SmartProvenance: A Distributed, Blockchain Based Data Provenance System

Aravind Ramachandran
The University of Texas At Dallas
Richardson, Texas
axr156530@utdallas.edu

Murat Kantarcioglu
The University of Texas At Dallas
Richardson, Texas
muratk@utdallas.edu

ABSTRACT

Blockchain technology has evolved from being an immutable ledger of transactions for cryptocurrencies to a programmable interactive environment for building distributed reliable applications. Although the blockchain technology has been used to address various challenges, to our knowledge none of the previous work focused on using Blockchain to develop a secure and immutable scientific data provenance management framework that automatically verifies the provenance records. In this work, we leverage Blockchain as a platform to facilitate trustworthy data provenance collection, verification, and management. The developed system utilizes smart contracts and open provenance model (OPM) to record immutable data trails. We show that our proposed framework can securely capture and validate provenance data that prevents any malicious modification to the captured data as long as the majority of the participants are honest.

KEYWORDS

Distributed systems, Knowledge Management, Data Provenance, Blockchain platform

ACM Reference Format:
Aravind Ramachandran and Murat Kantarcioglu. 2018. SmartProvenance: A Distributed, Blockchain Based Data Provenance System. In *CODASPY '18: Eighth ACM Conference on Data and Application Security and Privacy, March 19–21, 2018, Tempe, AZ, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3176258.3176333

1 INTRODUCTION

As the data used for research increases exponentially, ensuring information quality and preventing data manipulation has emerged as an important factor affecting the research results. For example, an audit conducted by the Cancer and Leukemia Group B, one of the multi-center cancer clinical trial groups sponsored by the National Cancer Institute found an incidence of fraud of 0.25 percentage of the trials conducted [11].

To avoid data frauds such as data fabrication, under-reporting and falsifying the results to match research objectives in critical research, the provenance of the data has to be maintained. In this context, data provenance is a meta-data that describes where the data of interest originated, who owns the data and what were the transformations that were done on the data. Data provenance facilitates the integration of data from diverse sources as well as providing information of these sources. Also, it acts as a yardstick for measuring how far the results of the experiments support the actual objectives of the research. This results in increased transparency and trustworthiness of the research. For example, in [16], authors highlight the increase in transparency and trustworthiness of research results due to data provenance tracking. Therefore, provenance details of the data must be recorded from its generation to the transformations to the productions of results. In section 7, we discuss a real-world setting where the provenance of data is crucial to prevent fraud.

Main challenges for a provenance system are secure collection and storage, verifiability and preserving the privacy of the collected provenance data. Data used in any form of research may come from a myriad of sources and may contain sensitive information such as patient records. A data provenance management system should ensure that the data is protected against unauthorized access, and privacy violations.

Due to the importance of collecting provenance information, systems such as Chimera [28] and myGrid [6] have been developed to store and process provenance information. Many of the existing provenance systems are based on a centralized storage model. The downside to the centralized system architecture is that if the central server is compromised, the whole data provenance trails could be compromised. In provenance systems based on distributed architecture, the security of the data provenance information is another area of contention. Any authorized users can corrupt the data stored in the provenance system. To our knowledge, the current provenance systems do not try to validate the changes before they are stored. Our proposed *SmartProvenance* addresses these issues by using Blockchain as a medium for storing provenance information and providing validations for each of the changes before logging the changes using smart contracts. Due to the immutable nature of the Blockchain environment, the approved provenance changes that are logged cannot be modified by any users once they are stored. In *SmartProvenance*, due to the distributed nature of the Blockchain, the data provenance trails are replicated on every node of the blockchain ensuring high availability and fault tolerance.

1.1 Overview of Our Contributions

To address the above-mentioned challenges and requirements, we propose a system, *SmartProvenance*, to securely capture scientific provenance data. *SmartProvenance* provides a platform, built using

ACM ISBN 978-1-4503-5632-9/18/03…$15.00
https://doi.org/10.1145/3176258.3176333

autonomous programs called smart contracts in the blockchain, for automated generation and verification of provenance data. The proposed system implements techniques such as secure storage of data trails, access control policies, voting mechanism, and penalty payments to ensure that no malicious changes are made to the provenance trails. The *SmartProvenance* system provides customized verification scripts for users to determine whether the submitted changes are valid or not.

We have implemented a *SmartProvenance* system on top of the Ethereum blockchain [5] along with Meteor framework [24] for developing interfaces for the user's client module. We evaluate the system in the real-world scenario of clinical drug trials. The results show that *SmartProvenance* system captures data provenance with fixed cost and moderate overhead.

The paper is structured as follows: Section 2 describes the system model. Section 3 discusses the system architecture and provenance life cycle. In Section 4, we take a detailed look at the various components of the system and their functionalities. Section 5 describes the voting process implemented for verification of data modifications. In Section 6, we analyze the security and privacy parameters of the system. Section 7 details the results obtained by implementing *SmartProvenance* in real-world environments. In Section 8 we compare *SmartProvenance* with other related blockchain based systems. We conclude the paper with Section 9.

