Proceedings of the 2019 International Conference on Parallel and Distributed Processing Techniques & Applications

ISBN: 1-60132-508-8 | Copyright © 2019 CSREA Press | United States of America

A Blockchain-based IoT Platform Integrated with Cloud Services

A. Debrath Banerjee[1]**, B. Hai Jiang**[2]
debrath.banerjee@smail.astate.edu, hjiang@astate.edu
Computer Science Department, Arkansas State University, Jonesboro, Arkansas, United States

Abstract – *Nowadays, Blockchain is not an enigmatic, technical term for many people rather a technology that holds promise as a way not only to record financial transactions but also to decentralize infrastructure and build a trust layer for business logic. Currently it is considered to be a game-changer that could propel us into next industrial revolution. Motivated by the recent explosion of interest around blockchains, blockchain development is undoubtedly accelerating and empowering many financial sectors. However non-financial application areas such as supply chain, health care or eCommerce industry have incredibly grown complex due to the involvement of IoT. So, although they intend to reshape the model, still lots of uncertainty exist in terms of privacy, integrity, scalability and effectiveness. In convergence of blockchain and IoT, there is no definite design model. With all of these considerations in mind, we have developed a blockchain based IoT platform to replace the traditional monolithic sales order management process. The design prototype is associated with the contract creation to enable automated fulfilment of orders from warehouse. This hybrid design represents the convergence of smart contract enabled private Ethereum Blockchain with IoT & also ensure IoT security and controls with combination of AWS-IoT cloud services.*

Keywords: *Private Blockchain, Internet of Things, Smart Contract, AWS-IoT, Convergence, Sales Order Management*

1 Introduction

In recent days, the blockchain technology is undoubtedly an ingenious breakthrough of secure computing without any dependency of centralized authority. From data management perspective to security, it provides an efficient mechanism through intelligent and decentralized utilization of cryptography with crowd computing. As a new and revolutionary technology, it faces lots of hurdles and touted as new internet after it was documented by Satoshi Nakamoto [1] in 2008. Its introduced as a platform for virtual cryptocurrency Bitcoin which uses blockchain to record all of its transactions. However, this ingenious technology has the potentiality to change the way of internet functions. This technology is not only to facilitate the exchange of virtual currency, but also to utilize it in the financial or non-financial field. After the induction of the Ethereum platform, most sectors including finance, supply chain, logistics, health care have been rapidly accelerated towards adopting this evolution. Furthermore, we are also witnessing the proliferation of Internet of Things (IoT) technologies which mostly make a big impact on supply chain or order management industry. However existing IoT technologies are poorly designed and implemented with diverse protocols and technologies based on centralized server/client paradigm. Convergence of the Blockchain and IoT has shown a new direction to transform the traditional model [2][3].

In reality, blockchain with IoT architecture is very complex and has certain limitations, especially regarding scalability. The design approach is not very transparent to the industry that sometimes make them unresponsive. Although public blockchain undoubtedly have greater consensus strength and visibility, it is not a secure approach to expose the organizational data publicly. Most of the real sector initially intended to apply blockchain, but they pushed back later due to the privacy constraint and lack design approach for private blockchain. Nowadays supply chain industry also have grown incredibly complex due to the involvement of various actors. However, it is unmanageable to track unethical practices [4][5]. Several industries are still accustomed to centralized digital contracts. Furthermore, involvement of IoT also comes with lot of constraints in terms of security, integrity and scalability. Most of the blockchain based IoT solutions face resource constraint issues.

So, based on the literature study and after analysis of various Blockchain perspective, we have taken an use case on sales order management process which is associated with contract creation to enable automated fulfillment of orders from warehouse with the convergence of Smart contract, Blockchain, IoT and Cloud services. Our implementation is based on micro-service architecture to restructure traditional monolithic model. Rather than using public blockchain infrastructure, cloud independent private blockchain framework is developed with smart contract. In our design, IoT is not involved as a direct blockchain node and also is not holding subset of blockchain for transaction validation. IoT is taking advantage of blockchain rather than involving as a peer to avoid resource constraint issues. Device integrity is managed with the exchange of digital certificates instead of

ISBN: 1-60132-508-8 | Copyright © 2019 CSREA Press | United States of America

using agents. AWS-IoT platform is used not only to ensure device integrity also to facilitate device controls. Furthermore, we have looked upon to protect the node servers from DDoS attack with the reverse proxy mechanism. This research work is conducted with some goals. Firstly, it is providing a clear exposure for those sectors who are planning to reshape their traditional model. Secondly, it will provide a gateway for non-security experts to gain better understanding of private blockchain and smart contract. In addition, it will help specialists and blockchain enthusiast to explore cutting edge technologies related to blockchain, IoT and Cloud services.

