Implementating Cross-Enterprise Document Sharing (XDS.b) based on Blockchain Technology

*Healthcare information sharing and interoperability between healthcare organizations are important factors to healthcare quality and safety since a patient may require medical services and consultations from different healthcare providers. Many challenges inhibit successful data sharing such as data integrity, security, and privacy. Integrating Healthcare Enterprise (IHE) provides Cross-Enterprise Document Sharing-b (XDS.b) profile that allows the adopted organizations to share health documents between each others. No specific security implementations were endorsed for XDS.b, which allows latest security technologies to be applied. Healthcare domain has become a major target in emerging cyber-security threats. These threats increase difficulty to maintain secure health information sharing network. These cyberthreats can compromise integrity and availability of data and affect patient’s life. Blockchain technology can be applied to prevent some of cyber threat issues of health information sharing. A novel method using Blockchain technology to ensure health information integrity and availability is implemented, demonstrated and freely available, allowing health document sharing through decentralized network while addressing cyber-security issues through unique characteristics of Blockchain technology.*

Keywords—health information, interoperability, information sharing, information security, blockchain, smart contract, ihe, XDS.b

# Introduction

The increasing demand for more secure and high-quality healthcare services, as well as operational efficiency, all have a significant impact on patient care and economic outcomes. Healthcare information sharing and interoperability among healthcare organizations are two critical components to improving the quality of healthcare services. Generally, patients' health records of an individual are scattered across multiple healthcare organizations, which may be due to the fact that various medical services are offered by various providers. Each healthcare provider's methods, procedures, and workflow are unique in how they manage healthcare information. This complicates interoperability between health information systems. Sharing health information with third parties who do not adhere to security standards and practices could expose patients, businesses, and organizations to risk. The diminishing risk-reward ratio associated with sharing patient information between healthcare institutions may be unfavorable if done incorrectly. These interoperation problems cause a huge decrease in efficiency in healthcare operations and result in a lower quality of healthcare service [1–8].

To address these mentioned issues, there are many medical informatics initiatives [9] like DICOM [10–11], HL7 [12], Fast Healthcare Interoperability Resources (FHIR) [13], and Integrating Healthcare Enterprise (IHE) [14] to standardize and make use of healthcare information technology with the goal of allowing healthcare organizations to be able to exchange patients' information with one another. Amongst them, IHE provides precise requirements and specifies integration profiles, enabling interoperability between each system of healthcare services. To address the issue of health information sharing between multiple organizations, IHE has provided the Cross-Enterprise Document Sharing Profile (XDS.b Profile) [15]. This profile offers a detailed framework for software developers and system architects about how different enterprises share their health documents with each other. The profile also acts as a guideline, providing essential protocols for system implementers to follow.

Besides the issue regarding health information sharing between different enterprises, cyber-security threats have been an important issue threatening the healthcare domain. Since 1989 [16], hospitals have been hit by ransomware denying the availability of healthcare information, and it has interrupted medical operations and severely disrupted healthcare services. In addition, the number of data breaches in healthcare domain has been increasing every year from 2012 to 2020 [16–20]. The breaches raise a concern about the integrity of data being used for healthcare operations requiring accurate data. Moreover, as traditional healthcare information infrastructure was separately developed as a centralized system, when one system goes down, all the connected systems are also affected. Thus, they become "The Single Point of Failure" (SPOF) [22]. The case of the Czech hospital hit by a cyberattack during a COVID-19 outbreak [23] demonstrates that failing to secure the integrity and availability of healthcare information combined with SPOF can causes a significant disruption in the continuity of medical operations.

To solve the above issues, the concept of decentralization was introduced. Decentralization eliminates the need for a centralized mediator for one system to operate. That means if one system amongst the decentralized network goes down, the connected system will remain in operation and withstand the failure. To achieve decentralization, blockchain technology was then introduced as its base technology. The immutable characteristics of blockchain's cryptographic components and consensus mechanism will ensure the integrity of the information, while decentralization of published data will help ensure its availability [23–26].

Some research work [28-30] proposed to use Blockchain to address issues regarding health information sharing between different enterprises. The first two works introduce an effective way to utilize Blockchain technology for information sharing in a healthcare enterprise environment. They have given a demonstration of how the decentralization offered by Blockchain can resolve trust issues where each enterprise requires "trust" before beginning to share their information with others. The last work proposed by Tanwar et al. (2020) [30] surveyed all the proposed healthcare blockchain concepts and summarizes the situation in the field of research. They also proposed the concept of using blockchain to act as an access control system for health information. Their contributions mark a current point of progression for research that wants to use Blockchain for healthcare information. In this work, we propose an implementation that combines the IHE XDS.b profile with Blockchain technology. Enable cross-enterprise health document sharing that can sustain SPOF problems and strengthen the sharing of health information among healthcare enterprises without the need for a new information interoperability standard.

The next section provides information about relevant works that inspire our design, following with background knowledge which our work is based on in Section III. Then, we move into the detail of design method in Section IV before dive into implementation techniques in Section V. At last, we discussed about the proposed concept in Section VI and end with conclusion of this work in Section VII.

# RELATED WORK

Many important Blockchain technologies used for health information exchange provide general guidelines for the design and implementation of our system. The major influences came from the works described as follows.

## A Blockchain-Based Approach to Health Information Exchange Networks

Peterson et al. (2015) [28] from Mayo Clinic proposed the concept using Blockchain as a medium for health information exchange network. The work utilizes FHIR protocol as a gateway which allows members of the network to access health information from each other, while ensure distribution of accessibility within the network by publishing those gateways to Blockchain. Every activity on the network will be recorded on the Blockchain providing audit trail for the network. They also suggest several conceptual ideas that would enable blockchain technology to have a more meaningful way to contribute to the healthcare environment. Especially, the idea of "Proof of Interoperability", as opposed to current consensus mechanisms like Proof of Work, which is cost inefficient. Moreover, the proposed concept of "Secure Index" and using FHIR Profile for data discovery in the blockchain network also pave the way for the following research to avoid publishing healthcare information directly into the blockchain while making health information sharing possible. This reduces the risk of Blockchain being used for health information sharing backfiring due to its persistence. However, the work did not directly mention about how Blockchain technology can be beneficial for health information sharing in term of cybersecurity.

From this work, we adopted the idea of using Blockchain as a medium for health information exchange networks. Additionally, several suggestions provided in the work also act as guidelines for our Blockchain design, which enables compatibility with the healthcare information environment.

## “MedRec” prototype for electronic health records and medical research data

MedRec [31] was proposed as a prototype for electronic health records by utilizing Ethereum’s smartcontract to store metadata about the record ownership, permissions, and data integrity representing existing medical records that are kept in individual nodes on the network. The concept helps reduce barriers to effective data sharing by addressing interoperability issues caused by economic incentives that encourage "health information blocking" [32]. At the same time, their proposal also benefits as the source of medical research data by providing anonymized healthcare data to research institutions in the form of a Blockchain participation reward. Their Blockchain implementation focuses on addressing four major issues for health information exchange, including fragmented data, which also slows access to medical data, system interoperability, patient agency, and improved data quality and quantity for medical research. Additionally, as MedRec was built on the work of Zyskind et al. (2015) [33], they utilized some cryptographical characteristics of Blockchain to provide an accessible "bread crumb trail" that allows data users to trace back medical history to improve operational efficiency.

From MedRec, we adopt the concept of using Ethereum’s smartcontract to add a mechanism to the Blockchain, enabling its usage with healthcare information.

C. Blockchain-based electronic healthcare record system for healthcare 4.0 applications

Tanwar et al. (2020) [30] surveyed proposed healthcare blockchain concepts from 2016 to 2019 that would benefit the healthcare industry by enhancing the capabilities of electronic health records. The work has effectively explained the overall concept of incorporating Blockchain technology into electronic health records that have evolved over time. They also proposed a blockchain-based access control system for electronic health records by using IBM Hyperledger fabric as a medium for health information exchange within organizations. Similar to MedRec, their concept emphasizes improving access control and audit trail over information exchange between each unit of the healthcare service workflow. The work sets another example of effectively implementing blockchain smartcontracts for the possibility of healthcare blockchain.

The work rounds up the proposed healthcare information-based blockchain concepts and sets a current reference point for the following research in the same field.

# BACKGROUND KNOWLEDGE

This section provides essential background knowledge for the reader to understand this work. Subsection A provides background regarding IHE and their XDS.b Profile. This subsection can be skipped if the reader is already familiar with the initiative and its framework. Subsection B provides the core general concept of blockchain technology and its components. The reader may skip this subsection if they already have full knowledge of the blockchain concept. After that, the last subsection is about Ethereum and its smartcontract which we adopt for the implementation of this work. It is recommended for the reader to at least understand the concept of Blockchain smartcontract before proceeding to the next section.

## Cross-Enterprise Document Sharing-b (XDS.b) Profile from Integrating Healthcare Enterprise initiative (IHE)

Modern healthcare operations have large amount of healthcare information flow between multiple systems. Throughout the age, many medical service providers and organizations have developed their own health information systems and databases to increase efficiency of operations in their medical services. As the time past, information of individual patients has scattered amongst different systems. This becomes a new challenge for healthcare enterprise to further enhance their medical service efficiency by sharing health information with other systems within healthcare industry.

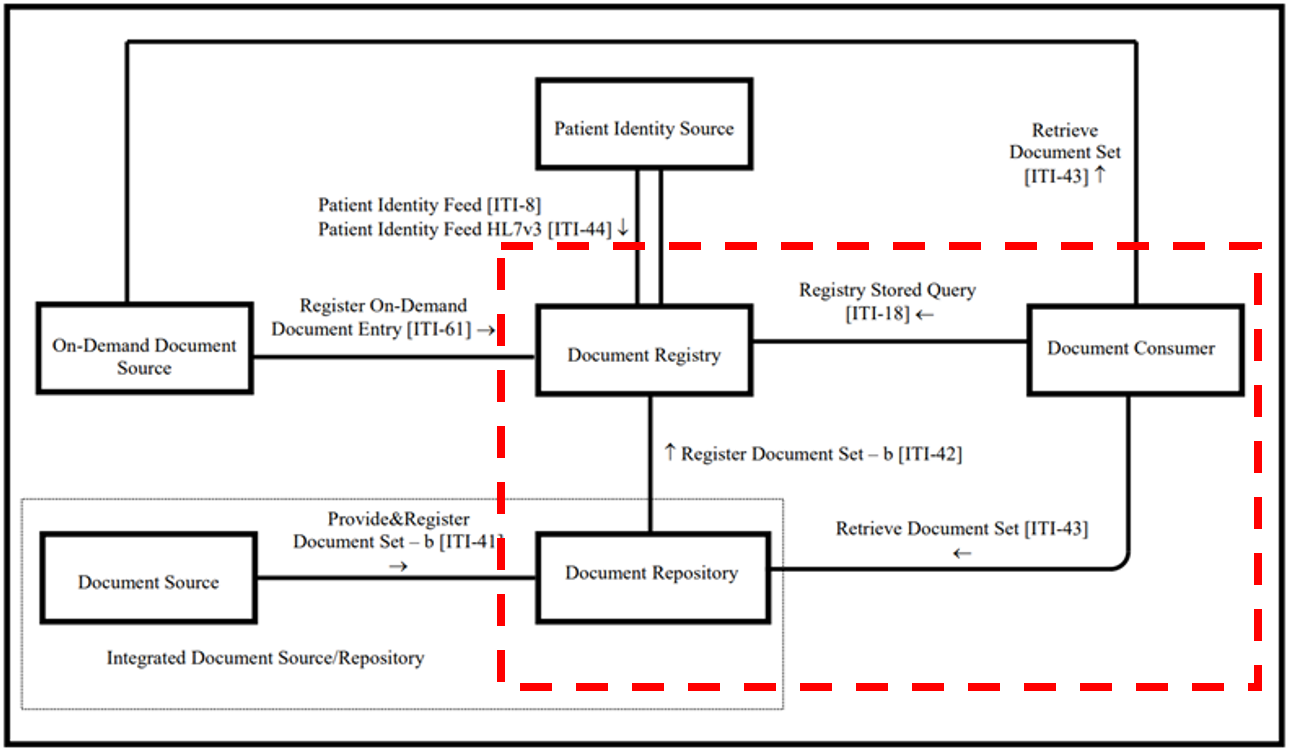


Figure 1 Cross-Enterprise Document Sharing – Set b [15]  
(Red-dash line) The scope of XDS essential for understanding this work