2 BACKGROUND

In this section, we discuss some of the tools used by our system and our threat model assumptions.

2.1 Ethereum

SmartProvenance is built on top of the Ethereum [30], a distributed public blockchain. Ethereum is a worldwide network of interconnected computers that execute and validate programs. Ethereum provides a decentralized Turing-complete platform called Ethereum virtual machines to run application codes called *smart contracts*. Smart contracts are codes that reside within the Ethereum blockchain environment that execute when specific conditions are met. Each unique entity (user or smart contract) in the network is identified by a unique public key known as an address. As the smart contracts reside on top of the blockchain, each execution of a smart contract is also recorded in the blockchain. A smart contract has two types of data storage: state storage which stores data of the variables in the smart contract and the event logs. Events are notification mechanisms in the smart contract that allow it to trigger some external functionalities. Event logs are as the name suggests an immutable record of the sequence of events that are emitted by a smart contract. A smart contract is called or triggered through transactions. A transaction can be viewed as a message that is sent between addresses in the network. Transactions may not involve value exchange. Ethereum also provides a currency called *ether* that is used to implement value exchange between parties in the platform. In the Ethereum blockchain platform, each computational step has a cost associated with it called *gas* [30]. To execute each transaction, the initiator of the transaction has to pay the corresponding gas price for each step executed in the transaction. At the time of writing this paper 1 gas is equal to 0.00002 ethers.

2.2 Provenance Model

The *SmartProvenance* system represents the data provenance trails using Open Provenance Model (OPM) [26]. In the OPM methodology, each action of the current system is represented using three parameters: 1) artifact (e.g., documents, files) before and after change versions, 2) an agent which represents the initiator of the change and, 3) the process which is the process that changes the artifact from the previous version to the current version. In our project, we represent the OPM model as a triple describing what the agents, artifacts, and process are, and also number code relationship edges between them. For example, the action of modifying a file can be represented in OPM as a tuple *(user, file: old version, file: new version, the process used for modifications).*

2.3 Threat Model

The *SmartProvenance* system can have two types of attackers: an external adversary and an internal adversary. An external adversary is a user who does not have access to the document/data in the system, but will actively try to corrupt the data provenance trails of a particular private document/data. An internal adversary has access to the document/data granted by the owner in the *SmartProvenance* system. The internal adversary is able to change the document and log the changes as provenance trails on the blockchain. The internal adversary tries to corrupt the provenance chain by logging incorrect details to the chain. For this work, we assume that the external adversary does not know the key to decrypt the document nor does he have access to the location in which the document is stored. The adversary only has knowledge of the document id and uses this information to mount an attack on the system. For the case of an internal adversary, we assume that he is not the owner of a document who can grant access to the document. The *SmartProvenance* guarantees truthful behavior if at least half of the users that can access the documents and associated provenance data are honest [13]. Finally, we assume that the cloud storage is not trustworthy and all the files version stored are encrypted through a shared key using a symmetric key encryption. We also assume that there exists a secure external key sharing platform through which the owner of the document can share the keys.

3 SYSTEM OVERVIEW

We consider a scientific setting where researchers keep their research records as a document stored in the cloud. The document (e.g., any data file) is encrypted by the owner of the document (e.g., the lead researcher). The owner of the research document provides access to the document to users by providing the key. For a user to log the provenance information in the *SmartProvenance* system, the owner of a document needs to grant access to the unique document log to the user. In the *SmartProvenance* system model, the changes to the documents are made through versioning. Each change related to a document is stored as a separate new version. The system assumes that only the latest version of the document/data file is used for modification. The system checks the condition that any document which contains changes not logged in the provenance data is ignored.

The system encourages truthful behavior by penalizing the users who submit wrong change provenance details. The voters are rewarded in the event they find a defective change submitted with a portion of the deposit amount for the change. The users log valid changes to the system using client applications running in each of the individual user's browser. Each of the client applications stores persistent data about the documents that the current user has access to using a database. For the current version of *SmartProvenance*, meteor JS [30] and MongoDB [25] are used to implement the client applications. The client applications communicate with the smart contract through a Geth node which is a program that Ethereum platform uses to communicate with the main blockchain network, running on the client side. The client applications monitor the relevant smart contracts for data change events and initiate verification process. The smart contract module implements functionalities such as access control policies and provenance trail storage.

4 SYSTEM DETAILS

The *SmartProvenance* system consists of two modules. The on-chain module which mainly consists of Ethereum Smart contracts for access control, generating and storing provenance trails and conducting voting process, and the off-chain module which consist of client application that interfaces with the smart contracts to log the changes, provide timers for voting processes and perform the verification of each file change using the cloud-based verification script.

4.1 On-chain module

The Ethereum blockchain provides executable programs, called Smart Contracts, that reside within the blockchain. The Smart Contracts execute only when called and are capable of maintaining state variables. *SmartProvenance* on-chain module mainly consists of two smart contracts which we discuss next.

