Decentralized Document Version Control using Ethereum Blockchain and IPFS

N. Nizamuddin[1] K. Salah[1] M. Ajmal Azad[2] J. Arshad[3] M. H. Rehman[4]

[1]ECE Department, Khalifa University of Science Technology and Research, UAE
[2]Dept. of Computing and Mathematics, University of Derby, UK
[3]School of Computing and Engineering, University of West London, UK
[4]Department of Computer Science, National University of Computer and Emerging Sciences, Pakistan

Abstract *– In this paper, we propose a blockchain-based solution and framework for document sharing and version control to facilitate multi-user collaboration and track changes in a trusted, secure, and decentralized manner, with no involvement of a centralized trusted entity or third party. This solution is based on utilizing Ethereum smart contracts to govern and regulate the document version control functions among the creators and developers of the document and its validators. Moreover, our solution leverages the benefits of IPFS (InterPlanetary File System) to store documents on a decentralized file system. The proposed solution automates necessary interactions among multiple actors comprising developers and approvers. Smart contracts have been developed using Solidity language, and their functionalities were tested using the Remix IDE (Integrated Development Environment). The paper demonstrates that our smart contract code is free of commonly known security vulnerabilities and attacks. The code has been made publically available at Github.*

Keywords*: Document Sharing, Version Control, Integrative Collaboration, Blockchain, Ethereum Smart Contracts, IPFS*

1. INTRODUCTION

Integrative collaboration has been one of the most important aspects of version control of documents, as it elevates trustworthiness among the parties involved. Management of accurate digital information and tracking changes in the digital asset when multiple parties are involved in preparing the document has become one of the major challenges faced in document version control. Document version control has been widely used in today's high paced environment facilitating shorter product developments and release cycles [1]. The advancement towards digitalization has introduced inaccuracy of content, document collaboration related issues, with 83% of productivity being consumed by version management issues [2]. Existing document version control systems [3] are mostly centralized and suffers from a single point of failure, featured by the increased time consumption, erroneous operations of the document updates allowing changes being made to a document without the knowledge of other users in the network. More importantly, with the centralized systems, the changes to the document and the update history can be tampered, therefore risking the credibility of changes and their update history. Hence, there is a need for a completely secure and decentralized platform [4] for the version management of digital documents.

Blockchain has become one of the promising technologies following the success of Bitcoin. The blockchain is the underlying technology of Bitcoin [5]. Blockchain provides a distributed ledger or database which is shared among all participants in the network based on the consensus mechanism. The need for a third party verifier is eliminated, making the system secure and completely decentralized. Any transaction which results in a modification to the Blockchain ledger is digitally signed, verified and validated by miner nodes which keep a duplicate of the ledger. This creates completely decentralized, secure, time-stamped and shared tamper-proof ledgers [5]. Blockchain technology has been utilized in many industries such as finance, healthcare, supply chain, logistics, document management and accounting

[6, 7, 8]. Due to its robust and decentralized infrastructure, blockchain technology is applied to handle issues related to trust, efficiency, privacy and data sharing [9]. This technology eliminates the requirement of a third party transaction authority by leveraging the potential of cryptography to provide trustworthy solutions for the entities participating in the chain.

Smart Contracts are codes that can be executed by the Blockchain mining nodes. A smart contract is a self-executing code that can verify the enforcement of predefined terms and conditions [10]. Instead of validating digital currencies, as in Bitcoin, a blockchain mining node executes, verifies and stores data in blocks. A smart contract is triggered by consigning a transaction to its Ethereum address and executing it depending on the input given for that transaction. Ethereum, as described in [11], is a blockchain-based, open source, distributed platform that features smart contract functionality. Ethereum allows users to write their code on top of the Ethereum platform enabling the development of bespoke applications. Ethereum uses Ether as a cryptocurrency for making payments for the transactions carried out on the Ethereum blockchain. Each participant in the Ethereum network is uniquely identified by an Ethereum Address (EA).

Blockchain has become one of the hyped technologies these days. However, storing large documents is still very expensive as the 1MB size limit per block in Bitcoin's blockchain would limit the file size that can be uploaded [12]. A pressing need for storing large size files was addressing using decentralized storage systems such as InterPlanetary File System (or IPFS), Storj, SWARM, and Sia. However, in this research work, we are using the most popular and well-established platform namely, IPFS. The IPFS is content addressable, peer-to-peer, open source, a globally distributed file system that can be used for storing and sharing a large volume of files with high throughput [13]. The blockchain is inefficient in storing large volumes of data. However, it has been proved to be effective when it stores hashes of documents in the chain, instead of the document itself [13]. A hash is generated every time a document is uploaded to the IPFS and this hash is stored in the smart contract which is used to access the document. The hash value changes each time, for any changes made in the content of the document.

Existing distributed version control systems are mostly centralized and therefore under the control of one central repository and user do not have complete control of the document or file [14]. With centralized systems, documents can be deleted, manipulated or tampered with. Moreover, considering the existing distributed version control systems, a developer/user associated with their account has the control to change entries stored on the central server. Figure 1 illustrates a traditional distributed architecture for the document version control which involves commands to 'update' the new versions into local repositories and 'push' commands to update the document onto the central repository. Control of the database still remains mostly centralized with administrators and a central authority. The verification in the distributed systems generally requires the signature of one authority, in case of a change, before it is 'committed' to the repository. This centralized notarization and verification control leads to core trust issues in the existing distributed systems. It is worth noting that the repository can be remote or cloud-based, and the repository can be local in nature hosted in the premise of the organization.

Document sharing and version control are one of the areas that can benefit from the blockchain technology. There is a need for a decentralized document version control solution without using a third-party authenticator to improve the security and scalability. Further, the document version control management issues approval of the new versions of the document need to be dealt with. The document version approval process generally involves many approvers and each of them has their own log. These logs have to be updated based on the signature received from all approvers along the distributed network. However, the registration process must have a unified log among all the parties which is not supported by the existing document version control systems.