IHE is significant initiative by healthcare professionals, organizations, and industry to enhance the way health information systems in healthcare integrate and share information. It promotes the coordinated use of established standards such as information technology standards, HL7 and DICOM [12] to address specific clinical needs in support of information system integrations and optimal patient care. This helps enable seamless health information exchange that is usable whenever and wherever needed. Health information systems developed in accordance with IHE integrate and exchange information with one another better, are easier to implement, and enable healthcare providers to use information more effectively. An IHE profile provides use of existing standards, specifications, tools, and services for interoperability. It also engages healthcare institutions, clinicians, health authorities, industries, and users to develop, test, and implement standards-based solutions to vital health information needs [14]. IHE provides convenient and reliable way of specifying a level of compliance to standards enough to successfully reach efficient interoperability.

Amongst many profiles created by IHE, there is one profile that serves to improve efficiency of health information sharing between different enterprises called "Cross-Enterprise Document Sharing Profile-b (XDS.b)" [15]. The main goal of the XDS.b profile is to allow enterprises that are members of a health document sharing network (called "XDS Affinity Domain") to discover shared health documents stored in the document repositories of other institutions using a central registry. This central registry stores a set of metadata attributes that represent health documents kept within their source repository. These sets of metadata attributes act as an index which allows health XDS Affinity Domain members to discover health documents in the network. Once the preferred health document is discovered, then the XDS Affinity Domain member can systematically request access to the document from its source repository.

In the XDS.b Profile, each system, server, and machine are logically labeled as "XDS Actors". Figure 1 shows all the XDS actors involved in the profile and the sequence of how they interact with each other to enable cross-enterprise document sharing. Each actor has their own role in the framework as shown in Table 1.

Table 1 XDS Actors and its role in XDS.b Profile framework

|  |  |
| --- | --- |
| XDS Actors | Role Description |
| Document Source | - Create/Generate health document, i.e., Computerized physician order entry (CPOE) or X-ray modality.  - Health documents generated by this actor are categorized as "stable” documents which will not be changed in the future, e.g., a record of events that have already happened in the past. |
| Document Repository | Store health document generated from Source, i.e., Health document repository. |
| Document Registry | Register an index of health documents from repositories which allows them to be discoverable by members of the XDS network. |
| Document Consumer | An interface for health document users or a system which acts as a health document user by itself, e.g., CPOE or data analytic programs. |
| On-Demand Document Source | Create or generate "on-demand" health documents (as opposed to "stable" documents) that always update for the current state of the document when called, e.g., bedside monitoring and vital sign monitoring devices. |
| Patient Identity Feed | Coordinate patient identities as defined by each XDS affinity domain member, e.g., John defined as patient #123 in hospital A while defined as patient #321 in hospital B, both hospitals can refer to the same John using this XDS actor. |

Each XDS actor has their own set of "transaction" formats to communicate with one another. XDS transactions act as a medium for the transition between XDS actors. These transactions have content that refers to the actual health document. The content can be a set of metadata attributes for document registering or a set of search keywords matching the metadata attribute values of the preferred document. Table 2 shows the XDS transactions and their descriptions essential for understanding this work.

Table 2 XDS Transactions and its description

| XDS Transaction | Transaction Description |
| --- | --- |
| ITI-42 Register Document Set-b | - Register a set of metadata attributes into the Document Registry.  - When newly generated health documents are saved in the Document Repository, this event is triggered.  - The Metadata attributes set directly represent each health document stored within the Document Repository. |
| ITI-18  Registry Stored Query | - Query for a set of metadata attributes registered within the Document Registry.  - The Document Consumers provide a set of keywords expected to be matched with metadata attribute values that represent the health document they need. |
| ITI-43 Retrieved Document Set | - Request for a health document from its source repository.  - The Document Consumer uses information provided in metadata attributes retrieved from ITI-18 transactions to access the Document Repository. |

The main process of XDS.b Profile composes of two main functions: document registration and document query. For explanation, we look at the common workflow of XDS.b where a document is generated into the system. Following Figure 1, a health document is first generated from an XDS Document Source actor. The generated document will take the format of an ITI-41 transaction before being sent to the XDS Document Repository actor. Upon receiving the document via ITI-41 transaction, the XDS Document Repository actor extracts a metadata attribute set from the document. The actual document itself will be kept within the XDS Document Repository actor while an extracted metadata attributes set will be used for the document registration.

The document registration begins with the XDS Document Repository assorting the extracted metadata attributes set into the ITI-42 transaction format. Then, the transaction was sent to the XDS Document Registry actor. The XDS Document Registry then interprets the transaction and extracts the metadata attributes set. After that, the metadata attributes set will be registered in the XDS Document Registry database. The document can now be discovered by all members of the XDS affinity domain and is ready for document query.

A document query begins after the health document user needs a document for healthcare service following their role. The document user will use a user interface provided by the XDS Document Consumer actor to start a document query. The XDS Document Consumer actor will prompt the user for an essential set of search keywords that refer to the health document they are seeking. This can be a patient's name, the type of healthcare service provided, the time, or a unique identifier, etc. All these keywords must match with metadata attribute values that represent the document. The XDS Document Consumer also specifies a query type appropriate to the expected query result. With both search keywords and query type specified, the XDS Document Consumer then proceeds to create a transaction for the query using the ITI-18 transaction format. The transaction is then sent to the XDS Document Registry actor. The XDS Document Registry actor will extract search keywords from the transaction and use them to search the registry for a document with a matched metadata attribute value. The XDS Document Registry then proceeds to return the result to the XDS Document Consumer using the ITI-18 transaction format. The document user then uses the information provided in the search result to locate and access the actual health document from its source repository. Document access will be done using the ITI-43 transaction format.

## Blockchain Technology

Blockchain is generally described as a distributed digital ledger of cryptographically signed transactions grouped into blocks. The technology is a method that applied cryptographic techniques to locally ensure integrity of data while rely on decentralization and consensus mechanism to ensure integrity and availability of all data existing in the network [34]. These cryptographical techniques include the one that form “Block” and another one that form “Chain”.

In Blockchain, those data being published are small fragment of information that represents proof of action in its own application. Therefore, it calls a “transaction”. A set of transactions received by all Blockchain member nodes in the same network at the same period will be hashed together, imagine like put these transactions into the same box and label it with its hash value, formed a “Block”. Additionally, the hash value of each block also includes hash value of previously generated block causing formation of a “Chain”. Any attempt to modify content of published block will cause change in hash value of entire chain which trigger rejection from the network.

Together with hash “chain”, the concept also relies on “decentralization” of data where copy of entire chain was kept by many participants of the network called “node”. Any node with the version of chain that has even a bit different from the majority in the network will be rejected and the node will be forced to adopt the version adopted by the majority. These two techniques form together to become “Blockchain” which prevents modification of published content and ensures integrity of data. This concept guarantee that no one can ever be able to modify any data existing in Blockchain. However, there still need for a way to add more blocks into the chain. This can be done by the utilization of the consensus mechanism [35].

According to [34], Blockchain can be categorized into two main types that are xxxxx based on its permission model which determines who can maintain blockchain ledger (i.e. publish block). Permissionless blockchain networks provide simplicity in scalability of the network by allowing anyone in the network to maintain blockchain ledger. At the same time, permissioned chain allows only selected members to maintain or even just to participate in the network which allows accountability and auditability of the network in exchange for simplicity in scalability. Permissioned blockchain networks may also be used by organizations that wish to work together but may not fully trust one another. They can establish a permissioned blockchain network and invite business partners to record their transactions on a shared distributed ledger. The transparency provides by Blockchain and consensus with suitable permission model then create a passive “trust” amongst the network as no one in the network have absolute right to rules and manipulate the network and its content at their own will [36].

Consensus mechanism was invented to ensure that no one in the network can freely attempt to modify content of transactions before it being published inside Blockchain, whether by select a trustable node who will verify certain Block being published to the chain or have majority of reliable node approve authenticity of newly formed Block [37]. Some consensuses, like Proof of Work (PoW), require participant nodes (called “miner”) who wants to verify a Block to compete to solve mathematic puzzle. The winner will be able to verify the Block and get reward based on each network policy. As the puzzle requires each node to spend huge amount of computational resource, given randomness which make it nearly impossible for miner node to be able to get a hand on prefer Block and attempt suspicious activity during verification process [38]. On the other hand, some consensus mechanisms like Practical-Byzantine False Tolerance (PBFT), invented to allow Blockchain network with limited computational resource to select trustable validator(s). This kind of consensus uses voting mechanism which sacrifices ability to welcome anonymous node for lesser computational resources required for maintaining Blockchain and increased in efficiency of verification process, that means it rely on the environment that most of the nodes are not corrupted [39]. These mechanisms enable transparency in publication and verification of transaction in Blockchain. Additionally, the network also needs compatible permission model for its member to maximize effectiveness of consensus mechanism.

## Ethereum and Smart-Contract

Ethereum [40] is one of the well-known open-source Blockchain platforms. The platform initially invented by a developer named Vitalik Buterin and further developed by the Ethereum community. The main approach of Ethereum Blockchain is about using Blockchain technology for applications other than cryptocurrency, i.e., create digital evidence for an actual contract between parties that can replace paperwork and cannot be deleted or tampered with, or act as digital evidence for supply chain tracing during transport and exchanging of goods. The platform proposed the concept of “smart-contract” [21,22]. Smart-contract allows developers to integrate their small-size computation algorithms or snippet of logic into Blockchain. This gives Blockchain characteristics [25] to those codes. Enable a wide variety of applications to work with Blockchain. The concept of smart contract later was adopted by other Blockchain platforms, created infinite possibilities of Blockchain application suitable with a variety of computational environment and usage.

While each Blockchain platforms have their own technical methods for implementation, Ethereum’s smart-contract relies on JavaScript-like language called ‘Solidity’. The language invented to allow codification of human-understandable logic into programming language format understandable by ‘Ethereum Virtual Machine (EVM)’ named ‘JSON-RPC’. EVM represents a computational resource that shares amongst Ethereum network which allows machines with different environments to interact with Ethereum Blockchain without the need for specific computational environment or hardware. This allows Ethereum network to formed by wide variety of machines with different operation system and internal environment. At the same time, Ethereum Blockchain can adopt variety of consensus mechanisms. The main Ethereum Blockchain initially adopted PoW. Due to limitation as it requires huge amount of computational resources to stay active, Ethereum network later forked the Blockchain line into several chain lines with different consensus. i.e., Proof of Stake and PBFT which adopt voting-like mechanism to allow reduction of computational resources consumption. As time passed, Ethereum community keeps on growing, now there are wide variety of consensus mechanism proposed to suit with different application and network environment.