We organize the rest of the paper as follows. Section 2 describes about the related works and challenges. Section 3 talks about the overview of key design components. Section 4 presents the overview of our approached design and its functional aspects. Section 5 introduces the implementation workflows based on our design. Section 6 describes about the experimental result. Section 7 concludes this paper.

2 Related Works

In this section, some blockchain based IoT projects are introduced. Most of them are recently and recognized by the academia and industry. IBM has unveiled its proof of concept for ADEPT, a system developed in partnership with Samsung. It uses the underlying design of Bitcoin to build decentralized Internet of Things. A paper [6] published by IBM and Samsung describes by how this will enable autonomous smart devices to directly communicate and verify the validity of transactions without the need for a centralized authority.

There are many use cases for blockchain based communications. A paper [7] published by IBM and Samsung describes by how blockchain can enable a washing machine to become "semi-autonomous device capable of managing its own consumables supply, performing self-service and maintenance, and even negotiating with other peer devices both in home and outside to optimize its environment". In spite of all its advantages, the blockchain model suffers its flaws and shortcomings. Bitcoin itself is dealing with scalability issues pertaining to blockchain, which are actually casting a shadow over the future of cryptocurrency. Furthermore, most devices of devices in IoT networks have very different computing capabilities, and not all of them can run the powerful cryptographic algorithms. Storage is another big constraint in this model.

Current centralized, cloud based IoT solutions are not scalable and incapable to meet security challenges. Several ongoing projects adopt blockchain enabled IoT solution and most of them utilize cloud services. IBM recently made contributions in developing blockchain platform for an IoT applications. It is based on the implementation of Hyperledger Fabric [8]. However, this model is also based on IBM Cloud. Though the concept is all about the decentralization but still it is not fully trustless and independent of central authority.

Our purpose is to approach a model based on private framework which will focus on scalability, integrity and privacy in order to address these common issues.

3 Overview of Key Design Components

In this section, key elements are introduced as the core ingredients to design this decentralized prototype. For blockchain enabled architecture, the selection of platform for decentralization is important as there are many platforms available for decentralization like Ethereum, Hyperledger, Lisk etc. Based on our system requirement, we have chosen Ethereum which is an open source, public blockchain-based distributed computing and operating system with smart contract functionality.

3.1 Ethereum Virtual Machine

The concept of Ethereum was introduced by Vitalik Buterin in November 2013. To turn a machine into a node in the Ethereum network, various Ethereum clients have been developed and the most popular ones are go-Ethereum (Geth) and parity. For Ethereum based development, Ganache is also very efficient solution to provide a virtual blockchain development environment, but it uses the same principle as any real node. It has the capability to sets up Ethereum addresses, complete with private keys and preload them with 100 simulated Ether each. Also, it can interact with smart contract in the same way as a real node does. Now in order to interpret the smart contract code, Ethereum Virtual Machine (EVM) is a key component to create ethereum smart contract byte code execution environment and act as a simple stack-based execution machine [9][10]. The stack size is limited to 1024 elements and is based on the LIFO (Last in First Out) queue. Since all the nodes execute all transactions that point to smart contract using EVM, every node does the same calculations and stores the same values. Every node executes the transactions and stores the final state. Furthermore, every transaction requires some computation and storage in the network. Therefore, there needs to be a transaction cost to maintain the network from being flooded with spam transactions.

3.2 Smart Contract

The term "smart contract' was first coined by Nick Szabo on 1997. The purpose of this protocol is to digital facilitate, verify and enforce the business logic by self-execution for security which is commonly superior property than traditional agreement. We can deploy the smart contract inside of Ethereum blockchain network which is executed by EVM. That makes the contract become immutable and decentralized as the code executed in every node as a part of transaction. A contract resides on blockchain in the Ethereum-specific binary format called EVM bytecode which is an assembly language one made up of multiple opcodes. Each opcode performs a certain action on the Ethereum blockchain. Now, as a process of development to deployment of the smart contract, the very first step of the smart contract is to write and compile the code. Truffle framework, Remix or from the geth JavaScript console helps build the contract and compile it through Solidity compiler (solc) [9]. Once compilation done, Application Binary Interface (ABI) and the *bytecode* is

Proceedings of the 2019 International Conference on Parallel and Distributed Processing Techniques & Applications