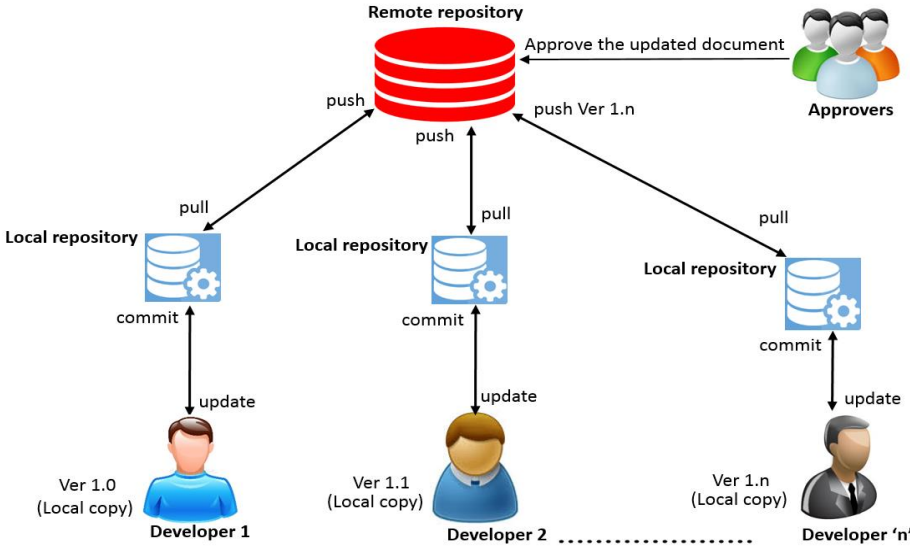

Fig. 1. Traditional document version control systems

Motivated by the need of having a reliable, trusted, decentralized document version control system, this paper proposes a blockchain-based solution for the controlled document sharing and version control. The blockchain is a decentralized technology, where the participating entities need not trust each other and maintains consensus about the status and existence of shared records in a trustless environment [9]. With this technology, tampering of records is eliminated as it utilizes cryptographic techniques to protect user identity and to ensure safe transactions by securing all information exchanged along the chain. Specifically, in the blockchain, every block is verified independently (i.e. consensus by all active participants) before adding to the chain. The registration requests of new users and approval of new versions of a document submitted by the users can be administered with the help of smart contracts. In our proposed solution, we utilize smart contracts to have code that automates the version control logic and workflow of digital documents with the ability to facilitate controlled or restricted data sharing mechanism. The smart contract code basically orchestrates all the interactions among multiple participants (including those approvers and developers) in a way that is completely decentralized

The primary contribution of this paper can be summarized as follows:

- We present a blockchain-based solution for document version control for digital documents using Ethereum smart contracts. Our proposed solution eliminates the requirement of a trusted third-party authenticator between the developers and the approvers.
- We highlight key aspects of our blockchain solution in terms of the overall system design and architecture, featuring main interactions among participants. These aspects are generic enough to be extended and applied for the version control of any form of documents in industries such as healthcare, finance, and land registry departments.

- We discuss key issues related to the registration of new developers or approvers, approval of new registrations, logical flow and interactions, implementation and testing of the overall system functionality.
- We perform the vulnerability and security analysis on the smart contract code to check for commonly known bugs and vulnerabilities.

The remainder of this paper is organized as follows. Section 2 summarizes the related work. Section 3 presents our proposed solution for a decentralized document version control. Section 4 describes key aspects of the implementation and testing of the smart contract. Section 5 presents smart contract security and vulnerability analysis. Section 6 concludes the paper.

2. RELATED WORK

In this section, we review work found in the literature related to blockchain-based version control and controlled data sharing of digital documents. We found most of the existing solutions reviewed in this section are abstract and high level in nature with no implementation details. Figure 2 shows the applications and objectives of existing systems.

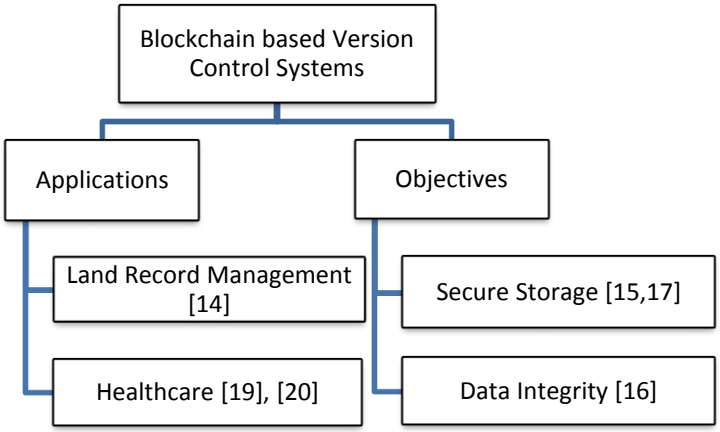

Fig. 2. Applications and Objectives of Blockchain based version control systems

A document management system that is based on blockchain was used by the Swedish government [14] for registering land documents and for recording land titles in order to digitize the real estate field. The Swedish National Land Survey department is working closely with a blockchain startup, *ChromaWay*, on a proof-of-concept model that checks how blockchain can minimize the risk of manual errors. This method aims to create a more stable and secure system for updating existing documents and transferring documents among the stakeholders. This proposed technology is also used to verify the user's identity who registered on a smart contract-based system. By harnessing the potentials of blockchain technology, the entire history of a land document can be tracked, restored and verified by all participating entities. The model is prominent but is still in the testing phase.

A publically available, open source, mineable blockchain ecosystem powered by high-level encryption and blockchain technology for the record management and document security was proposed by RecordsKeeper [15]. This organization aims to create a stable platform for secure transfer and authorization of data. The document access between peer groups over the decentralized storage network becomes easier with improved security. The blockchain based technology provides a platform to create immutable records which cannot be tampered, and which traditional database systems such as MySQL and Oracle do not provide. RecordsKeeper enables the end user to obtain a rigid structure for storing the documents on the blockchain which is verifiable at any point of time, thereby providing the sovereignty to the user to focus on the exact use case/ problem. Though this model is of high level in nature for management of records, there is no actual implementation for governing the version control of documents.