# METHOD

This section describes a method of how we implement XDS.b Profile based on Blockchain technology. It starts with a general use case scenario following with top view of the Blockchain network and consecutively narrowing into the design of XDS Blockchain, its components, and Smart-contract which took the main role to adapt the profile into the chain.

## A Use Case Scenario

Mr. Peter Parker was bitten by a strange spider. Mr. Parker then visited hospital A seeking for medical advice. A doctor at hospital A examined the wound and saw no sign of infected bite wound on Mr. Parker. The doctor just cleaned and dressed the bite wound and provide required medications. The encounter event and diagnosis result were recorded within hospital A. Later, during his travel to another city, Mr. Parker felt quite painful from the bitten area and forgot to bring the given medications with him. He then visited hospital B for further consultation. With health information sharing implemented between the two hospitals, the doctor at Hospital B could see the care record and diagnosis result from hospital A. With IHE XDS Blockchain implemented, even hospital B hit by ransomware and cannot access to a local computer, a doctor at hospital B can use an alternate computer to access the XDS Blockchain ledger and discover that there is the latest diagnosis result for Mr. Parker available at hospital A. The doctor could gain access to the diagnosis result and able to identify that Mr. Parker was afflicted by unknown genetic mutated symptom caused by a spider bite. Allowing the doctor to provide appropriate treatments and prevent further harmful consequences.

## Conceptual Design

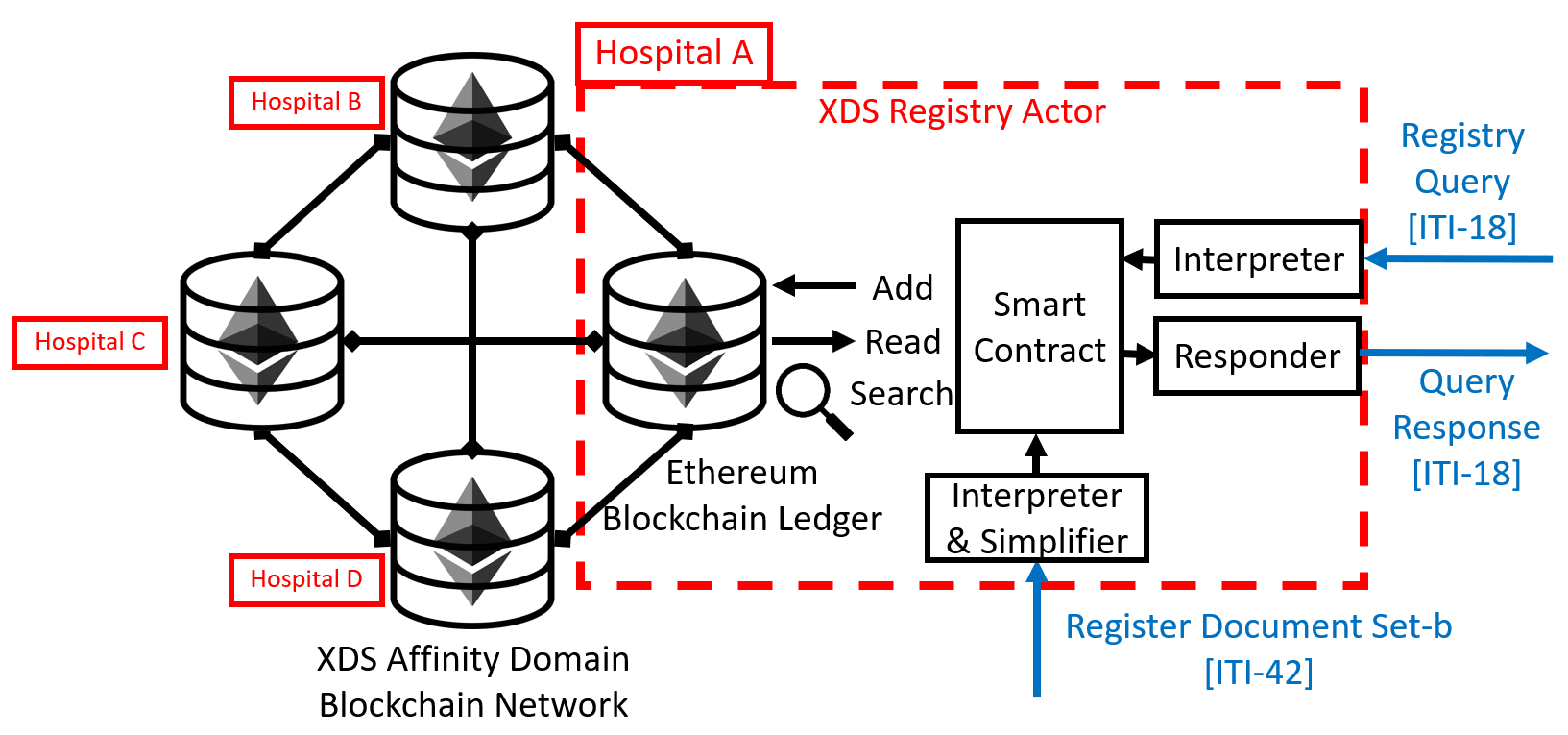
The unique nature of the healthcare information system and its environment emphasizes confidentiality of data that limits full use of Blockchain technology. Patient data cannot be put directly into Blockchain as it will become persistent following Blockchain characteristics while increasing difficulty in ensuring data confidentiality when its replica is distributed all over the network. Instead of risking confidentiality of healthcare data by publishing it directly into the Blockchain network, we propose using IHE XDS.b Document Registry Actor to act as a health document exchange medium for the Blockchain network. The profile is best compatible with Blockchain technology as decentralization will secure the availability of the health information exchange by eliminating the need for the organization that will act as the central hub for the exchange avoiding a single point of failure problem. At the same time, there is no need to publish health documents directly into the distributed network, reduce the risk against the data leakage. Additionally, we further extend the usability of the profile by allowing the organization that has shared health documents from its source to also act as an additional data backup for the original by providing additional access points (URLs) for the document. That means even the source of health document becomes unavailable due to unpredictable circumstances like a cyber-incident (i.e., ransomware threat), the network will still have a chance to access the compromised document via an alternative source available from a shared peer. This extends the benefit of the health document sharing network and encourages the growth of the health document sharing community even further.

## Blockchain Design

Considering the environment of the XDS Affinity domain network, the network is comprised of members who are hospital or health institution that entered the network intending to share their health document and access health document shared from other. Each member entering the network are expected to have been selected by the network and have an agreement with the network to voluntarily share health document from their XDS Document Repository Actor to the network while not exposing information shared within the network to the outside. Following Blockchain Types mentioned in the background knowledge, the network is not suitable with the Public Chain as they are not accepting anyone into the network without pre-selection and a proper agreement. At the same time, they are also not suitable with the Private Chain as the XDS Affinity domain network was designed for scalability and not fixed to any specific organization. That means the XDS Affinity domain network is best suitable with the Permissioned Chain type [34] as it was designed for scalability welcoming more members to join the network over time under the condition that joining members are accepted by the network and have a proper data sharing agreement.

Meanwhile, following the Blockchain consensus mentioned in the background knowledge, the XDS Affinity domain members were not pre-determined to invest a high amount of computational resources for entering the network and were not expected to gain direct profit from participation in maintaining the network. That means the consensus mechanism with computational resources inefficiencies like competition-based and profit-dependence mechanism like randomness-based are not the choice of consensus for the network. Then the remaining majority-based consensus will be the most suitable choice available. The starting point of majority-based consensus will be PBFT as the basis of the type. With the nature of the XDS Affinity domain network which pre-determined the joining member, this should prevent the chance for the bad actor to control the majority and secure Blockchain characteristics for the network consecutively.

Figure 2 Integrating Blockchain into XDS.b Profile



## Integrating Blockchain with XDS.b Profile

We exclude the ITI-61 transaction from the implementation because it is unnecessary for the current usage of the work. The patient identification was assumed to be already standardized amongst all members of the network, eliminates the need for the ITI-44 transaction from consideration. This left the XDS Document Registry Actor Blockchain to only interact with the remaining XDS Document Repository Actor via ITI-42 transaction and XDS Document Consumer Actor via ITI-18 transaction.

Each Blockchain node will receive ITI-42 transactions from their local XDS Document Repository Actor as a normal XDS Document Registry Actor would do, before transitioning the transaction into a Blockchain Smartcontract and publishing it into the Blockchain ledger. Likewise, the health document query via ITI-18 transactions from local XDS Document Consumer will be interpreted and interact with SmartContract consecutively. Note that the XDS Document Consumer Actor will still be required to directly issue ITI-43 transaction to the XDS Document Repository Actor hosting the health document to retrieve it. The SmartContract will act as a medium for each node to perform the task to add data, read data, and search for data within the Blockchain ledger (as shown in Figure 2) allow the Blockchain technology to effectively integrated into the XDS system.

## Design Functions of Document Register

This is where XDS Document Registry Actor registers health document metadata set within ITI-42 transaction received from XDS Document Repository into the Blockchain ledger. This function enables sharing of health documents to the XDS Affinity Domain Blockchain network as well as allowing the shared document to be registered as an alternative source. There are two SmartContract functions related. The first function is the document registering function, the function acts as part of the XDS Document Registry Actor to store the value of health document metadata interprets from ITI-42 transaction into the Blockchain ledger. This function acts in a fashion like a programming variable where certain values were assigned to a specific variable for usage within the program, a whole set of metadata value is assigned into single SmartContract function transaction. The function also automatically assigns each set of metadata with an identification number to be used for common understanding amongst the SmartContract to differentiate each set of metadata belong to each health document. The identification number also essential for search operation which would be further explained later. The second function is the checker function which will check for the last identification number assigned to the published set of metadata. This function allows SmartContract to keep track of the identification number previously used and prevent duplication. Each time a new set of metadata entering the Blockchain ledger, the identification number which would be assigned to the metadata set will be additively increased by one from the previous.

# IMPLEMENTATION

This section describes our implementation to demonstrate the implemented concept mentioned previously.

## Implementation Goal

Help reader, what we are going to do

To achieve the design mentioned in Section IV, we need a computing environment prepared for programming, creating smartcontracts, and maintaining a blockchain ledger for testing. Then, begin by developing XDS Actors following the requirements specified in the IHE IT Infrastructure (ITI) Technical Framework (TF). As mentioned in Section IV, our scope only focuses on XDS Document Repository Actor, XDS Document Registry Actor, and XDS Document Consumer Actor. Our main contribution is to the Registry Actor, which will be overhauled to work with Blockchain while still having its behavior follow the requirements specified in the IHE ITI TF. The Repository Actor and Consumer Actor will only be a program created for concept demonstration, mimicking the behavior of the actual actors working in a hospital information system.

Why use Javascript and Node.js

As we have seen from HL7 and FHIR, current healthcare information exchange standards are mostly web-based protocols. They require the capability to handle a huge number of transactions in a limited amount of time. That means the programming language for this type of work should be compatible with web-based environments and be able to handle multitasking and thread simultaneously. With its asynchronous nature and compatibility with website integration, Javascript is the choice for our implementation of this work. We further adopt the "Node.js" variant of Javascript for our programming environment as it was designed to build scalable network applications that handle many connections concurrently. Furthermore, Node.js also provides simple access to community-made node modules, which offer a wide variety of useful APIs for software development, which may reduce the difficulty of our implementation further.