ISBN: 1-60132-508-8 | Copyright © 2019 CSREA Press | United States of America

generated. The *bytecode* is what will actually go onto the blockchain to make the smart contract work. But there is no way from byte code to identify the contract functions. Calling such a contract would be next to impossible. This is where interface comes into play which is JSON representation of a contract, given by an array of function and/or event descriptions. It allows us to contextualize the contract and call its functions. Now the aforementioned bytecode is not yet accessible until it is deployed on an Ethereum network. The deployment of a contract bytecode is done through a transaction. We use Web3.js framework in order to deploy the contract. Web3 internally creates raw transaction and push that into node. After that once signed transaction is accepted by a miner and added to blockchain, the contract code will get executed at every node. A contract address is created accordingly for a deployed contract [11][12].

3.3 Internet of Things

In today's world, 'Internet of Things' is one of the popular areas where most of the industry starts to depend on it. By definition, IoT is a special purpose device which connects wirelessly to a network and engages with the transmission of data over that wireless connection to monitor or control a "thing". Essentially IoT devices contain sensors to acquire data and actuators to control or act on the data. There are few challenges to build blockchain based IoT platforms including device security, device management, scalability and of course control management. Mostly IoT systems are exposed to the outside world that imposes risk for vulnerabilities. Also, the control of a device is very critical if there is a network failure, it is hard to recover the last state of the device. In addition, IoT also have resource constraint issue if we would plan to include IoT devices as blockchain nodes. From the design perspective, these issues have to be properly addressed [13].

4 System Design

Our approached design intends not only to utilize the benefits of blockchain, but also look into various technical and functional aspects of this microservices prototype design. This hybrid model integrates private blockchain with smart contract for an immutable decentralized platform to secure the transactions data, support self-enforcement of business logic and maintain the privacy of the organizational data. In addition, AWS IoT platform has been incorporated to secure and manage IoT devices. The main design components are ethereum blockchain, smart contract, IoT, AWS cloud services and a front-end interface hosted on any webserver to initiate contract and other IoT transactions. Our approach is partially influenced by fog architecture where private blockchain will be hosted across different local servers which form middle layer of fog architecture for smart contract to reside as an immutable agreement. As an IoT device does not have the enough capability to hold a blockchain client , it will do the minimum computation and interact on main private networks by RPC. Parallelly IoT devices integrity and control will be managed by AWS-IoT platform. Sale order management system with sequential flow is shown in Figure 1 for the illustration of smart contract and transactions. Figure 2 displays the communication between IoT and blockchain.

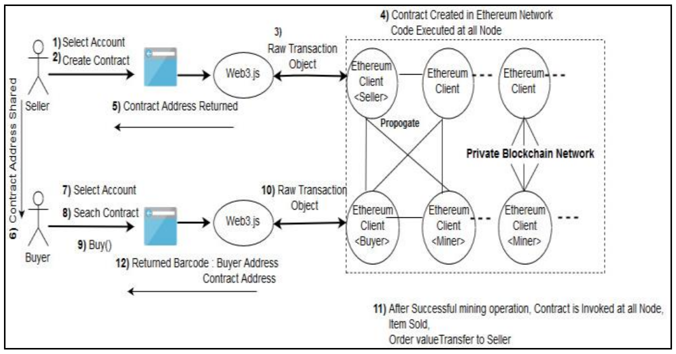

4) Contract Created in Ethereum Network
Code Executed at all Node
1) Select Account
2) Create Contract
3) Raw Transaction Object
Web3.js
Ethereum Client <Seller>
Ethereum Client
Ethereum Client
Seller
5) Contract Address Returned
Propogate
Private Blockchain Network
6) Contract Address Shared
7) Select Account
8) Seach Contract
9) Buy()
10) Raw Transaction Object
Ethereum Client <Buyer>
Ethereum Client <Miner>
Ethereum Client <Miner>
Buyer
12) Returned Barcode : Buyer Address Contract Address
11) After Successful mining operation, Contract is Invoked at all Node, Item Sold, Order valueTransfer to Seller