4.1.1 Document Tracker contract. The Document Tracker smart contract is used to keep track of changes to a given document. Document Tracker contract implements access control policies and restricts user access to the document's provenance trails. The contract also provides methods for provenance trail generation for a particular document. The generated document trails are stored as events in the event log of the Document Tracker contract. Event log storage of data provenance trails is preferred due to cost per storage consideration in the Ethereum blockchain environment [4].

The format of the change event is described in detail in our detailed paper [29]. Each change event also stores the digital signature of the initiator based on the message digest. The Document Tracker restricts access to all document functionalities such as *create* a document for tracking, *grant users rights* to add changes to a particular document provenance history, *revoke users access* rights to a particular document history and finally *generate and store provenance history* of a particular document to the log.

It is *important to note that, SmartProvenance does not store any sensitive information in plain text on the blockchain* because any information stored on the blockchain including the smart contract code is publicly accessible. In addition, due to storage costs and blockchain storage limits, actual data is stored off the blockchain, potentially in a cloud location.

The initial iteration of the provenance history is generated by the owner of a document when that document is added to the system. The contract enforces the constraint that granting access to adding provenance trails for a document is strictly controlled by the owner of the document. In the current implementation of *SmartProvenance*, access rights to a particular document are non-transferable. In addition to the main methods, Document Tracker also consists of helper methods for granting the user access to a document and methods to update the owner of the document. The Document Tracker rejects any unauthorized calls to the functions. Every provenance change event *has to be approved through a voting process by the vote contract.* The data trails are only logged if they are approved by the Vote contract.

We chose the *voting for verifying the submitted provenance information for two reasons*: 1) We want to efficiently prevent malicious changes that obviously violate data use constraints (e.g., not allow the deletion of a patient record from the drug trial data). 2) We do not want the verification process to leak any sensitive information. Unfortunately, verification process could vary in different settings. For example, for drug trials, main verification process could be to make sure that no patient is deleted (e.g., to boost the success rate of the drug) from the data set due to a fatal reaction. Also, if the verification is done in the contract, we need to do this in a way that discloses no information (e.g., using zero-knowledge proofs [20] since contract source code and execution are publicly observable). To our knowledge, the existing zero-knowledge techniques that are efficient are not general enough for all verification scenarios needed for our use case. At the same time, general zero-knowledge verification techniques are not efficient enough to implement for provenance capturing [3]. Due to these reasons, we allow each participant client program to run *the verification code automatically off-the-chain and use on-the-chain contract to vote for or against the change.* Below, we discuss the details of the voting process.

4.1.2 Vote contract. The vote contract implements the voting protocol. The contract implements two types of voting: simple majority voting and threshold voting which we discuss in Section 5. The initiator submits the change in an encrypted form along with his signature and document id to the vote contract. The vote contract receives the change and after verification generates a log event to initiate the voting phase for the change. The voting phase time interval is set as t_1 (t_1 is set to one hour in our experiments) during which the participant can vote on the change. For each vote that is submitted, the vote contract verifies whether the vote is valid for the current voting period. At the end of the voting period, based on the type of voting process, the vote contract rejects/accepts the change based on the voting results. The vote contract restarts the voting phase if the minimum threshold is not satisfied. At the end of the voting phase, if the decision is to accept the change, vote contract submits the change to the Document Tracker contract for generating the provenance event. The vote contract currently accepts only a single outstanding change for a particular document for ensuring the continuity of the data provenance chain and consistency. The current vote contract allows users to log provenance trails if the total count of users of a document is less than three.

4.2 Off-chain module

The off chain module, a JavaScript client, runs on the browser of each of the user machines. The client acts as an interface between the user and back-end smart contracts. The client is responsible for communicating with the smart contract for the storage of the changes, retrieval of the changes and verifying the validity of the changes. The client consists of different components such as the Client Interface module which mainly provides an interface for the user to interact with various functionalities of the on chain smart contracts. The Interface module implicitly generates the digital signature for all the operations that the user performs through the module. The client contains an event watcher component that monitors the vote contact for any change events. If a change is relevant to the current user, the event watch module calls a verification script and verifies the change. The watcher module uses a database to keep track of the documents that are relevant to the current user. In addition to the Javascript clients, *SmartProvenance* also has a verification script running at the cloud storage location, where versions of the documents are stored. The verification script verifies the validity of each change of a document submitted to the Document Tracker. Lastly, the client also has a timer module that is responsible for keeping track of the voting phases. The timer will trigger the termination of the voting process at the end of the voting interval.

4.2.1 Verification script: The verification script resides within the cloud storage of the system. The verification script validates the data file/document changes that are submitted to the *SmartProvenance* system. The input to the verification script includes current and previous cryptographic hash of the document (from the changes submitted to *SmartProvenance*) and the link to the latest version of the file. The verification script first verifies whether the hashes submitted to the files are valid. It then compares the current unconfirmed data file with the last stable version of the file. If any other changes to the file other than the ones mentioned in the change request are identified, the verification script notifies the user of a mismatch. If there are no invalid changes in the file, the verification script confirms the change as valid to the user. Once the changes are verified as valid, to prevent further manipulation of the document version, the verification script restricts the write access to only the owner of the original document. The verification script *can be customized according to the usage scenario* of the *SmartProvenance* system and is developed as a plug-in module. In the current implementation, we have developed the verification script based on Google appscript [14] to support Google Drive Storage. The verification script automates the verification of the changes without relying on a trusted third party. A well-written verification script allows for the secure generation, verification and logging of the provenance trails.