All actors within the IHE XDS Profile communicate with each other using XML message transactions. As we utilize Javascript as our main programming language for the implementation, these XML messages need to be interpreted into programming objects to allow simpler handling methods within the program. So, we prefer Javascript Object Notation (JSON), which is a lightweight data-exchange format for programming objects that was created specifically for this purpose. It is simple to read and write for humans, and it is simple to parse or generate for machines. The JSON format will have a major impact on reducing complexity for the implementation of the XDS Document Registry actor program with Blockchain smartcontracts.

## Environment Preparation for Development

Machine and VM preparation

The machine hosting the development must have the specifications required to host all the software shown in Table 3. At the same time, the blockchain ledger can consume a significant amount of space and increase even further over time. That means the host machine must have significant space available and can also be later increased when needed. Additionally, in this work, we demonstrate the result of our implementation by using the Quorum 7-Node Example, which is like a simulation of seven machines acting as a Quorum Blockchain node at once. That means the host machine should have a significant amount of memory (RAM). During the implementation, we discovered that 8 GB of RAM is barely enough to support seven nodes, each executing their function at once.

Table 3 List of software and programming modules required for implementation.

|  |  |
| --- | --- |
| Software/Module | Role |
| Go-Ethereum (Geth) | - Blockchain engine software for Ethereum-based Blockchain.  - Control, access, and maintain the local Blockchain node's activity.  - Connect and sync the blockchain ledger in the local node with the blockchain network via the devp2p protocol. |
| Quorum | - An Ethereum Blockchain variant that provide Blockchain environment alternative to the main Ethereum Blockchain which support consensus mechanisms other than PoW and PoS. |
| Quorum’s  7-Nodes Example | - A virtual machine-like environment for testing Ethereum-based smartcontracts in a simulated Quorum Blockchain with 7 active nodes.  - Execute the programmed behavior specified in the smartcontract. |
| Remix IDE | - An interactive development environment (IDE) for the Solidity programming language.  - Use for the creation of smartcontracts.  - Compiled smartcontracts give ABI code and byte code which will be used with Javascript (or other) programs. |
| Node.js | - Compile and run Javascript code.  - Node.js also provides "npm", which is built-in software that helps manage node modules API. |
| xml2js | - A node module for Node.js which helps convert XML messages into JSON format. |
| web3js | - A Node.js node module that allows Javascript programs to interact with smartcontracts. |

Geth Installation Source

The Geth client is available on Github [43]. Geth can be installed as a standalone or included in the installation package of the Ethereum platform variant or other kinds of services interacting with the Ethereum blockchain (i.e., the Ethereum Wallet Client, “Mist”). Geth's interface was initially designed to operate on the Linux OS and later extended compatibility to other OS via the Linux console simulated platform or work under the graphic user interface of another client. The Geth client has no specific system requirements as it is only a set of Golang scripts that have no restrictions on any system, but as it is mostly integrated within another client, it requires the machine to be compatible with the main client for installation.

‘How to’ for Geth in this implementation

In this implementation, the test machine has Geth globally installed by installing Quorum. Geth can be installed as a standalone to ready the machine for working with the Ethereum Blockchain using a Linux installation from a source repository command, i.e., "apt-get install ethereum" in Ubuntu. The installation instructions can be located at their website [44]. All available command lines for Geth can be located at their repository on Github [45]. The most commonly used command is "geth attach <IPC path or link>", which is required for accessing the Geth console of each active node. The console is where the user can input the command line to directly control the behavior of each Ethereum node.

Introduce Quorum, Ethereum Blockchain variant

Quorum is an Ethereum-forked that allows the blockchain to adopt a consensus mechanism other than PoW and PoS, which default to the Ethereum Blockchain. Quorum's source code and installation package can be accessed at their repository on Github [46]. The installation method for the latest stable release is located in the repository [47]. The platform was designed to specifically operate with a Linux-based interface and can be compatible with non-Linux OSes with the aid of third-party software as a medium. Other than that, the platform has no specific system requirements for installation. However, from the tests during the implementation, it is recommended that the machine running Quorum should have more than 6 GB of available RAM. Otherwise, there will be a performance issue that occurred during the run. In this implementation, as the test machine runs on Ubuntu 18.10, there is no other 3rd party software required to operate Quorum. The Geth client was included in the Quorum installation package, which means a developer can immediately start their Quorum Blockchain development right after the installation.

Introduce Quorum 7-Nodes Example

Other than the Blockchain platform, Quorum also offers the "7-Nodes Example" for developers to locally deploy on their machines to test the functionality and performance of Smartcontracts during development. The source code can be cloned directly from its repository available on Github [48]. The "7-Nodes" will simulate seven Blockchain nodes in the host machine in a similar fashion to a virtual machine using the required libraries called "Tessera" [49] and "Constellation" [50]. They are included in the package as a running engine. The source code can be initiated, activated, and simultaneously controlled using the control script provided within the example. Each node can be accessed using the Geth client. For the implementation, the control scripts provided within the example were further modified to be compatible with our usage.

How 7-Nodes modified for this implementation

The Blockchain initiation script was modified to be able to re-initiate the entire blockchain by deleting the existing chain and replacing it with an empty one. This allows a reset of the published blockchain during the development. The transaction publishing script was modified to run other specific code developed for the XDS Document Registry Actor and its Smartcontract allows testing and running of XDS Document Registry-related code on the 7-Nodes. It must be noted that transaction-related scripts only operate on specified single nodes amongst the seven, not the entire set of seven nodes. That means the activity of each node is independent. All initial configuration instructions are available in the Quorum 7-Nodes example repository [51]. Primarily, it is required to configure the genesis block for the blockchain ledger and issue the initiation command using the script provided. Each script is specific to each consensus mechanism. In this implementation, we only use the script "istanbul-init.sh" (execute using Linux Bash syntax) to initiate the genesis block for the blockchain ledger, as we are going to use IBFT as its consensus mechanism. The script will generate an "istanbul-genesis.json" file as a configuration script for the genesis block. Then the activation of the IBFT Blockchain can be done using the "istanbul-start.sh" script, which will start the activation of all seven nodes and bring the 7-Node Blockchain network to life. It must be noted that this activation process may take several minutes.

Addition from above

For this implementation, we further created a script to reduce the complexity of the initiation and activation processes of the 7-Node Blockchain. The script "rebirth.sh" will delete all data of an existing set of 7-node blockchains and initiate a new genesis file to simplify the reset of the blockchain during the development of the implementation. The script "runmy7nodes.sh" will issue the whole command line required for activating the 7-Node Blockchain, simplifying the Blockchain activation process. These scripts will help reduce the complexity of command-line manipulation during the implementation.

Dev. environment for Ethereum Smartcontract

For Smartcontract programming, Ethereum provides a web-based IDE for the Solidity language that can compile, test, and deploy Smartcontracts to a specific Ethereum node called "Remix" [52]. In the implementation, Remix was accessed using Google Chrome from the main machine and the session was saved locally using the Remix client to avoid unexpected issues in the Solidity code. Each Smartcontract is validated and tested within the IDE before actual implementation on the test machine. The Smartcontract ready for implementation will be compiled using the Solidity compiler provided by the Remix community-based plugin. Each compiled solidity code gives an ABI code and a Binary code which will be used on Web3JS code to interact with the Blockchain Smartcontract. After successfully compiling the Solidity Smartcontract code, the ABI code and Byte code will be automatically generated by the IDE. These codes can be copied and passed into Web3JS code by assigning a variable to store the code as its value. It should be noted that byte code defines the behavior of the smartcontract and is only required on the first deployment of the smartcontract, which is required only once in this implementation, while ABI code defines the interface for communicating with the smartcontract and is always required every time the native program communicates with the smartcontract.

Smartcontract preparation before use

Before beginning registering health document metadata into a Blockchain Smartcontract, the Smartcontract must be first deployed to act as a contract format for the entire Blockchain ledger for the implementation. The Smartcontract is simply deployed into the Blockchain ledger by the Web3js script. It required both the byte code and the ABI code received from the Solidity compiler to be completed. Once the smartcontract is deployed, any later communication with the smartcontract will only need ABI code to act as an interface for the communication. This Smartcontract deployment process is only required once at the initiation of the Blockchain ledger and no longer needs to be performed ever again for the rest of the Blockchain ledger life cycle.

Connecting another program to Smartcontract

The coding environment must be provided with essential coding components for the coding process of all non-Blockchain native Javascript programs. In this implementation, we use NodeJS as a compiler and coding environment for all JavaScript programs. NodeJS is available on their website [53]. The essential coding components of the node module can be installed using the Node Package Manager (NPM), which comes together with NodeJS [54]. The coding of the program for the native side of XDS Actor requires node module names "Web3" (Web3js) [55], "xml2js" [56], "fs" [57], "net" [58], "util" [59], "moment" [60], and "cryptr" [61], which can be installed using the command-line.

To connect our program to an Ethereum smart contract, we can use the Ethereum API tools, which is Web3, as a middle. Web3 allows smart contract control through a preferred programming language and transitions logic and variables from the language to Solidity. Web3 provides a programming API for Javascript called "Web3JS," which allows the Javascript program to interact with Ethereum-based Smartcontracts. The API can be accessed using the node module provided via Node.js.

## Implementing XDS Actors for XDS Blockchain

XDS Document Repository

For this work, the XDS Document Repository program was created to read ITI-42 transactions prepared for demonstration. The Repository program then sends the transaction read to the local XDS Document Registry Actor.

- Detail for XDS Document Repository –

Pseudocode of XDS Repository

|  |  |
| --- | --- |
| **Input:** Trigger XDS Document Register  **Output:** New document registered into XDS Document Registry Actor | |
| 1: | If Trigger XDS Document Register = TRUE |
| 2: | DocNum = Prompt Input (Integer) |
| 3: | New ITI-42 Transaction = Read File (DocNum) |
| 4: | Send ITI-42 to XDS Document Registry Actor |
| 5: | Wait for response |
| 6: | If Ack Received = TRUE { |
| 7: | } |
| 8: | End |

The XDS Document Consumer Actor program is composed of two major parts. The first part is a user interface where the program prompts the user for input specifying the query type and essential search keywords for document discovery. The second part is the ITI-18 transaction generator. When the query type and all the search keywords the user wants are set, the XDS Document Consumer Actor program then uses these inputs to generate an ITI-18 transaction.

Implementation UI for XDS Consumer

For this implementation, we design that the user interface will take the form of a command-line program that can be run via Windows command prompt or Linux terminal. The program will prompt the user to specify search type, including META-attributes value, and specify the value. The actor then accepts these values to create an XML message following ITI-18 format before sending it to a local or accessible XDS Document Registry actor to query for matching document and start search operation. (JS part how to)

How XDS Consumer UI Work

The command-line interface of XDS Document Consumer begins with prompt the user to input registry query types or choose to quit the program. The user will need to specify digit numbers corresponding to the choice. Then, the user will be prompted to input essential metadata attributes required for the query type (i.e., FindDocuments will require attributes included DocumentEntryPatientId and DocumentEntryStatus) before prompt to input other metadata attributes as optional depending on the information the user known. When there are no more metadata attribute values to add, the user can choose to start a query for the document.

Inspecting ITI-18

(It should not be here, will move to XDS Consumer)

XML code of RegistryStoredQueryRequest [ITI-18] transaction sample composes of three main sections. The first section labeled “query:AdhocQueryRequest” is where XML schematic information are located and the label also act as marker which tell interpreter program to recognize it as ITI-18 transaction. The second section labeled “query:ResponseOption” mark the expected format of query result that will return to Document Consumer. The third section starts from label “rim:AdhocQuery” contain all search keywords issued by Document Consumer. These search keywords are selected Metadata attributes and its value.

