impact our design, and how market volatility could impact user behaviour through the variable nature of gas pricing and transaction completion times. The stability and security of the Ethereum network is currently seeing novel research which needs to be constantly monitored. We would like to further explore the storage options for artefacts, metadata and reviews, to optimise for cost and flexibility.

We believe that distributed ledgers are key to decentralised trust in collaborative processes. In our case, these guarantees can be provided at a level of *£1.7 ($2.3)* per Smart Paper transaction. Future work needs to address the cost of more frequent operations like comments. We would also like to explore the mapping of the Smart Papers workflow to a relevant ontology (for example PWO[8]) to allow each paper to be traced in a semantically standardised way.

References

1. ETH Gas Station (accessed January 2018), https://ethgasstation.info/FAQpage.php
2. Alharby, M., van Moorsel, A.: Blockchain-based smart contracts: A systematic mapping study. CoRR abs/1710.06372 (2017), http://arxiv.org/abs/1710.06372
3. Bartling, S.: Blockchain for Open Science and Knowledge Creation. Tech. rep., Blockchain for Science, http://www.blockchainforscience.com/2017/02/23/blockchain-for-open-science-the-living-document/
4. Benet, J.: Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561 (2014)
5. Capadisli, S., Guy, A., Lange, C., Auer, S., Sambra, A., Berners-Lee, T.: Linked Data Notifications: A Resource-Centric Communication Protocol. In: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (eds.) The Semantic Web, vol. 10249, pp. 537–553. Springer International Publishing, Cham (2017)
6. Capadisli, S., Guy, A., Verborgh, R., Lange, C., Auer, S., Berners-Lee, T.: Decentralised Authoring, Annotations and Notifications for a Read-Write Web with dokieli. In: Web Engineering. pp. 469–481. Lecture Notes in Computer Science, Springer, Cham (Jun 2017)
7. Casati, F., Giunchiglia, F., Marchese, M.: Liquid Publications: Scientific Publications meet the Web. Technical Report 1313, University of Trento (2007)
8. Gangemi, A., Peroni, S., Shotton, D., Vitali, F.: The publishing workflow ontology (pwo). Semantic Web 8(5), 703–718 (2017)
9. Garcia-Castro, A., Labarga, A., Garcia, L., Giraldo, O., Montaña, C., Bateman, J.A.: Semantic Web and Social Web heading towards Living Documents in the Life Sciences. Web Semantics: Science, Services and Agents on the World Wide Web 8(2-3), 155–162 (Jul 2010)
10. Haak, L.L., Fenner, M., Paglione, L., Pentz, E., Ratner, H.: Orcid: a system to uniquely identify researchers. Learned Publishing 25(4), 259–264 (2012)
11. Heller, L., The, R., Bartling, S.: Dynamic Publication Formats and Collaborative Authoring. In: Bartling, S., Friesike, S. (eds.) Opening Science, pp. 191–211. Springer International Publishing, Cham (2014)
12. Hürsch, W.L., Lopes, C.V.: Separation of concerns. Tech. rep., NorthEastern University (1995)
13. Kuhn, T., Dumontier, M.: Making Digital Artifacts on the Web Verifiable and Reliable. IEEE Transactions on Knowledge and Data Engineering 27(9), 2390–2400 (Sep 2015)
14. López-Cózar, E.D., Robinson-Garcia, N., Torres-Salinas, D.: Manipulating google scholar citations and google scholar metrics: Simple, easy and tempting. arXiv preprint arXiv:1212.0638 (2012)
15. Wattenhofer, R.: The science of the blockchain. Inverted Forest Publishing, Erscheinungsort nicht ermittelbar, first edition edn. (2016), oCLC: 952079386
16. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper 151 (2014), https://github.com/ethereum/yellowpaper