5 VOTING PROCESS

The overall view of the voting process is described in Figure 1. The voting process starts when the initiator submits a change to the Vote contract. The initiator client triggers a timer to initiate the voting phase of the newly submitted change. The vote contract generates an event which indicates the commencement of the voting phase for the submitted change. The Event listener module in the client applications reads the newly generated vote event. The client application verifies if it is a stakeholder in the current document change event. It then calls the verification script residing within the cloud along with the links to the current and previous versions of the file and the file hashes. The call to verification process occurs in every node based on the voting protocol policy. If the verification script returns as true then the client application notifies the user of the result and casts its *vote automatically* on the decision to accept or reject the changes. The vote contract on receiving the vote from a client application records the user decision. [1] The timer module will terminate the vote contract at the end of the voting period. The vote contract counts both for and against votes and rejects the change if the majority have voted against the change. If the change is accepted, the deposit by the initiator is refunded back. If the change is rejected then, the deposit is divided among the participants of the voting phase. This way we incentivize truthful behavior by the participants. If the majority is to accept the vote then the vote contracts submits the changes to the Document Tracker contract for logging.

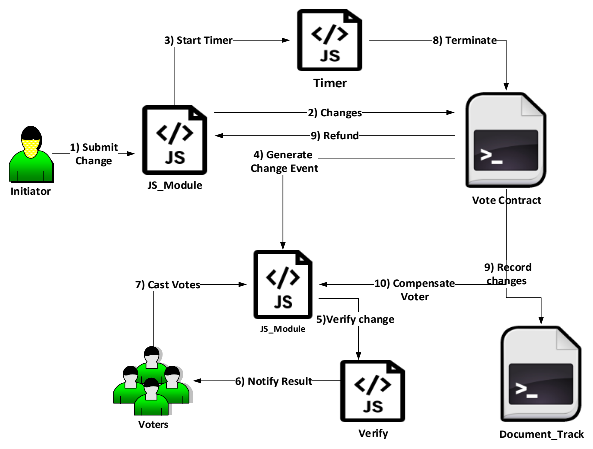

Figure 1: Voting procedure for a document change.

In our current implementation of *SmartProvenance*, we have implemented two types voting protocols: simple majority voting and randomized voting.

5.1 Randomized Threshold Voting

Every client voting for each and every change is not efficient for systems that contain a large number of changes and users. For such scenarios, we propose randomized threshold voting. In randomized threshold voting, the contract requires that a minimum percentage of votes accept or reject the change. Suppose the document has n users, to accept or reject a change, the vote contract threshold is s. To ensure that each voting phase for a change receives s votes, the contract tries to get expectedly t votes for $t \geq s$. The threshold t ensures that the minimum number of votes s is received for each change.

[1]We would like to stress that in our case, this entire voting process is automated using the verification script without any user manual input.

To determine whether to take part in change voting phase, each client generates a random number based on the formula:

$$Ks = Hash(Bno, ETxt, Diff, Glim, Addr) \bmod n$$

In the formula, *Ks* is the random number generated by the client by hashing Bno - the current block number, Etext - the encrypted text in the change event, Diff - the current gas limit and the Addr - the initiator's address. If the generated number is below the threshold number t set by the vote contract (i.e. $Ks < t$), the client votes based on the result of the verification script. Once a vote is submitted, the vote contract generates the random number for each vote in a similar manner and verifies that the submitted vote is legitimate.

In this technique, the voting for the change is based on secure pseudo-random numbers; and it is not feasible to know which clients vote on which changes since the inputs to the hash function differ for each vote almost in a random manner. At the end of a voting period, if the vote contract finds that the total number of votes is below the threshold s, the vote contract restart the voting process. The probability of a restart event can be bounded by choosing appropriate t and s values as described in our detailed paper [29]. If after a predefined maximum number of restarts, the required number of votes are not received, the change is rejected and the deposit is refunded to the initiator of the current change. We can set the system parameters t and s in such a way that this is very unlikely.

6 SYSTEM ANALYSIS

In this section, we analyze the security and privacy aspects of our *SmartProvenance* system. Specifically, we discuss how *SmartProvenance* system handles attacks from the two type of adversaries discussed in Section 2.3.

6.1 Security Analysis

An external adversary can try to attack the current system by submitting an invalid change request for a particular document ID. The *SmartProvenance* contract would stop any such attempts by enforcing access control policies on documents. The Document Tracker contract will only accept change requests from users who have been granted access by the owner of the document. All other change requests are simply rejected by the contract. The Document Tracker also penalizes the external adversary by withholding the deposit amount for the change for the attack attempt. Document Tracker prevents replay attack by keeping track of the latest change timestamp for a particular document. Any message carrying timestamp older than the latest timestamp for that document is ignored.