A global business group, Iron Mountain [16] has a dedicated medium to store, protect and manage digital information on a large scale. With the advent of blockchain technology, entities that are completely unaware of one another are enabled to carry out trusted transactions, believing in the authenticity of digital assets being exchanged. Traditionally, there are many ways to track the evolutionary development and the integrity of a physical document, such as digital signatures and digital watermarks. But the authenticity of assets in a digital form pose a major challenge as the digital content can be modified, copied or corrupted. Though currently there are many products available to provide secure and authenticated document management, they rely on third-party services which are not cost effective and completely reliable. This organization aims to provide robust information management and secure storage platform [16]. Each participant in the network gets a copy of a document and if any changes are made to the document, a new block is added and the revised file is synchronized with the previous version, in the form of a chain. Such a blockchain based structure acts an audit trail as well as a version tracking system. In such a structure, any participant can trace back to check the changes made in the network.

A novel approach for securing document transfer using Ethereum to provide secure storage and transfer for various kinds of financial, legal or other types of sensitive documents was developed by Eleks Labs [17]. The organization created an environment where legal transactions can be processed safely and with no third-party intermediary required. The developed system is a permissionless blockchain, where anyone can join as a participant in the network and can view or commit all transactions. The purpose of this system is to provide a secure storage and transfer mechanism for any kind of documents, such as legal agreements, financial documents, personal documents and more. The main aim of developing such a system was to ensure effective and safe transactions, eliminating the need for an intermediate. Practically, in order to verify a legal agreement, the participating entities require a certifier who verifies the content, signs and registers it. Blockchain technology is cost-effective as it eliminates the need for a notarization authority. Here, Ethereum based smart contracts perform the functions of verification and maintenance of documents on IPFS. By using cryptographic techniques, the signatures from various parties can be ensured. The smart contract provides an interface for restricted access and for tracking the changes in the document. All the legitimate participants in the network could change existing documents and view the updates that are tracked along the chain if they possess an encryption key. The importance of sharing, updating or modifying and reproducing research and scientific journal data is emphasized by authors in [18]. The biomedical research community encourages sharing the journals' research data. According to Zigmond and Fischer [19], the advancement of scientific contributions not only depends on individual discoveries but also on the consent of researchers to share the outcome of the research. The journal publishers play a major role in handling effective and secure data sharing. The authors of this paper evaluated various scientific research journal data sharing policies, considering different research platforms in biomedical science such as Medical Informatics, Biomedical Engineering, Cell Biology and more statistical methods such as chi-square test were employed to test the association of data sharing policies and open access status of journals. Two major problems were identified with the journal data sharing policies. Firstly, it was the low percentile of acceptance by journal publishers to implement a strong data sharing policy. Secondly, the policy guidelines are often ambiguous and irrational which makes the journal publishers deny the policy on a large scale.

The authors in [20] presented a methodology for sharing clinical trials documents using a controlled data access approach. As defined by the standard medical regulations authority, the IP details must be restricted within the doctor-patient environment as open access data might become a vulnerable platform for attackers or outsiders to gain access to sensitive medical documents that the patient chose not to disclose. In this paper, the restricted sharing of sensitive medical documents is discussed. The authors claim that de-identifying a patient using anonymization techniques is also not a successful option for safeguarding the sensitive data when more number of stakeholders are present in the chain. By combining the factors like age, gender, location, date of consent, health parameters and by using closest-match algorithms the critical dataset of a patient can be identified [20]. The authors propose a "formal data sharing agreement" to facilitate data transfer among various participants in the network. Indirect identifiers are removed, and data is transferred based on the appropriate handshake mechanism both at the senders' and receivers' side. This appears to be a better approach for controlled data sharing but has shortcomings such as a messaging system which may be prone to attacks and hence this proposed implementation is considered for further investigation.

3. PROPOSED BLOCKCHAIN-BASED SOLUTION

In this section, we propose and describe our solution which utilizes Ethereum blockchain and smart contracts to approve, track, and carry out the version control functions for the document stored on IPFS. Our solution eliminates the need for a trusted centralized authority and provides transactions and records to share and keep track of different versions of online documents with high integrity, resiliency, and security.

3.1 System Overview

This paper proposes a new paradigm using Ethereum smart contracts to enable controlled data sharing. Our solution focuses on permissioned document sharing, approval, and tracking of new versions of the document stored on the IPFS, the decentralized file system. Each participant is identified in the network using a unique account address and has a public key and a private key. Our solution focuses on the usage of smart contracts, executed to trigger events and enable the participating entities to continuously monitor, track and trace every transaction. In Etherem, events are broadcasted from smart contracts to all users and miners on the network. In case of violation of predefined terms, it helps in rebounding immediately and provides a chance to restore the conditions to an optimal level. Figure 3 shows the general overview of the system architecture, highlighting the main components and participants. As shown in Figure 3, the main components are the developers, the approvers, the blockchain that has the EVM (Ethereum Virtual Machine) executing the smart contract and IPFS – a distributed and decentralized file system.