Deal with creating ITI-18 transaction

The XDS Document Consumer Actor program will then accept the input and assort it into the ITI-18 transaction before sending it to XDS Document Registry Actor. After the query has been sent, the XDS Document Consumer will wait for the response from XDS Document Registry Actor. When the response is received, the XDS Document Consumer then shows the metadata attributes the value of the query result in the terminal or just terminates the program if there is no matched result registered.

Pseudocode of XDS Consumer

|  |  |
| --- | --- |
| **Input:** Trigger XDS Document Query  **Output:** XDS Consumer receive Search Result from  XDS Registry | |
| 1: | If Trigger XDS Document Query = TRUE |
| 2: | SearchType = Prompt Input (Integer) |
| 3: | for (Loop until All Keywords Specified; i++) { |
| 4: | METADataAttributeType = Prompt Input (Integer) |
| 5: | SearchKeywords[i] = Prompt Input (String) |
| 6: | } |
| 7: | New ITI-18 Transaction (SearchType) |
| 8: | Add value in ITI-18 Transaction (SearchKeywords) |
| 9: | Send ITI-18 to XDS Document Registry Actor |
| 10: | Wait for response |
| 11: | If Query Response Received = TRUE { |
| 12: | Extract (Query Response) |
| 13: | Read Search Result |
| 14: | Display Search Result |
| 15: | } |
| 16: | End |

0. XDS Registry Actor

0.1 XDS Registry Intro.

0.1.1

The XDS Document Registry Actor program must be able to communicate with the simulated XDS Document Repository actor and XDS Document Consumer actor. At the same time, the software will need to act as an interface for XDS system to interact with the Blockchain ledger. The actor will wait until after receiving the XML message transaction and react differently to ITI-42 and ITI-18 transactions.

0.1.2 Behavior of XDS Registry

Unlike the XDS Document Repository Actor and XDS Document Consumer Actor, the XDS Document Registry was made to act as the medium between the native system and the Blockchain Smartcontract. This section will break down the component of the XDS Document Registry Actor in a pattern different from the rest where it separated by its main function interacting with health document metadata which included Document Registering and Document Search. Each section will break down into JavaScript native program part and Smartcontract part. For this implementation, the XDS Document Registry Actor will open a TCP connection to receive the transaction on a specified port. Upon receiving the ITI-42 transaction, the Actor then converts the XML message into JSON using xml2js. When ITI-42 was interpreted into JSON, the actor then assorts the object and sorts it into the Smartcontract compatible format.

Pseudocode of XDS Registry

|  |  |
| --- | --- |
| **Input:** Received ITI-42 Transaction or Received ITI-18 Transaction  **Output:** Response depend on transaction type | |
| 1: | If Received ITI-42 Transaction = TRUE { |
| 2: | XML = Read ITI-42 Transaction |
| 3: | META JSON = Convert XML to JSON |
| 4: | META String = Convert XML to String |
| 5: |  |
| 6: | If META JSON validity = TRUE { |
| 7: | META Data = Assort META JSON |
| 8: | for (Loop on all attributes; i++) { |
| 9: | META Value [i] = META Data[i] to String |
| 10: | } |
| 11: | Add META Value to Smartcontract |
| 12: | } |
| 13: |  |
| 14: | If META String validity = TRUE { |
| 15: | Add META String to Smartcontract |
| 16: | } |
| 17: |  |
| 18: | Publish Smartcontract into Blockchain |
| 19 | Response ACK to XDS Repository Actor |
| 20: | } |
| 21: | End |
| 22: |  |
| 23: | If Received ITI-18 Transaction = TRUE { |
| 24: | XML = Read ITI-18 Transaction = TRUE { |
| 25: | Query JSON = Convert XML to JSON |
| 26: | Search Type = Read from Query JSON |
| 27: | Search Keywords Set = Read from Query JSON |
| 28: |  |
| 29: | for (Loop on registered document smartcontract) { |
| 30: | for (Loop on all Search Keywords; i++) { |
| 31: | if (META Value = Search Keyword) { |
| 32: | Matched Value |
| 33: | } |
| 34: | } |
| 35: | if (All Search Keyword Matched = TRUE) { |
| 36: | Document Found for Search Result |
| 37: | Search Result = META String of this document |
| 38: | } |
| 39: | } |
| 40: |  |
| 41: | New ITI-18 Response Transaction as JSON |
| 42: | Add Search Result to ITI-18 Response JSON |
| 43: | Convert ITI-18 Response JSON to XML |
| 44: | Send ITI-18 Response to XDS Consumer Actor |
| 45: | } |
| 46: | End |

0.2 XDS Registry interaction with XDS Repository

0.2.1

ITI-42 response side + flow

If the Document Registry Actor successfully received the transaction, they must return response. The response transaction included only XML schematic information, message UUID number, and status type “successful”. This response will let the Repository finish its process and end messaging attempt. For document registering, XDS Document Repository Actor register document Metadata attributes into XDS Document Registry Actor using IHE ITI-42 transaction. XDS Document Registry Actor then interprets the transaction into a programmable object before check if the transaction is ITI-42. Then, the actor proceeds to pass the retrieved object into Blockchain Smartcontract and publish it into a Blockchain ledger.

0.2.2 Interpret ITI-42 transaction

0.2.3 Registering document into Blockchain (before smartcontract)

0.2.4 How Smartcontract made for XDS Registry

(Maybe move to or move from Smartcontract section)

In the implementation, Smartcontract was designed to store string values and will return the stored value when called by the corresponding Smartcontract function. The prepared JSON must be converted into a string variable before entering Smartcontract. This is due to the limit of the Ethereum Smartcontract which can cover a limited number of programming variables so, we simplify our program to avoid that limit by storing the whole JSON in string form as a single variable. Because the Ethereum Blockchain requires a particular amount of gas to run the Smartcontract, the length of the variable could produce an error if there isn't enough gas available. That means, we need to increase the limit amount of gas for executing Smartcontract from the default value. Although this change may not affect the implementation, it may affect the network where its member prefers to use actual cryptocurrency like Ether to maintain Blockchain where an increase in required gas may accelerate the depletion of currency circulating in the network and severely reduce the maintainability of the Blockchain. By the Smartcontract design, XDS Document Registry actors can keep Metadata attributes of each document by storing them as JSON string variable inside Blockchain using one Smartcontract per document. At the same time, the actor can perform search operation by sequentially call upon each published Smartcontract one-by-one until the result was found or until the last in the case which no matching result. It must be noted that publishing of Smartcontract always requires gas to execute the task while calling for the variable value stored in the Smartcontract is not consuming any Blockchain resource. It must be noted that at the beginning of the XDS Blockchain network, the base Smartcontract must be initially deployed into the Blockchain to allow further registration of health documents for the rest of the Blockchain ledger.

0.2.5 Smartcontract behavior

0.2.6 Smartcontract programming structure

0.2.7 Transact and Block validation

0.3 XDS Registry interaction with XDS Consumer

0.3.1 XDS Registry handling ITI-18

Like IHE ITI-42 transaction handling, the XDS Document Registry actors also wait for ITI-18 on the TCP channel. The received transaction will be converted into JSON while matching UUID with its corresponding metadata attributes. The transaction is specified with the header “QueryResponse” and compose of Metadata attributes value input by Document Consumer. These values will be used in search operation which will seek for the Smartcontract with matching metadata attribute values. After the result was found, the actor then proceeds to create a response XML message following the ITI-18 format.

0.3.2 Extract ITI-18

0.3.3

Begin Search operation at XDS Consumer

providing search operation type and some Metadata attributes value as search keyword via ITI-18 transaction. XDS Document Registry will check if the transaction is ITI-18 before performing search operation matching specified search type using provided keyword Metadata attributes value.

0.3.4 Explain search operation in detail

The search operation will be performed by consequently call for each registered Smartcontracts until all contracts with matched attributes value were found. XDS Document Registry Actor then returns all query result in XML format following specification for ITI-18 responding. Upon receiving the query response, XDS Document Consumer then interprets the transaction and displays the result to the user in a human-understandable format.

0.3.5

.

0.3.6 ITI-18 query response.

When Document Registry Actor received the transaction, they will use search keyword provided to search for registry with matched Metadata attribute values. Then it returns the result to Document Consumer Actor as response transaction. With header labeled “query:AdhocQueryResponse”, the transaction contains search result depend on query type specified in ITI-18 transaction. If the query expected for “LeafClass” as result, the response would return Metadata attributes of all matched result in detailed. Otherwise, if the query expected for “ObjectList”, the response would return object reference number of all matched results. These two types of response specifically selected depend on search behavior of Document Consumer Actor’s user. The query which specified “LeafClass” as its search result must provide keyword which unique to its corresponding document, such as document unique ID or object reference UUID. At the same time, “ObjectList” are used to search for wide range of document with generic search keyword and value where discovery of document existent is the main goal.

## Implementation Result

(Kind of wrap up, what we have done so far)

Implemented system demo.

The Blockchain network environment for the implementation will be demonstrated using the 7-Nodes Example Blockchain. That means there will be seven IBFT Blockchain nodes simulated within the environment of the test machine. The test sample will be monitored from node number one (which would later refer as “1st Node”). To indicate that the implementation has the compatibility to operate with XDS Actors in a common XDS system, we created transaction samples based on the example provided by the IHE ITI framework. Transaction samples provided by the framework are including ITI-42 Register Document Set-b transaction, ITI-18 Registry Stored Query transaction and its corresponding response transaction. However, there is much more specification provided in the framework that far exceeds the element provided within the example. There may consist of an element that differs from the transaction used in the actual system which may cause an error and need further adjustment in the actual deployment. The main group of XDS Actors in the implementation are XDS Document Registry Actor, XDS Document Repository Actor, and XDS Document Consumer Actor. All files corresponding to the implementation must be included within the directory that contains the 7-Node Examples. The XDS Document Registry Actor is the actor that acts as the medium between the XDS system and the Blockchain network. At the same time, the main function and process triggered in the implementation including Document Register and Document Search (Document Query) are centered around the XDS Document Registry. The XDS Document Registry will always active and standby for incoming transactions.

Explain how XDS Repository & Registry work

The XDS Document Repository Actor is the actor that triggers the Document Register function registering new health document metadata attributes set into the Blockchain ledger. The Document Register function is triggered once the XDS Document Repository Actor program was called to register new health document metadata set into the XDS Affinity Domain network. The XDS Document Registry Actor then received the Document Register transaction and proceeds to broadcast the transaction into the Blockchain network publishing the transaction into the Blockchain ledger through a Block validation process. After the transaction was successfully published into the Blockchain ledger, the XDS Document Registry Actor response back to the Document Repository Actor to report that the process was completed which terminating the XDS Document Repository Actor program.

Explain how XDS Consumer work

The XDS Document Consumer Actor is the actor that triggers the Document Query function receiving user input search keyword values and asks the XDS Document Registry Actor to search for an existing registered document with the matching metadata attributes value as specified in the search keywords. Once called, the XDS Document Consumer Actor program will prompt the user for document query type then prompt for essential search keyword values and followed by the addition of optional search keyword values. After all search keyword values from the user were set, the XDS Document Consumer Actor program then proceeds to assort the input into the ITI-18 transaction format and send it to the XDS Document Registry Actor. Upon receiving the transaction, the XDS Document Registry Actor program then uses the provided search keywords to search the Blockchain ledger for the metadata set with a matching value. Whether the matching result was found or not, after searching on all registered metadata set, the XDS Document Registry Actor then response search result to the XDS Document Consumer Actor. The XDS Document Consumer Actor then interprets the response and displays the search result to the user.