An internal user can be the owner of the document or one of the users who has been granted access to the document by the owner. An internal adversary who is not the owner of the document can try and corrupt the data provenance trails by submitting defective changes. Since *SmartProvenance* system requires each of the changes to be approved by a minimum number of users, this attack from the internal adversary succeeds only if he can control more than half of the total number of users allowed for the document. The randomized threshold voting further ensures that the adversary cannot know in advance who among all the voters can take part in the voting for a particular change, making it difficult to mount the attack. The internal adversary who is an owner can corrupt the system if he colludes with other stakeholders and votes for the change. The owner is the only user who can grant access, the system can be at a disadvantage if the owner selects a group of users who are loyal to him and corrupts the provenance trail. Although this type of attack may be successful, it still leaves a traceable trail on the blockchain that could be used to detect the attack.

6.2 Privacy Analysis

The privacy protection for the provenance data trail is achieved by the use of hashing and encryption. An external user can infer only the document id and the number of changes that are made to a particular document id by looking at the event logs. Each change event encrypts the payload of the event so that all an external adversary could get is the document id, the ciphertext, and the signature. The link to the cloud location where the actual file is encrypted. The other information that an external user can deduce from watching the contract transaction trails are to see which users are associated with a particular document id. This information is deducible by observing iterations of the voting contract. The Ethereum platform provides anonymization of users through the use of random public addresses. The users of *SmartProvenance* do not reveal their identity in the environment but instead use public addresses to perform operations in the system. In *SmartProvenance*, only the file owner will see the document user. An adversary observing multiple voting iterations could at most deduce the public addresses associated with each document.

In *SmartProvenance* system, each change in the document is represented as a separate record. In the current system, we take each change as a standalone change and do not allow multiple outstanding changes to the same document. This can be restrictive in certain use cases. The system could be modified to accept non-conflicting changes in different parts of the tracked document. The system could accept changes to the document as long as they are non-conflicting, thereby increasing the concurrency. The above modification may involve adding an extra step to the verification script to check for non-conflicting changes.

7 EXPERIMENTAL EVALUATION

To evaluate the *SmartProvenance* system, we test it on two real-life scenarios and calculate the average cost for each of the individual operations of the smart contract. In both scenarios, we find that *SmartProvenance* system performs at a constant cost for individual operations and within a reasonable overhead. We provide the details of one of the use cases below. The details of the other cases can be found in our extended paper [29].

For all use cases, we use the following evaluation setup: the client applications implemented using Meteor JS ran in a laptop (Core i7 2.4GHZ) and a desktop computer (Core i7 3.40GHZ) running Ubuntu 16.04.2 LTS. The smart contracts developed using Solidity language ran on Ethereum Ropsten Testnet[2]. For all scenarios, we simulate the tests for a setting where we have 100 users for each of the document/data file. For the cloud-based storage, we used Google Drive and the verification scripts were developed using Google AppScript.

[2]https://ropsten.etherscan.io/

7.1 Clinical Drug trial

As the first use case, we consider the scenario of a clinical trial [10] of an experimental drug. In phase 3 of the drug trial process, the drug is tested with 300—1000 patients. The objective of the trial is to find the side effects of varying dosages on the patients. The drug trials are conducted by various doctors in various locations and each of the results is recorded in a common document. Each of the experiment group updates the same document every month for a twelve month period. In the research setting, some of the patients may show an adverse reaction to the drug. Researchers with a vested interest may try to remove those records that would show the side effects of the drug, and successive iterations of the same document will be missing records that would adversely affect the trustworthiness of the trails. To avoid the omission of records, the verification process for each change iteration should ensure that the original patient set is maintained. [3]

7.1.1 Add Document. The *Add Document* function is used to add a document to the system for the purpose of maintaining its provenance. The owner of the document could be the head physician who initiates the whole process. The owner generates the initial form of the file that includes the entire initial set of patient details and initial drug dosage and adds it to the contract. The Add Document functions generate a unique document id for each file added. For the drug trials scenario, we need to add only a single file. The average gas cost per file added is 139552. Please note that in our setting, *SmartProvenance* keeps *fixed size provenance records irrespective of the original data file size.*

7.1.2 Add User. The *Add User* function deals with granting access to users for a document. The user who creates a particular document is recorded as the owner of the contract. Access to a particular contract can only be granted by the owner of that contract. Figure 2 gives the gas used per user added for one document where each transaction is the addition of a new user. The average gas used per transactions is 90559. The user details are stored as the hash of the user address. The spikes in the figure 2 represent the difference in the hashing requirements for the inputs.

7.1.3 Initiate Change. The *Initiate Change* function deals with triggering the voting process for logging a particular change. The Initiate Change requires the initiator of the change to deposit an amount with the contract while calling the contract. The Initiate Change function is called in the current scenario at the end of every month by the doctors to record the side effects (if any) of the current dosage. The average gas used for the changes is 731768. Figure 2 gives the gas distribution per initiation of voting phase for different transactions.