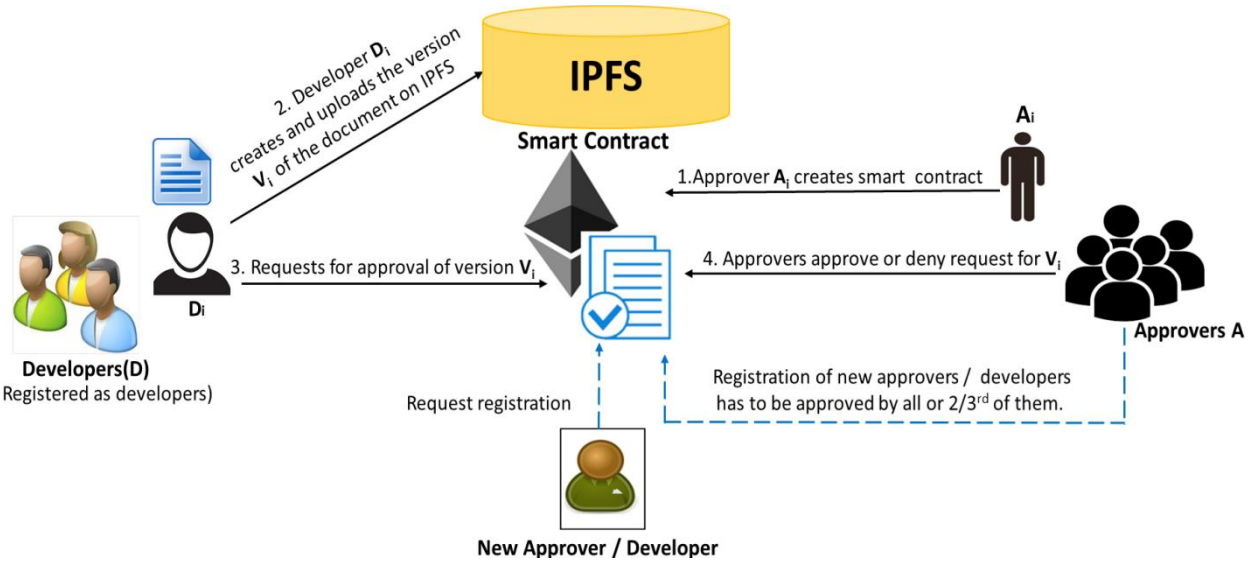

Fig. 3. Ethereum smart contract-based system for controlled document sharing

3.2 System Design

In this subsection, we describe the role and interactions among participating entities; namely, the developers, the approvers, and the smart contract.

Developers (D) and Approvers (A): The system primary participants/actors include the developers and approvers who are the key players involved in the document version control system and are identified by their Ethereum addresses. A smart contract that governs the interaction among the participants is created by one of the approvers. The approvers are registered and can join the other list of approvers if all other approvers agree. There is one smart contract per document. Initially, the creator of the document is the creator of the smart contract, and after other approvers join in, the approval of the document version is done by consensus by all other approvers. The developer creates and uploads (off-chain) the document to the IPFS distributed file system, and subsequently requests for approval from the

registered approvers, by providing the IPFS hash. Uploading a document off-chain indicates that the transactions or events take the value outside of the blockchain which are the transfer agreements between parties in general. An off-chain transaction is one that is not carried out over the blockchain but over an out-of-band Internet connection. An off-chain transaction is however still secured by the blockchain by 2-party payment channels which reduces costs and improves the speed of the transaction. For every approval granted, a new version of the document is stored in the file system. In addition, our solution also provides a way for registration of new developers or approvers which is approved only when 'all' or if 2/3rd of the approvers grant permission.

Smart Contract: All the interactions among the participants and transactions are governed by the smart contract. It handles the registration of users, requests for approval of new version on IPFS, approval and denial history of documents, and registration of new developers or approvers in the chain. The smart contract contains the following:

- **Methods.** Methods are functions that specify what the contract does. Some methods have restrictions that only allow a certain entity to execute them; others might be available to be accessed by all participants. The methods in the smart contract are directly related to the functionality of the contract.
- **Modifiers.** Modifiers can be used to modify the behavior of a function. They can be used to specify or check a condition prior to execution. Modifiers are used to strict the access and execution of the function within the smart contract to only certain participants.
- **Variables.** Variables are the values that change, depending upon the conditions or function calls. Depending on the contract, variables may be able to store a specified data type.

To ensure the document version control in a decentralized environment, the smart contract governs the interaction among the entities. The system comprises of a team of registered developers (D), a group of registered approvers (A), contract code, programmed in solidity, residing on the Ethereum blockchain that allows developers and approvers of the documents to register themselves. An approver A_i creates the smart contract following which a developer D_i creates a document version V_i and uploads it on the IPFS. The hash of the document stored in IPFS is saved in the smart contract. The developer D_i then requests for approval of the version V_i from the approvers. Upon successful approval by 'all' approvers, the version reference of the document is stored in the smart contract. The smart contract keeps an immutable record of versions approved by all or two-thirds of all approvers.

4. IMPLEMENTATION DESIGN AND TESTING

In this section, we provide details related to the implementation, such as the attributes, events, functions, and modifiers. Furthermore, the testing section specifies the use case scenarios and their results. We have made the full code of the smart contract available at Github[1].

4.1 *Implementation Details*

Our smart contract has two main participating entities; namely, the developers and the approvers. Each of the entities has an Ethereum address and can participate by calling functions in the smart contract at certain times. An entity is not allowed to make any direct function call, as this is done through modifiers. The modifiers restrict the functions to be called by only specific parties. Figure 4 illustrates the sequence flow for the successful approval of a document version of a scenario. Firstly, as mentioned in the previous section, one of the registered approvers A_i creates the smart contract which includes details of the document including the name of the document and its IPFS hash. Meanwhile, a registered developer D_i creates and uploads a document with version V_i on IPFS and stores the hash value in the contract. Then approval is requested by the developer D_i by providing the hash, to update the new version off-chain.

[1] The full code is available at: https://github.com/SmartContract1/DocumentVersionControl

Upon successful approval by at least two-thirds of all approvers A, the version reference is saved in the smart contract. This process continues, for each developer uploading a new version.