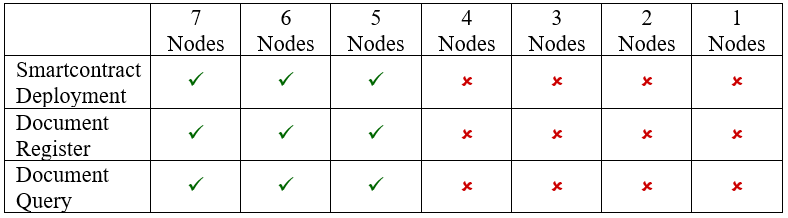


Table 5 Functionalities experiment result

# evaluation

 The goal of the evaluation is to test the functionalities and the performance of the implemented system. This includes the Document Register function and Document Query function. The evaluation will indicate the compatibility of the implemented system to an actual healthcare operation environment. The implemented system should be able to sustain a huge amount of data that continuously flows through the system without a failure. Each setup will be tested with the mockup transactions and measure the processing time required for the system to complete the specific process. There are ten transactions created for the experiment as test samples. In each transaction have its metadata attributes modified varied in each transaction for the test as. Each transaction will have about the same file size but different metadata attributes values to prove functionalities of the Document Query function. All experiments will be tested with these transaction samples resulting as 10 times test for each setup ("transaction number #n" will be referred to as "Document#n").

## Functionalities Test

 The main functions of the implemented system are Document Register and Document Query. This setup will test the functionalities of each function under the normal circumstance where there are seven active Blockchain nodes and there is one node performing the task. The criteria to check if the Document Register function correctly is the registered document must be discoverable within the Blockchain ledger when its SmartContract was called. This SmartContract must return metadata attribute values of its corresponding document when it was called by the system. The XDS Document Registry Actor signal that it successfully published the metadata attributes set into the Blockchain ledger using SmartContract. When the published SmartContract was called, it returns the value of metadata attributes set. That means the metadata attributes set of “Document01” is now live on the Blockchain ledger. This proves that the Document Register is functions correctly.

The criteria to check if the Document Register function correctly is when users input corrected search keywords that match one of the documents registered within the Blockchain ledger, the XDS Document Registry actor must be able to find the matching document and return the search result to the XDS Document Consumer Actor to display it to the user correctly. The XDS Document Consumer user interface that prompts the user to input search keywords matching the document, they are seeking and also show the list of search keywords that match with the Document01. The XDS Document Registry Actor received the query from the XDS Document Consumer Actor and initiate search program. When the XDS Document Consumer found the matching document as the search keywords provided, the XDS Document Registry Actor then returns the search result to the XDS Document Consumer Actor. After the XDS Document Consumer Actor received the search result returned from the XDS Document Registry Actor and display the result to the user. In this case, the result show metadata attributes set for the Document01 correctly. This proves that the Document Query is functioning correctly.

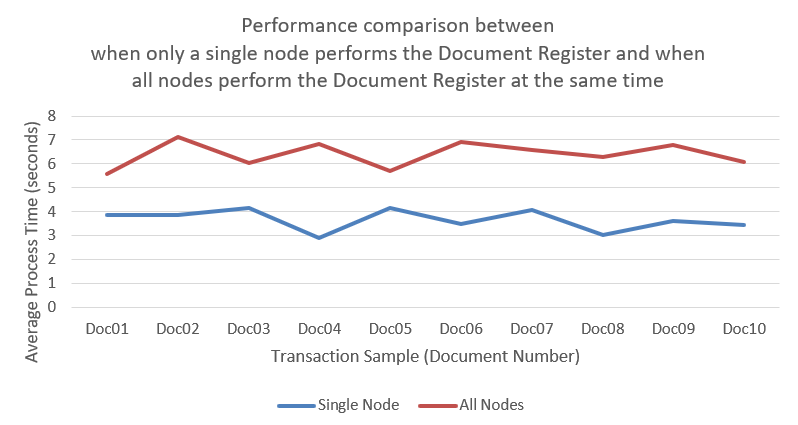


Figure 6 Performance comparison of the system performing the Document Register function between when performed by only a single node and

On the hypothesis that a changed number of active nodes may cause a change in the functionalities of the XDS Blockchain, to test the functionalities, the experiment performs by varying the number of nodes and test if the system could perform any function. Start from the setup with seven active nodes as the reference, followed by six, five, four, three, two, and one active node for the variation. To create a situation where there are fewer than seven active nodes, the 7-Nodes Example configuration must be modified to meet the situation. In Quorum 7-Nodes Example, the number of active nodes was defined in the "permissioned-nodes.json" file. Each active node represents by its "enode" identifier number together with its communicating TCP port number. The node will become inactive if its enode number was excluded from the file. That means, in each setup for a specific number of nodes, enode number of inactive nodes must be excluded from the file before initiate the 7-Nodes Blockchain network. At the same time, the "numNodes" variable value in the "istanbul-init.sh" file must be set to the specific number of active nodes (i.e., numNodes = 6 for six active nodes) to declare that there will be file directory prepared for the specified number of nodes at the initiation of the 7-Node Blockchain network. After the configuration was modified, the 7-Nodes Example will automatically generate Blockchain nodes to meet with the specified number upon the initiation of the Blockchain ledger. That means every time the configuration file was modified for a new setup, the Blockchain network must be reset and re-initiated to ensure that there is no conflict with another setup which could affect the accuracy of the experiment result.

Table 2 shows that the system result from the implementation can function normally with 7, 6, and 5 active nodes. The system stops functioning when there are active nodes lesser than 5. After looking into the root cause of the result, it is turn out that the 7-Nodes Example cannot resolve the Block validation process cycle in a situation with fewer than 5 active nodes. When investigate the log, it shows that the Block validation process cycle which would repeat continuously and endlessly simply stopped at the first Block and there is no further upcoming process appeared in the log or none of any error notification appeared. Noteworthy, this is not even related to the condition of IBFT consensus where it requires at least two-third of all nodes to vote for the same Block version to complete the validation process. The system should be able to operate normally without error even there are fewer than five active nodes when all active nodes agreeing on the same Block version and there is no "bad actor node" present in the system to propose a falsified Block version that interrupts the vote. We think that a possible reason could be that the 7-Nodes example may not be developed for the situation with 4,3,2, or 1 active node as it was only built to aid Smartcontract developers to easily deploy and test their SmartContract in the proper 7 active nodes simulated environment closely similar to the actual Blockchain network.

It also must be noted that it is possible for the case that contains an even number of nodes may cause failure in Block validation attempt due to the majority of Block approval from nodes not meet the least requirement of two-third of all nodes in the network. For example, if ten nodes are participating, five nodes proposed Block named 'ABC' while another five nodes proposed Block named 'CBA'. This example will result in failed Block validation attempt. If this happened, the network was expected to try another attempt until the majority meet the least requirement of two-third and all the remaining transactions were successfully added into the valid Block and published into the Blockchain ledger. However, the evaluation of the implementation will not cover the case due to complications in creating the “bad actor” node. We assume that the feature was not directly included with Ethereum, Quorum, or its 7-Nodes Example as it was not created for a malicious purpose so, it is too difficult for us to create the situation that half of the participate nodes attempt to propose falsified Block for the evaluation.

## Performance Test

 On the hypothesis that a changed number of active nodes or change in the amount of XDS transactions entering the system may cause a change in the performance of the XDS Blockchain, the performance test will be performed by triggering the Document Register function and/or the Document Query function depends on the setup. The prepared transaction samples will be tested on the setup with a specific number of nodes then measure the time the system took to complete the process. Additionally, to increase the accuracy of experiment results and reduce the effect from other factors outside the consideration, the experiment will be repeated ten times on each transaction sample. The final performance result of each transaction will be derived from the average values of these ten times repeated.

Test if the number of transactions would cause a change in performance of the XDS Document Registry to perform the Document Register function. This setup is performed by trigger the XDS Document Repository Actor to start the Document Register function which will send the ITI-42 transaction containing metadata attributes set of the selected document (Doc01 - Doc10) to the XDS Document Registry Actor. The processing time the XDS Document Registry took to complete the process publishing the metadata attributes set into the Blockchain ledger will be measured for the experiment result. The "Single Node" line was the averaged processing time monitored at the 1st Node performing the Document Register function when there is only one node performing the function. The "All Nodes" line was the average processing time monitored at the 1st Node performing the Document Register function when all active nodes performed the function within the same time. Both setups measure the processing time the node used to complete the Document Register function since the moment it received the ITI-42 transaction from the XDS Document Repository actor. Figure 5 shows that when all nodes perform the Document Register function at the same time, it took a longer time to complete the function when compared to the situation when there is only a single node perform the Document Register function. The raw results can be further inspected in Appendix Section C. Evidently, the nature of the 7-Nodes Example which its Blockchain nodes need to share processing units of the same host machine could be the main factor affecting the performance. The more processing threads executed together at once mean the more processor resource of the host machine divided for each thread. So, the shared processing unit environment may be affecting the performance of each node resulting in a difference in performance as shown in Figure 5.

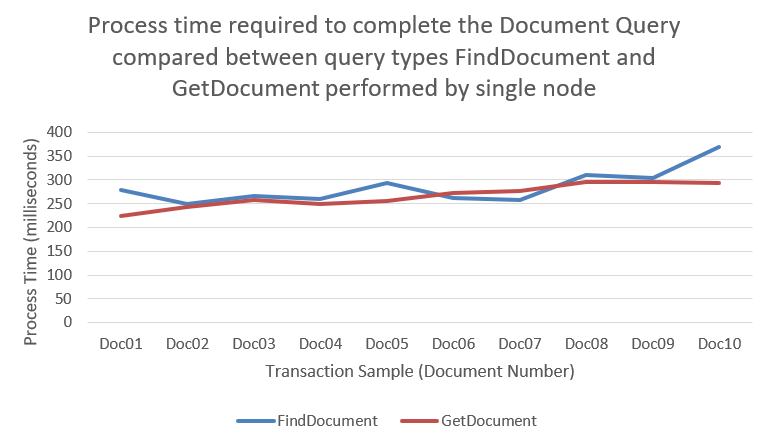


Figure 8 Processing time to complete Document Query  
compared between FindDocument and GetDocument   
performed by single node

Test if the number of search keywords would cause a change in performance of Document Query function. This setup is performed by trigger the XDS Document Consumer Actor to start the Document Query function which prompts for input from the user. The user then selected for “FindDocument” query type and inputs metadata attributes values of each transaction sample (Doc01 – Doc10) into the program. The program then proceeds to query for the matching metadata set registered in the Blockchain ledger from the XDS Document Registry Actor in the 1st Node (only one active node received the query). The processing time since the moment the XDS Document Consumer Actor sent the ITI-18 transaction to the XDS Document Registry Actor until the XDS Document Consumer received the search result from the XDS Document Registry will be measured for the experiment result. The "minimum" search keywords will include only two attribute values while the "maximum" search keywords will include fifteen attribute values. The setup for “Single Node” and “All Nodes” is like the first case. The Document Query with the "FindDocument" query type performed by a single node using minimum keywords took an average process time of 285.25 milliseconds to complete the function. FindDocument with maximum keywords took an average process time of 287.74 milliseconds to complete the function. Performed by all active nodes, minimum keywords took an average process time of 754.12 milliseconds to complete the function while maximum keywords took an average process time of 767.97 milliseconds to complete the function. Figure 6 and Figure 7 show no significant difference between the Document Query with maximum search keywords compared to the Document Query with minimum search keywords. That means the amount of search keywords input is not affecting the performance of the Document Query function. A full version of the experiment result can be further inspected in the Appendix section. The Document Query function program only used provided search keywords to search for the matching transaction registered in the Blockchain ledger. A greater number of search keywords may take a little more process to finish the search, but the change will not add more significant processing time to the whole process which was expected not to affect the performance of the system.