7.1.4 Voting phase. Once the change has been initiated, the client programs running in the voting quorum will verify the changes and cast their votes. The vote of each of the participant is recorded by the smart contract and tallied up. The average gas used during this process is 89176. This is due to the initialization that occurs at the start of the voting intervals.

[3]In our experiments, we choose only this constraint for the automatic verification process. Other constraints could be added for different scenarios.

7.1.5 Termination. The result of the voting process determines whether to accept or reject the changes. On rejection of a change, the voters who verified and voted are awarded the deposit amount of the initiator. On acceptance of the change, the change is recorded in the event log of the Document Track contract and the deposit is refunded to the initiator of the change. In Figure 2, we can see that there are two large spikes. These are the cases in which the changes are rejected after the voting process. The gas used for these are more because all the voters are awarded a part of the deposit in the case of a rejection. The average gas used for termination is 249812.

7.1.6 Verification Script: In the drug trial scenario, the verification script verifies if the same set of patients given in the original trials are maintained across the various iterations of the data collection phase. The client initiates the verification script by providing it with the link to the current file version and hashes submitted with the change. The verification script generates its own hashes and compares with the submitted hashes. The script then compares patient identification columns with the previous files to ensure that none of the original patients have been omitted from the currently submitted version. The verification script then notifies the client of the result and votes accordingly. In the drug trials scenario, the verification script checks if all patient records are retained in subsequent iterations. The run times of the verification script which depends on the data file size and the verification complexity, for data files that contain 1000 to 5000 patients, the verification runtime vary from 7 to 31 secs.

7.2 Operation Cost

By observing the system contract executions in the above scenarios, we see that for an individual function such as add user or add document, the gas used per transaction remains almost constant. The cumulative gas used for any individual function is a nearly linear function (e.g., as shown in Figure 2 for vote function).

The gas usage is calculated at an average of 0.00000002 ethers per gas used. At the time of experimentation, a single ether cost is 90 US dollars. The Table 1 shows the average gas used for various operations of the *SmartProvenance* system. As the results indicate most operations can be executed with relatively little cost.

Table 1: Cost of operations in the *SmartProvenance* System.

Operation	Avg gas spent	cost (USD)
Vote	89176.33	0.1605173
Add User	90559	0.1630062
Add Document	139552	0.2511936
Record Change	249812	0.4496616
Initiate Change	731351.5	1.3164327

7.3 Contract Execution Duration

The time taken to perform each of the operations in the system is represented in the Table 2. We can see that all the operations take near constant time to perform. The time taken for each operation is reported as the average time taken per thousand operations.

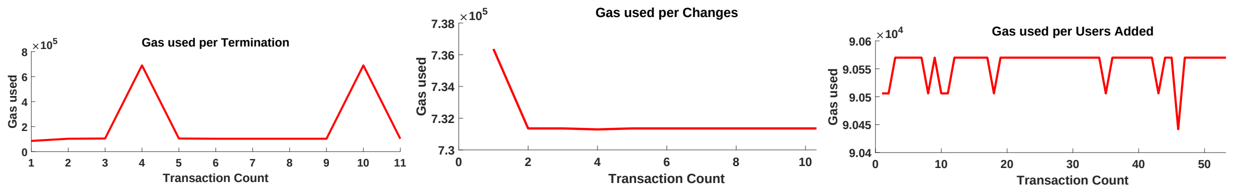

Figure 2: Gas used for various operations.

The execution time of each of the above operations depends on the network speed and the speed of mining of the blocks, but in long run, these times remain near constant and take less than a second in all of the usage scenarios. The experiments show that the *SmartProvenance* generates and stores provenance data in a secure trusted manner with moderate overhead.

Table 2: Time for operations in the *SmartProvenance* System.

Operation	Time Taken (ms)
Vote	829
Initiate Change	858
Add User	877
Add Document	926
Record Change	950

8 RELATED WORK

Recently there have been several research studies that leverage Blockchain as a platform for building trusted systems. Below, we summarize this work and discuss its relationship to our work.

Access control: In [18], authors explain the use of Blockchain as a trans-organizational authentication system. The Medrec system [1] implements access control for medical records across medical institutions through the usage of the public blockchain. Fair access system [27] is a decentralized access control system for the Internet of things devices using blockchain technology. In our *SmartProvenance* system, we also implement access control policies, but our focus is in the capturing of provenance data.

Trusted Authority system: The legal aspect of using blockchain as a verifiable trusted source was further expanded upon by the Common Accords Group [2, 12]. These work describe leveraging data stored in a public blockchain as a verifiable evidence in a court of law. Namecoin [8] system uses the blockchain technology as a trusted source for the Domain Name System (DNS). Our *SmartProvenance* system eliminates the need for storing data on transactions by using the event logs of the smart contract to store the provenance trails. The smart contracts on top of the Ethereum platform act as a decentralized trusted authority regarding all provenance trails stored. The provenance trails generated by *SmartProvenance* is trustworthy as they are verified using the verification policy and stored. *SmartProvenance*, therefore, acts as a decentralized trust-based system for data provenance.