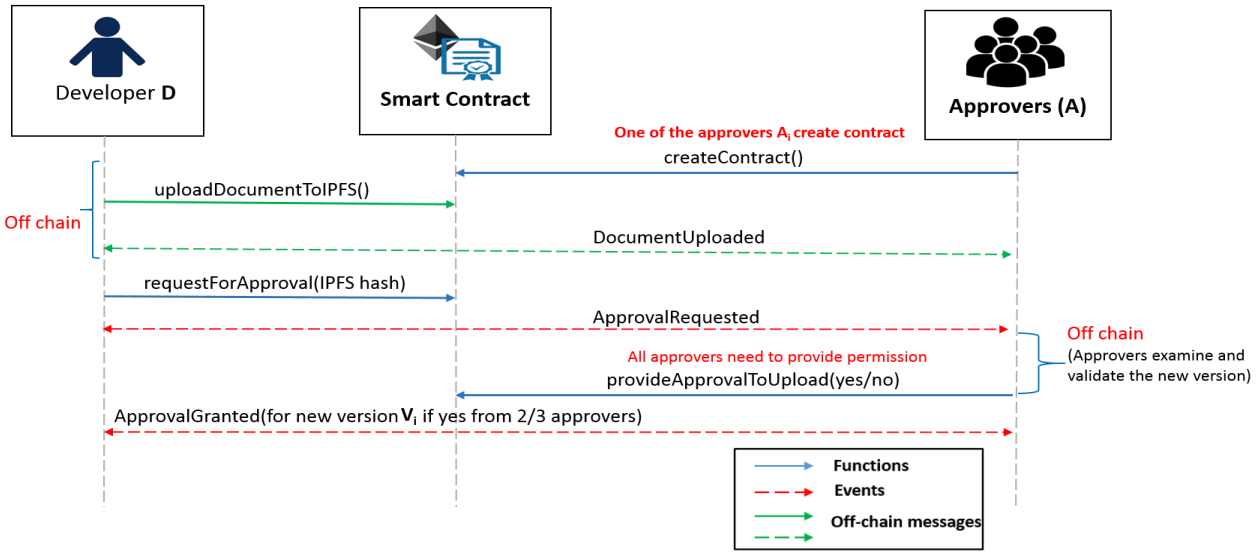

Fig. 4. Message sequence diagram for document approval

Additionally, the smart contract is deployed to monitor the registration of new developers or approvers. For a successful 'new' registration, all registered approvers must acknowledge and approve the request. Figure 5 demonstrates the message sequence diagram for new registration requested by a developer or an approver. In this case, the new entity, requesting the registration would submit a request to the smart contract. The request for a new registration would be only granted if the request is approved by all the active participants. If 'all' approvers' do not grant consent, then the registration request by a new participant, as a developer or as an approver is denied.

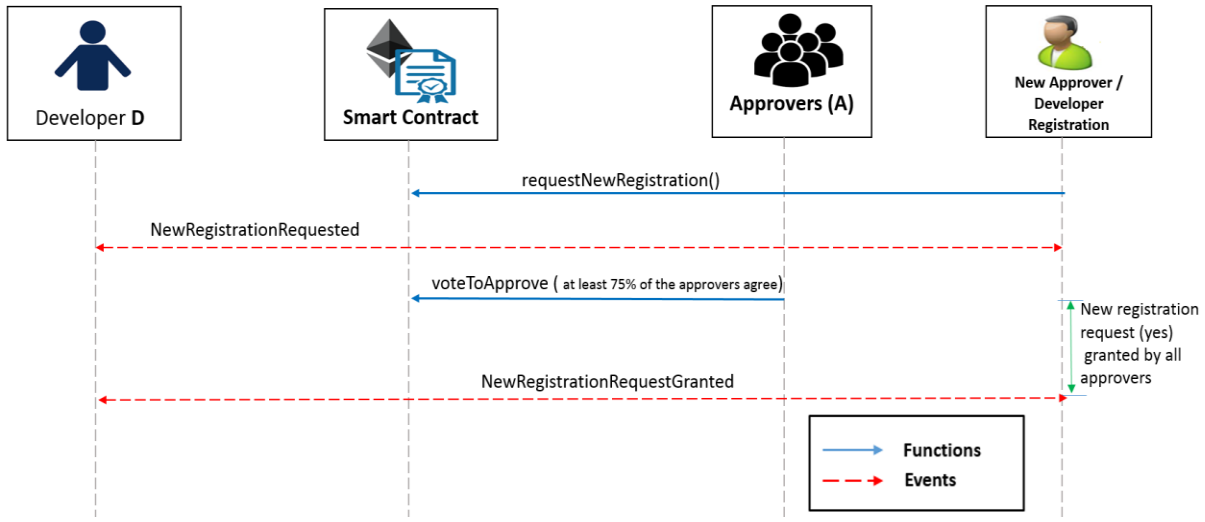

Fig. 5. Message sequence diagram for successful registration of new developers/approvers

Algorithm 1 shows the details for a registered developer requesting approval from the approvers for updating the document version on IPFS. It is clear from Algorithm 1 that the function requesting approval for the document version will execute only when the developer requesting the approval is registered and the document hash matches the IPFS

hash stored in the smart contract. If these conditions are not met, then the transaction aborts and returns to its initial state. At this stage, the state of contract changes from `Created` to `WaitForApproversSignature` and the state of the developer changes to `SubmittedForApproval`. The transaction notifies with an event requesting validation of the document from at least two-thirds of all approvers.

Algorithm 1: Approving document versions of requests made by developers

Input : *DeveloperEthereumAddress*(EA),
IPFS hash,
D is the list of RegisteredDevelopers

ContractState *is* ***Created***
DeveloperState *is* ***ReadyToSubmit***
d *is the set of RegisteredDevelopers*(D)
$d1$ *belongs to the list of* 'D'
Restrict access to only $d \in D$
if *developer is registered and IPFS hash = true* **then**
    ContractState *changes to* ***WaitForApproversSignature***
    DeveloperState *is* ***SubmittedForApproval***
    Create a validation message requesting validation from 'all' approvers
end
else
    Revert ContractState and show an error.
end

Next, we present Algorithm 2 in which all the approvers provide the consent for the document version uploaded on the IPFS. As shown in Algorithm 2, the requesting developer must be registered and the hash provided as input must be the same as the IPFS hash stored in the contract. The state of contract and developer state changes to `SignatureProvided` and `ApprovalProvided` respectively on a scenario when successful approval is provided by the approver. The state of the approver is `ApprovalSuccess`. Algorithm 2 also illustrates the scenario when the approval is not provided. Here, the contract state changes to `SignatureDenied` and the developer state changes to `ApprovalNotProvided`. Further, it is clear from Algorithm 2 that approvers state changes to `ApprovalFailed`.