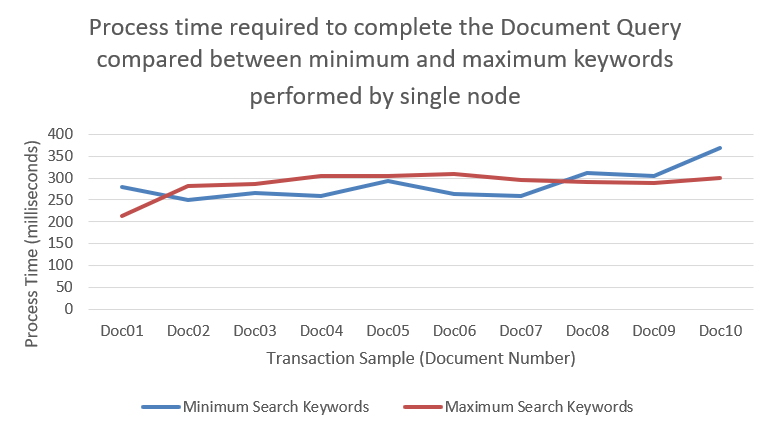


Figure 7 Processing time to complete Document Query   
compared between minimum keywords and maximum keywords performed by single node

Test if different query types would cause a change in performance of Document Query function. This setup is performed by trigger the XDS Document Consumer Actor to start the Document Query function which prompts for input from the user. The user then selected for query type and inputs metadata attributes values of each transaction sample (Doc01 – Doc10) into the program. The program then proceeds to query for the matching metadata set registered in the Blockchain ledger from the XDS Document Registry Actor in the 1st Node (only one active node received the query). The processing time since the moment the XDS Document Consumer Actor sent the ITI-18 transaction to the XDS Document Registry Actor until the XDS Document Consumer received the search result from the XDS Document Registry will be measured for the experiment result. The test begins with the "FindDocument" query type with minimum keywords before the "GetDocument" query type with required keyword (1 keyword). The result then compares the performance of each query type. The setup for “Single Node” and “All Nodes” situation is like Section 4.4.2.1. The Document Query with the "FindDocument" query type performed by a single node using minimum keywords took an average process time of 285.25 milliseconds to complete the function. The "GetDocument" took an average process time of 266.35 milliseconds to complete the function. Performed by all active nodes, the "FindDocument" took an average process time of 754.12 milliseconds to complete the function while the "GetDocument" took an average process time of 765.78 milliseconds to complete the function. Figure 8 and Figure 9 show no significant difference between the Document Query with the "FindDocument" query type compared to the Document Query with the "GetDocument" query type. That means the query type is not affecting the performance of the Document Query function. A full version of the experiment result can be further inspected in the Appendix section. The different query types only change the end content of the search result while remain operates on the same base algorithm, so the change will not add more significant processing time to the whole process which was expected not to affect the performance of the system.

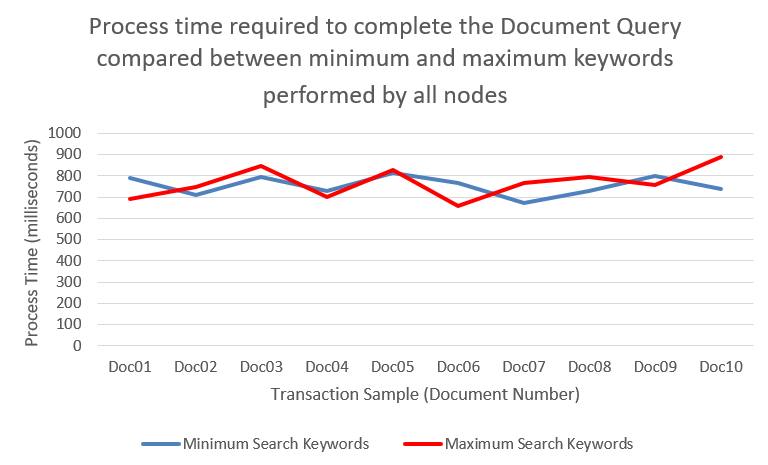


Figure 9 Processing time to complete Document Query   
compared between minimum keywords and maximum keywords   
performed by all nodes

Test if the number of transactions would cause a change in performance of Document Query function. This setup compares all graphs resulting from all cases above to show the overall difference between the case when there is only one single node (1st Node) perform the Document Query function and when all active nodes perform the Document Query function (inspect process time from 1st Node). Figure 10 shows a significant difference between the setup when the Document Query function is performed by a single node and the setup when the Document Query function is performed by all active nodes. When transactions are entering all nodes at the same time, the system took a significantly longer time to complete the Document Query function when compared to the processing time when there is only one transaction from a single node performing the function. That means the number of transactions is affecting the performance of the Document Query function. This could be the effect from the 7-Nodes Example which its Blockchain nodes need to share processing units of the same host machine. This may be the main factor affecting the performance when all nodes need to perform the Document Query function at once at the same time. So, the shared processing unit environment may be affecting the performance of each node resulting in a difference in performance as shown in Figure 10.

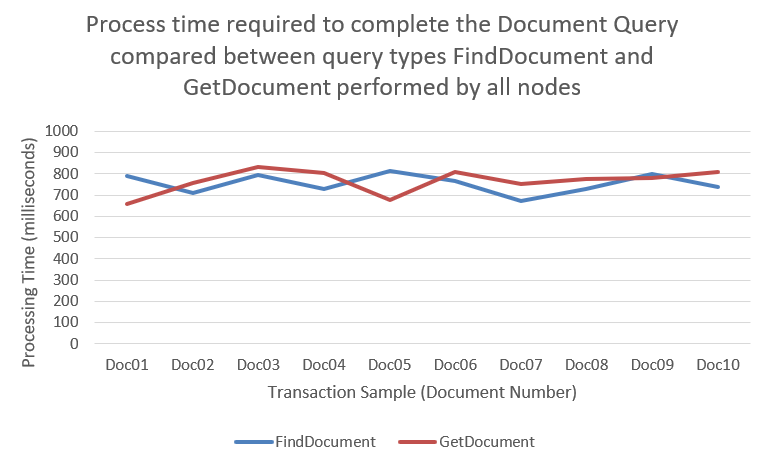


Figure 10 Processing time to complete Document Query compared between FindDocument and GetDocument performed by all nodes

# discussion

Amongst many metadata attributes, there are attributes that value can expose patients' confidential information into the Blockchain network. The attribute "sourcePatientInfo", for example, directly contains personal information for patients whose medical record the document is associated with. This attribute can contain multiple values such as patient name and address. So, in the actual adoption of the proposed concept, these metadata attributes values must be anonymized before entering the Blockchain ledger. This can be done by replacing the value with its hash counterpart when the attributes entering the Blockchain ledger via the Document Register function. At the same time, when the attributes were required for the search operation as search keywords input, the XDS Document Consumer will only need to hash the input value and use the hash to allow the XDS Document Registry actor to search for the matching hash value registered within the Document Registry Blockchain. This allows the concept to maintain confidentiality of patients' data while preserving the functionalities of the concept. During the study on the IHE profiles, we found that required information including transaction formats, data structures, and specific protocols are specifically designed for each IHE profile and published separately in different volumes of IHE documents. Some part of the document was deprecated or obsoleted in the present. There are a limited number of transaction samples available from the framework which makes it difficult for a beginner to develop a system that fully complies with the IHE specifications and relevant standards. It’s a time-consuming process. For Ethereum, the usage of SmartContract was limited by its gas requirement. Every task performed by SmartContract requires gas which represents the computational power of the network to perform. That means there will be more factors for the platform to be addressed before the actual adoption. We suggest that the adopted network must be able to provide enough amount of gas or disregard the component from the consideration by integrating it into the common agreement amongst the networks.

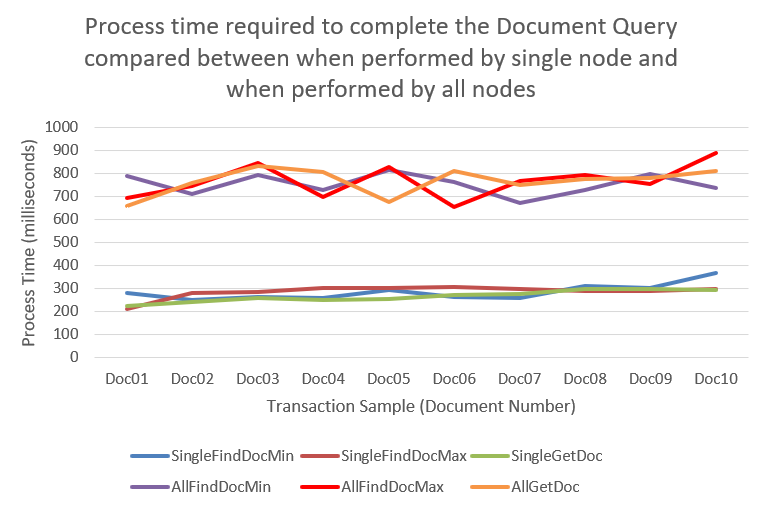


Figure 11 Processing time to complete Document Query compared between   
when performed by single node and when performed by all nodes

# Conclusion & Future Works

We achieved a system for the XDS Blockchain which acts as the medium for health document sharing amongst the XDS Affinity Domain Network. The prototype was designed to be compatible with normal XDS Actors while also act as a medium for a common XDS network to interface with the IBFT Blockchain ledger. This enhances the IHE XDS.b Profile with the Blockchain characteristics while appreciating the network to further share their health document to further benefit from the network for both operational interoperability and cyber-security. However, due to the ever-changing nature of the software platform, further adoption of the implementation will need to be updated as the platform released newer version of the software to avoid version conflict of the source code.

In the implementation of the proposed concept, we excluded an XDS On-Demand Document Repository Actor and XDS Patient Identity Feed actor to reduce the complexity of the concept demonstration. For future work, those XDS Actors should also be implemented to the XDS Blockchain concept too. The XDS On-Demand Document Repository would enhance the benefit of the XDS Blockchain as it provides On-Demand health document type which gave flexibility and a wider range of usability of shared health documents to healthcare operation.

At the same time, The Patient Identity Feed Actor would aid the member of the XDS Affinity Domain Blockchain network by establishing the medium identifier for all members to seamlessly share their health documents. The Patient Identity Feed Actor may even further integrate into the SmartContract and eliminate the need for centralized identity feed in the network. Eliminate the cost which would be spent on maintaining the Patient Identity Feed Actor for the network.

Furthermore, the SmartContract also has the potential to become the exchange medium for ITI-43 transactions where the XDS Document Consumer negotiates with XDS Document Repository for retrieving actual health documents, allow health documents exchanging activities in the network to be recorded in the Blockchain ledger which could be further used in the incident investigation during the cyber-incident. These would maximize the potential of Blockchain technology implemented on the Cross-Enterprise Document Sharing Profile.