Privacy preserving blockchain systems: The DECENT system [22] uses the blockchain along with the smart contracts to implement key management services. It implements the idea of secret sharing to securely share keys in a public environment. The Hawk system [20] implements the concept of zero-knowledge proofs combined with encryption to implement privacy preserving blockchain systems. The Hawk system uses two components: an on-chain component which uses smart contracts and zero-knowledge proofs to facilitate betting protocols and the off-chain component which generates zero-knowledge proofs for the system. The Hawk system shows how secure computations can be implemented on top of a public system such as the blockchain.

The use of secret sharing techniques for protecting sensitive information is further discussed in [17]. Compared to these works, the *SmartProvenance* system utilizes encryption and hashing to preserve the privacy of the data stored in the public Ethereum blockchain and secure communication channels between the smart contract and client machines to preserve the privacy. For efficiency, verification of the captured provenance data is done off-the-chain.

The common security vulnerabilities in the smart contracts are discussed in [19]. This work illustrates a number of security issues in smart contracts such as call stack bug, block hash bug, and miners withholding the addition of blocks to gain an unfair advantage. This work further discusses how to avoid these pitfalls by including additional access verification and cryptographic primitives like encryption and hashing. Compared to these works, *SmartProvenance* implements digital signatures to avoid malicious logging of provenance data. It uses an encrypted form of the provenance trails to avoid revealing details such as the location of the files and the user access information. *SmartProvenance* further restricts the access to methods based on checks implemented on the user address.

Data provenance: Leveraging blockchain as a data provenance tracker was first discussed by the Project Provenance [23]. In this work, blockchain transactions are used to store provenance details of food products from production to the consumer. In addition, in [15], the use of blockchain as provenance platform is presented as one of the four breakout cases of the blockchain platform. The use of Bitcoin as a data provenance system for research scenario was further explored in [9]. The author suggested the idea of storing the research objectives as an encoded file in the data fields of Bitcoin transactions. Compared to these works, our the *SmartProvenance* system adopts the immutability of the blockchain environment and implements a full stack privacy-preserving, verified data provenance store with access control policies. The provenance chains that are generated by the *SmartProvenance* system are stored as event logs thereby saving costs on storage. The system facilitates the verification of these provenance events by any authorized users. *SmartProvenance* provides a platform to implement custom verification scripts suited for the application area. The system ensures privacy by using public key encryption and preserves integrity by the use of digital signatures.

The ProvChain [21] system provides a data provenance system based on Blockchain technology. The ProvChain system uses monitor programs called "hooks" to track the changes that occur in the cloud storage system and records each and generates events corresponding to the actions of the users. The user events thus recorded are then stored on the blockchain as transactions. The verification process is achieved by an external entity known as the auditor. The auditor generates transaction receipts using Tieron API [7]. The Provchain system verifies the changes after the information is logged on to the blockchain. The *SmartProvenance* differs from Provchain by implementing automated verification scripts and rejecting the invalid changes. The change hash-chain generated by the *SmartProvenance* records only the changes that are verified by the verification script. This guarantees that the changed document is always valid and prevents any chance of collusion between the auditor and the stakeholders. Another major difference compared to Provchain is that *SmartProvenance* implements incentivized voting using smart contracts to penalize the users who try to log invalid changes to the system. The use of randomized voting reduces the centralization of the verification process. Therefore, there is no need for a physical verifier as the verification script verifies the changes before voting on the changes. The advantage of developing verification script is that a verification script for a scenario could be reused by similar applications thereby reducing the cost of development.

9 CONCLUSION

The *SmartProvenance* is a Blockchain based system that provides access control based privacy-preserving data provenance trails. In the *SmartProvenance* system, an authorized user can verify the changes that are made to any data file. It also provides a proof of change with the use of digital signatures and timestamps. The system ensures that the change logs in the blockchain environment are only accessed by the authorized users with appropriate keys. The *SmartProvenance* system further enhances the trustworthiness of the data trails by implementing randomized voting for the captured change trails and any deviation is punished by a monetary penalty using smart contracts. The evaluation of the system based on two real-life scenarios shows that individual operations of the system run with acceptable cost in near constant time.

10 ACKNOWLEDGEMENT

The research reported herein was supported in part by NIH award 1R01HG006844, NSF awards CNS-1111529, CICI-1547324, and IIS-1633331 and ARO award W911NF-17-1-0356.

REFERENCES

[1] Thiago Vieira Andrew Lippman Ariel Ekblaw, Asaf Azaria. 2016. MedRec: Medical Data Management on the Blockchain. (2016). version: 57e013615dbf3f3300152554.