Algorithm 2: Providing consent by Approvers for uploaded documents

```
Input : DeveloperEthereumAddress(EA)
ContractState is WaitForApproversSignature
DeveloperState is ReadyToSubmit
ApproversState is WaitingToSign
d is in the set of RegisteredDevelopers(D)
Restrict access to only d ∈ D
if documenthash[developerAddress]=IPFSHash of Document then
    ContractState changes to SignatureProvided
    Change DeveloperState to ApprovalProvided
    ApproversState changes to ApprovalSuccess
    Create a validation message stating request ids granted
end
else
    ContractState changes to SignatureDenied
    Change DeveloperState to ApprovalNotProvided
    ApproversState changes to ApprovalFailed
    Create a validation message stating document version approval failed
end
else
    Revert ContractState and show an error
end
```

Algorithm 3 presents the pseudocode for a developer requesting a new registration entry in the network. Initially, the new registration state is `WaitToRegister` as shown in Algorithm 3. Upon a successful request scenario, the state of the contract changes to `NewRegRequested` and that of the new registration state changes to `NewRegistrationRequested` and a notification message is broadcasted to all registered approvers and developers to approve of the new registration. The contract first checks if the request is from a new entry following which the state change occurs.

Algorithm 3: Processing new registration requests

```
Input : EthereumAddressOfNewEntrants(EA)
ContractState is SignatureProvided
new RegistrationState is WaitToRegister
if NewEntrant(EA) is already registered then
    ContractState reverts and shows an error
end
else
    ContractState changes to NewRegRequested
    new RegistrationState changes to NewRegistrationRequested
    ApproversState changes to ApprovalFailed
    Create a notification message to grant permission for new registrations
end
else
    Revert ContractState and show an error
end
```

4.2 Testing

In this section, we focus on testing the correct interaction and functionality among system participants of the Ethereum smart contract. Our smart contract has been implemented and tested using Remix IDE. All functions were tested to ensure that the logic and the state of the contract worked correctly. In Remix, three Ethereum addresses were used for

testing purpose. For testing purpose, we consider the addresses of the participants i.e., approver, developer and new developer to be `0xca35b7d915458ef540ade6068dfe2f44e8fa733c`, `0x14723a09acff6d2a60dcdf7aa4aff308fddc160c` and `0x583031d1113ad414f02576bd6afabfb302140225` respectively, such that each having 100 Ether to test the contract code. Functions are made to be executed in a certain sequence using the contract state as a requirement to execute any function call. The approver creates a smart contract. The registered developer creates and uploads a document in the IPFS and stores the hash of the document in the smart contract. Figure 6 shows the developer requesting the version approval of the document from the approvers. The developer is required to provide the IPFS hash of the document while requesting for approval and the event `ApprovalRequested` is broadcasted throughout the chain. The IPFS document hash of the initial version of the document is `QmXgm5QVTy8pRtKrTPmoWPGXNesehCpP4jjFMTpvGamc1p` which is given as one of the input parameters as shown in Figure 6.

Fig. 6. Logs showing developer requesting for approval of the document

The approver is notified about the event and `provideApprovalToUpload()` function is executed. Figure 7 shows the occurrence of an error when the approval is provided and the function is executed by another entity other than the registered approver address.

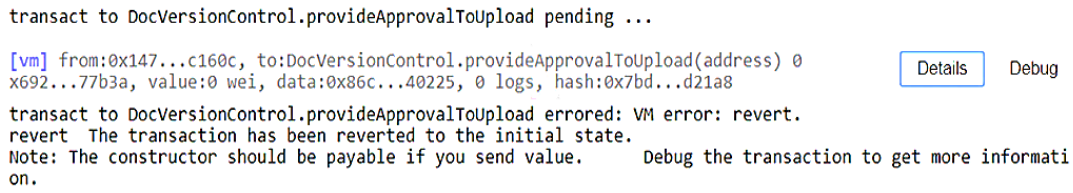

Fig. 7. Error message when the approval is provided by non-registered approvers

Furthermore, all the events are checked in the log after executing the functions to assure that the correct event was triggered. If the `provideApprovalToUpload()` function is successfully executed, events `NewVersionSigned` and `ApprovalGranted` are triggered and the log will have the message "`Document Version 1.0: Approved by All Registered Approvers in the Chain`" as shown in Figure 8. This message will only be displayed in the log when the requesting entity is a registered developer and has also entered the correct IPFS hash of the document stored in the file system while testing. It is also noted that, for a successful approval by the approvers, the IPFS hash should be of type 'string' and it must be the same hash of document which

is stored in the contract while testing. The result of the function is successful only if all registered approvers grant permission to upload the document. The approver compares if the hash provided by the developer during testing and hash of the document uploaded are same and grants permission for the new version.

Fig. 8. Log after executing provideApprovalToUpload() function successfully

Figure 9 shows the developer being denied permission for the document version by the registered approvers in the chain. We test this scenario with the address of the developer to be "`0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db`", who requests permission from the approvers by providing a different hash; Events `SignatureNotProvided` and `ApprovalRejected` are triggered to notify the requesting developer that the document was not approved by all registered approvers to update on IPFS.

Fig. 9. Logs showing event SignatureNotProvided and ApprovalRejected

Now, we test the smart contract for new registrations of developers or approvers by executing the function `requestNewRegistration()`. The contract checks whether it is a new registration request by using modifier `NotDeveloper1`. We test this scenario with an address "`0x583031d1113ad414f02576bd6afabfb302140225`" as the new entity request. Figure 10 shows the scenario of successful new registration with the execution of

`requestGranted()` function. The events `ApprovedSuccess` is triggered and the logs will show the list of approvers and registered developers who grant permission to register as a new entity.

Fig. 10. Log showing event ApprovedSuccess triggered

Finally, we test with the address "`0x583031d1113ad414f02576bd6afabfb302140225`" for the failure scenario where the new registration request is not successful and not approved by all registered parties. Figure 11 shows event `DenyRequest` is triggered and the logs show the message "`Registration Denied`". In this case, since one of the approvers did not validate the registration, the event `DenyRequest` was triggered. As discussed earlier, the new registration is only accepted when all or 2/3rd of the registered approvers/ developers approved the registration of the new entrant.