Other than the Cross-Enterprise Document Sharing Profile, the IHE IT Infrastructure is providing much more profiles and various tools for use in achieving healthcare interoperability. There remain a lot more possibilities of using the framework to maximize the potential of Blockchain technology and the future technology to come. As Blockchain technology still has a long way of development and research path to go through, the concept proposed in this work also could be further developed into a more advanced version for actual adoption in the future.

The source code of the project can be downloaded at URL.

# References

[1] D. H. Interoperability, “Digital Healthcare Interoperability,” no. October, 2016.

[2] PolicyMedical, “Interoperability in Healthcare: To Have or Not to Have.” https://www.policymedical.com/interoperability-healthcare/ (accessed Sep. 22, 2018).

[3] Carestream Health, “Interoperability : Connecting the Healthcare Enterprise to Deliver Responsive Patient Care,” pp. 1–9, 2015, [Online]. Available: http://www.carestream.com/clinical-collaboration/sites/default/files/WhitePaper\_CCP\_Interoperability\_LTR\_201508\_en\_Web.pdf.

[4] Dr.David Hay, “Why is interoperability so important for healthcare organisations? | Orion Health.” https://orionhealth.com/global/knowledge-hub/blogs/why-is-interoperability-so-important-for-healthcare-organisations/ (accessed Apr. 27, 2019).

[5] Paige Goodhew, “Why Healthcare Interoperability Matters | Redox.” https://www.redoxengine.com/blog/why-healthcare-interoperability-matters/ (accessed Apr. 27, 2019).

[6] Oracle, “Interoperability : A Key to Meaningful Use,” *Solutions*, no. November, 2010, [Online]. Available: http://www.oracle.com/us/industries/healthcare/interoperability-wp-188782.pdf.

[7] HIMSS, “What is Interoperability?” https://www.himss.org/library/interoperability-standards/what-is-interoperability (accessed Apr. 27, 2019).

[8] Healthcare Information and Management Systems Society, “Definition of Interoperability,” *Himss*, p. 2013, 2013.

[9] W. E. Hammond and J. J. Cimino, *Standards in Medical Informatics*, no. December. 2001.

[10] S. Rosslyn, “DICOM PS 3.15 Security and System Management Profiles,” *DICOM Stand.*, 2011.

[11] DICOM Library, “About DICOM SOPs.” https://www.dicomlibrary.com/dicom/sop/ (accessed Oct. 22, 2018).

[12] The Royal College of Radiologists Board of the Faculty of Clinical Radiology, “DICOM and HL7 standards.”

[13] “FHIR v4.0.1.” https://www.hl7.org/fhir/ (accessed Oct. 13, 2020).

[14] IHE International Inc, “About IHE.” https://www.ihe.net/about\_ihe/ (accessed Sep. 11, 2018).

[15] IHE International Inc, “IHE IT Infrastructure ( ITI ) Technical Framework Volume 1 Integration Profiles,” *Int. J. Healthc. Technol. Manag.*, vol. 1, no. 8.0, pp. 1–177, 2008, doi: 10.1504/IJHTM.2008.017371.

[16] “First known ransomware attack in 1989 also targeted healthcare.” https://www.beckershospitalreview.com/healthcare-information-technology/first-known-ransomware-attack-in-1989-also-targeted-healthcare.html (accessed Oct. 29, 2021).

[17] Healthcare IT News, “The biggest healthcare breaches of 2017.” https://www.healthcareitnews.com/slideshow/biggest-healthcare-breaches-2017-so-far?page=1 (accessed Sep. 11, 2018).

[18] Healthcare IT News, “The biggest healthcare data breaches of 2018 (so far).” https://www.healthcareitnews.com/projects/biggest-healthcare-data-breaches-2018-so-far (accessed Apr. 27, 2019).

[19] HIPAA Journal, “Largest Healthcare Data Breaches of 2018.” https://www.hipaajournal.com/largest-healthcare-data-breaches-of-2018/ (accessed Apr. 27, 2019).

[20] “The 10 Biggest Healthcare Data Breaches of 2019, So Far.” https://healthitsecurity.com/news/the-10-biggest-healthcare-data-breaches-of-2019-so-far (accessed Mar. 04, 2021).

[21] “UPDATE: The 10 Biggest Healthcare Data Breaches of 2020.” https://healthitsecurity.com/news/the-10-biggest-healthcare-data-breaches-of-2020 (accessed Mar. 04, 2021).

[22] V. Puri, A. Kataria, and V. Sharma, “Artificial intelligence-powered decentralized framework for Internet of Things in Healthcare 4.0,” *Trans. Emerg. Telecommun. Technol.*, p. e4245, 2021, doi: 10.1002/ETT.4245.

[23] “Czech hospital hit by cyberattack while in the midst of a COVID-19 outbreak | ZDNet.” https://www.zdnet.com/article/czech-hospital-hit-by-cyber-attack-while-in-the-midst-of-a-covid-19-outbreak/ (accessed Mar. 04, 2021).

[24] D. Cosset, “The 4 characteristics of a blockchain - DEV Community.” https://dev.to/damcosset/the-4-characteristics-of-a-blockchain-2c55 (accessed Oct. 29, 2018).

[25] Deloitte, “Key Characteristics of the Blockchain,” Accessed: Oct. 29, 2018. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/in/Documents/industries/in-convergence-blockchain-key-characteristics-noexp.pdf.

[26] Data Flair, “6 Major Features Of Blockchain | Why Blockchain is Popular?” https://data-flair.training/blogs/features-of-blockchain/ (accessed Oct. 29, 2018).

[27] Techracer-Medium, “4 Key Features of Blockchain – Techracers – Medium.” https://medium.com/techracers/4-key-features-of-blockchain-5a4aff025d38 (accessed Oct. 29, 2018).

[28] K. Peterson, R. Deeduvanu, P. Kanjamala, and K. Boles, “A Blockchain-Based Approach to Health Information Exchange Networks,” *Mayo Clin.*, no. 1, p. 10, 2016, doi: 10.1016/j.procs.2015.08.363.

[29] A. Ekblaw, A. Azaria, J. D. Halamka, A. Lippman, I. Original, and T. Vieira, “A Case Study for Blockchain in Healthcare: " MedRec " prototype for electronic health records and medical research data,” *IEEE Technol. Soc. Mag.*, pp. 1–13, 2016, doi: 10.1109/OBD.ta b2016.11.

[30] S. Tanwar, K. Parekh, and R. Evans, “Blockchain-based electronic healthcare record system for healthcare 4.0 applications,” *J. Inf. Secur. Appl.*, vol. 50, p. 102407, Feb. 2020, doi: 10.1016/j.jisa.2019.102407.

[31] A. Ekblaw and A. Azaria, “MedRec: Medical Data Management on the Blockchain,” Accessed: Sep. 26, 2018. [Online]. Available: https://viral.media.mit.edu/pub/medrec.

[32] A. Medical Association, “What is Information Blocking? Part 1.”

[33] G. Zyskind, O. Nathan, and A. S. Pentland, “Decentralizing privacy: Using Blockchain to Protect Personal Data,” *Proc. - 2015 IEEE Secur. Priv. Work. SPW 2015*, pp. 180–184, 2015, doi: 10.1109/SPW.2015.27.

[34] D. Yaga, P. Mell, N. Roby, and K. Scarfone, “Blockchain Technology Overview (NISTIR-8202),” *NISTIR*, p. 59, 2018, doi: 10.6028/NIST.IR.8202.

[35] “Blockchain Consensus: A Simple Explanation Anyone Can Understand.” https://blockgeeks.com/guides/blockchain-consensus/ (accessed Oct. 13, 2020).

[36] D. Drescher, *Blockchainbasics*. Apress, Berkeley, CA.

[37] D. Mingxiao, M. Xiaofeng, Z. Zhe, W. Xiangwei, and C. Qijun, “A review on consensus algorithm of blockchain,” *2017 IEEE Int. Conf. Syst. Man, Cybern. SMC 2017*, vol. 2017-Janua, pp. 2567–2572, 2017, doi: 10.1109/SMC.2017.8123011.

[38] “Proof-of-Work, Explained.” https://cointelegraph.com/explained/proof-of-work-explained (accessed Oct. 13, 2020).

[39] S. De Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, “PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain,” *CEUR Workshop Proc.*, vol. 2058, pp. 1–11, 2018.

[40] “Home | ethereum.org.” https://ethereum.org/en/ (accessed Nov. 12, 2021).

[41] V. Buterin, “A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM.”

[42] henriquegaia, “ethereumbook/07smart-contracts-solidity.asciidoc at develop · ethereumbook/ethereumbook · GitHub.” https://github.com/ethereumbook/ethereumbook/blob/develop/07smart-contracts-solidity.asciidoc#what-is-a-smart-contract (accessed Aug. 23, 2020).

[43] “ethereum/go-ethereum: Official Go implementation of the Ethereum protocol.” https://github.com/ethereum/go-ethereum (accessed Jun. 05, 2021).

[44] “Installing Geth | Go Ethereum.” https://geth.ethereum.org/docs/install-and-build/installing-geth#install-on-ubuntu-via-ppas (accessed Jun. 10, 2021).

[45] “Command-line Options | Go Ethereum.” https://geth.ethereum.org/docs/interface/command-line-options (accessed Jun. 10, 2021).

[46] Quorum, “GoQuorum,” 2020. https://github.com/ConsenSys/quorum (accessed Jun. 03, 2021).

[47] “Install - GoQuorum.” https://docs.goquorum.consensys.net/en/stable/HowTo/GetStarted/Install/ (accessed Jun. 10, 2021).

[48] “Quorum 7-nodes Example.” https://github.com/ConsenSys/quorum-examples/tree/master/examples/7nodes (accessed Jun. 03, 2021).

[49] “ConsenSys/tessera: Tessera - Enterprise Implementation of Quorum’s transaction manager.” https://github.com/ConsenSys/tessera (accessed Jun. 05, 2021).

[50] “ConsenSys/constellation: Peer-to-peer encrypted message exchange.” https://github.com/ConsenSys/constellation (accessed Jun. 05, 2021).

[51] “quorum-examples/README.md at master · ConsenSys/quorum-examples.” https://github.com/ConsenSys/quorum-examples/blob/master/README.md (accessed Jun. 10, 2021).

[52] “Remix - Ethereum IDE.” https://remix.ethereum.org/ (accessed Mar. 08, 2021).

[53] “Node.js.” https://nodejs.org/en/ (accessed Jun. 10, 2021).

[54] “npm.” https://www.npmjs.com/ (accessed Jun. 10, 2021).

[55] “web3 - npm.” https://www.npmjs.com/package/web3 (accessed Jun. 10, 2021).

[56] “xml2js - npm.” https://www.npmjs.com/package/xml2js (accessed Jun. 10, 2021).

[57] “fs - npm.” https://www.npmjs.com/package/fs (accessed Jun. 10, 2021).

[58] “net - npm.” https://www.npmjs.com/package/net (accessed Jun. 10, 2021).

[59] “util - npm.” https://www.npmjs.com/package/util (accessed Jun. 10, 2021).

[60] “moment - npm.” https://www.npmjs.com/package/moment (accessed Jun. 10, 2021).

[61] “cryptr - npm.” https://www.npmjs.com/package/cryptr (accessed Jun. 10, 2021).

[62] “web3.eth.Contract — web3.js 1.0.0 documentation.” https://web3js.readthedocs.io/en/v1.3.4/web3-eth-contract.html (accessed Mar. 08, 2021).