[2] David Bollier. 2015. Reinventing Law for the Commons. (2015). http://www.commonaccord.org/

[3] Zvika Brakerski, Jonathan Katz, Gil Segev, and Arkady Yerukhimovich. 2011. Limits on the Power of Zero-Knowledge Proofs in Cryptographic Constructions. In *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings.* 559–578. https://doi.org/10.1007/978-3-642-19571-6_34

[4] Jonathan Brown. 2015. Storing compressed text in Ethereum transaction logs. (2015). http://jonathanpatrick.me/blog/ethereum-compressed-text

[5] Vitalik Buterin. 2015. A Next-Generation Smart Contract and Decentralized Application Platform. (2015). September.

[6] Tim Clark, Paolo Ciccarese, and Carole A. Goble. 2013. Micropublications: a Semantic Model for Claims, Evidence, Arguments and Annotations in Biomedical Communications. *CoRR* abs/1305.3506 (2013). http://arxiv.org/abs/1305.3506

[7] Tierion coporation. 2017. Tierion API. (2017). https://tierion.com/

[8] Vincent Durham. 2010. *NAMECOIN.* https://namecoin.org/.

[9] The Economist. 2016. Better with bitcoin. (2016). http://www.economist.com/news/science-and-technology/21699099-blockchain-technology-could-improve-reliability-medical-trials-better.

[10] US Food and Drug Administration. 2017. *Clinical Research.* https://www.fda.gov/ForPatients/Approvals/Drugs/ucm405622.htm.

[11] Buyse M George SL. 2015. Data fraud in clinical trials. *PMC* 5, 2 (2015), 161–173. https://doi.org/10.4155/cli

[12] Bela Gipp, Jagrut Kosti, and Corinna Breitinger. 2016. Securing Video Integrity Using Decentralized Trusted Timestamping on the Blockchain. In *Proceedings of the 10th Mediterranean Conference on Information Systems (MCIS).* Paphos, Cyprus.

[13] Oded Goldreich. 2004. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, New York, NY, USA.

[14] Google. 2017. *Google Appscript.* https://developers.google.com/apps-script/.

[15] Gideon Greenspan. 2016. Four Genuine Blockchain Use Cases. (May 2016). http://www.coindesk.com/four-genuine-blockchain-use-cases/

[16] R. R. Downs R. Duerr J. C. Goldstein M. A. Parsons Hills, D. J. and H. K. Ramapriyan. 2015. The importance of data set provenance for science. (2015). version: doi:10.1029/2015EO040557.

[17] Roman JagomÃďgis, Peeter Laud, and Alisa Pankova. 2015. Preprocessing-Based Verification of Multiparty Protocols with Honest Majority. Cryptology ePrint Archive, Report 2015/674. (2015). http://eprint.iacr.org/2015/674.

[18] Cruz Jason, Paul and Kaji Yuichi. 2015. The Bitcoin Network as Platform for Trans-Organizational Attribute Authentication. *IPSJ SIG Notes* 2015, 12 (feb 2015), 1–6. http://ci.nii.ac.jp/naid/110009877764/en/

[19] Ahmed Kosba Andrew Miller Kevin Delmolino, Mitchell Arnett and Elaine Shi. 2015. Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab. Cryptology ePrint Archive, Report 2015/460. (2015). http://eprint.iacr.org/2015/460.

[20] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2015. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. Cryptology ePrint Archive, Report 2015/675. (2015). http://eprint.iacr.org/2015/675.

[21] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. 2017. ProvChain: A Blockchain-based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid '17).* IEEE Press, Piscataway, NJ, USA, 468–477. https://doi.org/10.1109/CCGRID.2017.8

[22] Peter Linder. 2016. DEcryption Contract ENforcement Tool (DECENT): A Practical Alternative to Government Decryption Backdoors. Cryptology ePrint Archive, Report 2016/245. (2016). http://eprint.iacr.org/2016/245.

[23] Project Provenance Ltd. 2015. Blockchain: the solution for transparency in product supply chains. (2015). https://www.provenance.org/whitepaper

[24] MeteorJs. 2016. (May 2016). https://www.meteor.com/

[25] MongoDB. 2017. MongoDB. (Jan. 2017). https://www.mongodb.com/

[26] Open Provenance model 2007. *Open Provenance Model.* Open Provenance model. http://openprovenance.org/.

[27] Aafaf Ouaddah, Anas Abou El Kalam, and Abdellah Ait Ouahman. 2016. FairAccess: a new Blockchain-based access control framework for the Internet of Things. *Security and Communication Networks* 9, 18 (2016), 5943–5964. https://doi.org/10.1002/sec.1748

[28] Eric F. Pettersen, Thomas D. Goddard, Conrad C. Huang, Gregory S. Couch, Daniel M. Greenblatt, Elaine C. Meng, and Thomas E. Ferrin. 2004. UCSF Chimera - A visualization system for exploratory research and analysis. *Journal of Computational Chemistry* 25, 13 (2004), 1605–1612. https://doi.org/10.1002/jcc.20084

[29] Aravind Ramachandran and Murat Kantarcioglu. 2017. Using Blockchain and smart contracts for secure data provenance management. (2017). https://arxiv.org/abs/1709.10000

[30] Gavin Wood. 2017. ETHEREUM: A secure decentralized generalized transaction ledger. (2017). http://gavwood.com/paper.pdf