Fig. 11. Logs showing event DenyRequest triggered

5. SECURITY AND VULNERABILITY ANALYSIS

It is crucial to ensure the implementation of the smart contract is free of bugs and vulnerabilities and secure against attacks. In this section, we analyze the security vulnerabilities of Ethereum smart contracts. It is important to safeguard

the code and the information on the network, as they may be vulnerable to attacks. There is a definite need for blockchain engineering, addressing this issue posed by smart contract programming and other applications running on blockchain [21]. The vulnerability issues are due to poor and negligent programming practices in the solidity language. Unlike web applications, Ethereum platform is directly linked with monetary funds and benefits and any exploitation results in immediate and detrimental financial loss. These significant economic losses have created an appreciation for an effective security model that most fields have had a hard time achieving. Testing smart contracts for vulnerabilities has become an important part of gaining confidence in a contract's soundness and stability.

For example, the smart contract for the decentralized autonomous organization (DAO) which was built on the Ethereum platform had serious code vulnerability and was hacked in 2016. This resulted in 3.6 million Ether being stolen [22]. The DAO was meant to operate like a venture capital fund for crypto and decentralized platform. The DAO attack and consecutive events have changed the viewpoint on the security of the Ethereum platform and also put limelight on the security of cryptocurrency and Blockchain technology. Defects in code and hackers always remain a threat to the security of the smart contract. Table 1 shows the classification of most commonly identified vulnerabilities in Ethereum smart contracts.

Table 1: Classification of vulnerabilities in Ethereum smart contracts

Vulnerability Source	Vulnerability Type
Solidity Code	Call to unknown address
	Gasless send
	Reentrancy attack
	Exception disorders
	Type Casts
Blockchain	Generating randomness
	Unpredictable state
	Time Constraints
EVM (Ethereum Virtual Machine)	Immutable bugs
	Ether lost in the transfer
	Stack size limit

We verified our smart contract code for popular security vulnerabilities. To accomplish this, we utilized a Smart Contract analyzing tool called "ChainSecurity" [23]. This tool has the ability to check the smart contract code to find vulnerabilities and bugs. It also provides implicit guidelines to smart contract developers about insecure coding patterns and any unexpected ether flows. We tested our smart contract with Securify tool against the commonly known security vulnerabilities and attacks. As shown in Figure 12, our smart contract is secure enough any of the known attacks as no issues were reported.

Fig. 12. Securify tool for smart contract vulnerability analysis.

Another important smart contract analyzing tool is Oyente [24]. Oyente is compatible with any Ethereum based programming languages such as Solidity, Serpent, and low-level Lisp-like language (LLL). We have tested our smart contract with both tools against the commonly known security vulnerabilities and attacks. As shown in Figure 13, Oyente tool generated a report showing that all reported vulnerability results were "False" and our smart contract proved to be secure enough and bug-free of any of the known vulnerabilities.

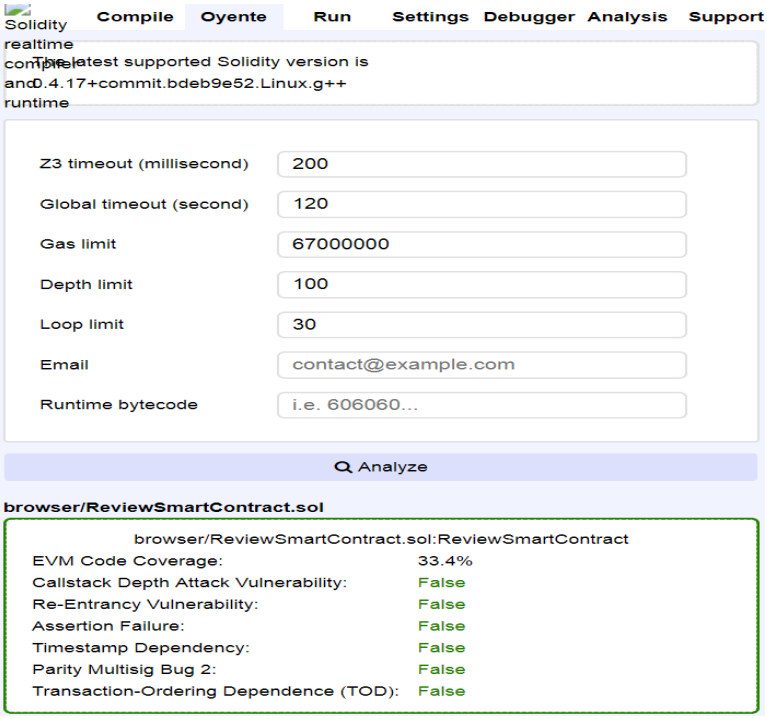

Fig. 13. Smart contract vulnerability output report generated by Oyente tool

We proposed a novel blockchain based decentralized version control framework and compare it with traditional centralized and decentralized version control systems (See Table 2).

Table 2: Comparison between conventional and blockchain based version control frameworks.

Features	**Centralized (Conventional)**	**Decentralized (Conventional)**	**Blockchain based (Decentralized)**
Mode of Operation	• Client-server • Cloud-based • P2P	• Client-server • Cloud-based • P2P	• P2P
Point(s) of Failure	• Single	• Multiple	• None
Merge Strategy	• Linear • Random	• Linear • Random	• Consensus Based
Trust Level	• Low	• Medium	• High
Graphical User Interfaces	• Web-based • Desktop-based • Integrated	• Web-based • Desktop-based • Integrated	• Web-based • Integrated
Monopoly	• High	• High	• Low
Strength	• Ad hoc merge and split	• Ad hoc merge and split	• Consensus-based merge and split
Weaknesses	• Roll back possible	• Roll back possible	• No Rollback

6. CONCLUSION

We proposed a decentralized framework and solution for version control and sharing of documents. Our solution leverages the benefits and features of Blockchain, smart contracts, and IPFS file system. The proposed system is completely decentralized, secure, and resilient, eliminating dependency on the trusted third party. We have implemented and tested the functionalities of our solution with the Remix IDE. Also, we applied popular security analysis tools; namely ChainSecurity and Oyente, to verify and demonstrate the resiliency and security of the developed smart contract against commonly known attacks. The framework presented in this paper with the system architecture, design, algorithm, and the smart contract code is generic enough and can be extended to similar systems involving version control and access to shared digital assets and contents which may include video, audio, and photos.

ACKNOWLEDGEMENT

The authors like to thank the AE and anonymous reviewers for their assessment and valuable feedback which significantly enhanced the quality and the presentation of the paper.

REFERENCES

1. T. Weddepohl, Oracle Technology Network, http://www.oracle.com/technetwork/topics/ikan-continuous-integration-084660.html, IKAN Software, Last accessed 2018/8/30.
2. Perforce, The Case for Better Document Collaboration, Knowledge Worker Survey, http://info.perforce.com/rs/perforce/images/versioning-report.pdf, Perforce Software, Last accessed 2018/8/21.
3. Git, Getting Started - About Version Control, Chapter 1, https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control, gist --distributed-even-if-your-workflow-isn't, Last accessed 2018/8/30.
4. H. F. Rashvand, K. Salah, J. M. A. Calero, L. Harn, "Distributed security for multi-agent systems - review and applications," IET Information Security, Vol. 4, No. 4, December 2010, pp. 188-201.
5. S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Whitepaper, 2008.
6. M.Khan and K.Salah, IoT security: Review, blockchain solutions, and open challenges, Future Generation Computer Systems, vol. 82, pp. 395-411.
7. M. S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli and M. H. Rehmani, "Applications of Blockchains in the Internet of Things: A Comprehensive Survey," in IEEE Communications Surveys & Tutorials. doi: 10.1109/COMST.2018.2886932
8. K. Salah, M. H. U. Rehman, N. Nizamuddin and A. Al-Fuqaha, "Blockchain for AI: Review and Open Research Challenges," in IEEE Access, vol. 7, pp. 10127-10149, 2019. doi: 10.1109/ACCESS.2018.2890507
9. J. Leon Zhao, S. Fan, and J. Yan. Overview of Business Innovations and Research Opportunities in Blockchain and Introduction to the Special Issue in Financial Innovation, 2016, vol. 2, No.1, pg. 1.
10. T. Bocek, B. B. Rodrigues, T. Strasser, and B. Stiller, Blockchains Everywhere - A Use-case of Blockchains in the Pharma Supply-Chain, 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, 2017, pp. 772-777.
11. G. Wood, Ethereum: A secure decentralized generalized transaction ledger, 2014, Ethereum project yellow paper, 151, pp 1-32.
12. J. Benet, IPFS-content addressed, versioned, P2P file system, 2014, arXiv preprint arXiv: 1407.3561.
13. M. Ernst, Version control concepts and best practices, 2014, https://homes.cs.washington.edu/~mernst/advice/version-control.html/, Last accessed 2018/8/31.
14. P. Rizzo, Sweden Tests Blockchain Smart Contracts for Land Registry, https://www.coindesk.com/sweden-blockchain-smart-contracts-land-registry/, last accessed 2018/5/20.
15. RecordsKeeper, https://www.recordskeeper.co/, last accessed 2018/5/21.
16. Iron Mountain, http://www.ironmountain.com/resources/general-articles/w/what-is-blockchain-and-why-should-records-management-professionals-care, last accessed 2018/5/21.
17. Eleks, Secure Document Transfer Built on Top of Blockchain Technologies, https://labs.eleks.com/ last accessed 2018/5/22.
18. N. A. Vasilevsky, J. Minnier, M. A. Haendel, and R. E. Champieux, Reproducible and reusable research: are journal data sharing policies meeting the mark?, PMC, US National Library of Medicine National Institutes of Health, PeerJ, April 25, 2017, Vol.5, p.e3208.
19. B. A. Fischer, and M. J. Zigmond, The essential nature of sharing in science, Science and engineering ethics, 2010, vol.16 (4), pp.783-799.
20. M. R. Sydes, A. L. Johnson, S. K. Meredith, M. Rauchenberger, A. South, and M. KB. Parmar, Sharing data from clinical trials: the rationale for a controlled access approach, Trials, March 2015, Vol.16, pp.104-110.

21. G. Destefanis, A. Bracciali., M. Marchesi and Robert M. Hierons, Smart Contracts Vulnerabilities: A Call for Blockchain Software Engineering?, International Workshop on Blockchain Oriented Software Engineering (IWBOSE), 2018, pp. 19-25.
22. D. Siegel, Understanding the DAO attack, Coindesk, June 2016. [Online]. Available: https://www.coindesk.com/ understanding-dao-hack-journalists/ [Accessed: 17-July-2018].
23. P. Tsankov, D. Andrei, D.D. Cohen, A. Gervais, F. Buenzli, and M. Vechev, Securify: Practical Security Analysis of Smart Contracts, 2018, arXiv preprint arXiv: 1806.01143.
24. L. Luu, D.H.Chu, H. Olickel, P. Saxena, and A. Hobor, Making Smart Contracts Smarter, ACM, In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 254-269.

Authors Biographies:

Nishara Nizamuddin currently works as a Researcher at the Department of Electrical and Computer Engineering, Khalifa University, UAE. Nishara conducts research on a project involving Blockchain and cybersecurity. She recently published a number of articles on Blockchain applications and infrastructure. She obtained her BSc and MSc degrees in Computer Science from VIT University, India, in 2010 and 2016, respectively.

Khaled Salah is a full professor at the Department of Electrical and Computer Engineering, Khalifa University, UAE. He received the Ph.D. degree in Computer Science from Illinois Institute of Technology, USA in 2000. His research interests include Cloud and Fog Computing, IoT, Blockchain, and Cybersecurity. Contact him at khaled.salah@ku.ac.ae

Muhammad Ajmal Azad

Junaid Arshad
Junaid Arshad is a Senior Lecturer at the University of West London. Junaid achieved his PhD from the University of Leeds, UK where he investigated the challenge of effective intrusion severity analysis for clouds. Junaid's research areas involve distributed computing, high performance computing such as grid and cloud computing, service oriented computing and Internet of Things emphasizing security challenges including intrusion detection and response, trust establishment and management, and security event classification.

Muhammad Habib ur Rehman is an assistant professor at National University of Computer and Emerging Sciences, Lahore, Pakistan. His research interests include blockchain, edge computing, big data, and Internet of things. Rehman received a PhD in mobile distributed analytics systems from the Faculty of Computer Science and Information Technology at the University of Malaya; Malaysia.