Figure 1 Design Overview of Contract Creation

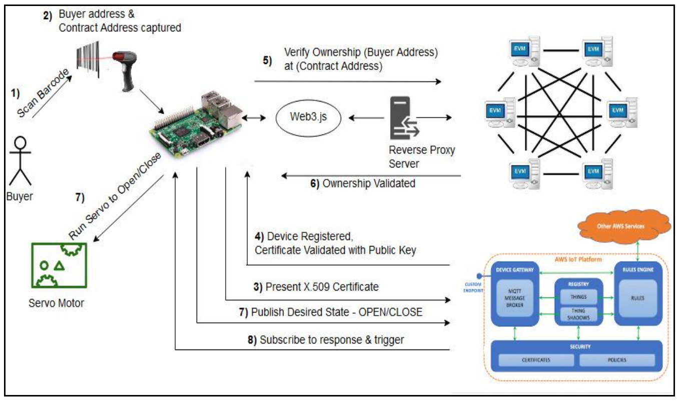

2) Buyer address & Contract Address captured
1) Scan Barcode
Buyer
5) Verify Ownership (Buyer Address) at (Contract Address)
Web3.js
Reverse Proxy Server
6) Ownership Validated
EVM
7) Run Servo to Open/Close
Servo Motor
4) Device Registered, Certificate Validated with Public Key
3) Present X.509 Certificate
7) Publish Desired State - OPEN/CLOSE
8) Subscribe to response & trigger
Other AWS Services
AWS IoT Platform
DEVICE GATEWAY
MQTT MESSAGE BROKER
REGISTRY
THINGS
THING SHADOWS
RULES ENGINE
RULES
SECURITY
CERTIFICATES
POLICIES

Figure 2 Communication Model between IoT and Blockchain

4.1 Creation of Contract

Sales order management module is solely depended on three actors where initial two primary actors are Seller/Order creator and Buyer/customer and third one is IoT client/devices. Our design is concerned with private blockchain framework. Each of the primary actors has an ethereum client installed as a private node which is further connected with other private ethereum nodes. It is not always mandatory that buyers or sellers to have ethereum clients. They should have public key and private key in order to participate in transactions. For real time applications Go-Ethereum client will be used to build private blockchain framework. Private nodes are not supposed to be connected with publicly available real production ethereum main-net. So, blockchain framework has to be established within private networks. New genesis blocks are created to communicate over inter process communication. Sellers or buyers participate in transactions or contract creation through separate browser applications. Sellers will first create a sale contract with the details like Order name, Serial number, Price (in Ether) etc. Then Web3.js JavaScript API handles the request and interact with ethereum network over JSON /RPC.

This order creation process actually initiates the creation of new smart contract with these attributes.

4.2 Transaction Process

Once contract is created, raw transaction will be sent over to the Seller’s local Ethereum node. The transaction carries the information about contract’s byte code, account details and related parameters. The Ethereum network needs to know that seller actually own that account to make sure someone else does not execute this transaction on the seller’s behalf. The way to prove this to the network is by signing the transaction using the private key corresponding to that account.

Once transaction is validated locally using a public key, the signed transaction is broadcast by Seller’s Ethereum node to its peers who in turn broadcast it to their peers and so on. Once transaction is broadcast, local node also outputs transaction id which is the hash value of the signed transaction object. With this transaction hash we can examine the transaction detail using *eth.getTransaction()* and *eth.getTransactionReceipt().* The contract address belongs to transaction receipt which is returned to seller in browser application. However, not all nodes will accept the transaction. Some of these nodes might have a setting to only accept transactions with certain minimal gas price. If a gas price is lower than that limit, that node will just be ignored in the transaction.

4.3 Mining of the Contract

In Ethereum network miners are the ones who include the transactions in the blocks. Miners maintain a transaction pool where our transaction are added for evaluations. The miners store all the transactions in the pool and sort them in certain order for example by gas price. The higher the gas price, the more likely the transaction is included in the next block. But in case of private blockchain, miners from the organization should select the transactions irrespective of gas price. Miners eventually pick transaction to include in the block along with other transactions. Once the miners select the transactions to include in a block, the transactions are validated and included in a pending block. After Proof of Work is done, miner broadcasts the new blockchain with the updated block. Eventually all local nodes will receive this new block and synchronize its local copy in the blockchain. Upon receiving the new block, each local Ethereum client executes all the transaction in the block. So eventually contract code gets executed inside EVM across the Ethereum network. So right now, a new contract is successfully deployed across the network.

4.4 Invocation of Deployed Smart Contract

With the shared contract addresses, the buyer can access to deployed contract objects. Once product validation is done, buyer agrees upon the contract to purchase that order item. Transaction object will be created to carry the code and invoke the buy() function within the deployed smart contract. Once a raw transaction is created, it will be signed by the buyer’s private key in the same manner as aforementioned process. The transaction will be included into the blockchain once the miner validates it. Eventually, contract code will be executed at all local blockchain nodes. After byte code gets executed, item status will be changed, and appropriate order value (Ether) will be transferred from buyer account to seller account. The balance will only be transferred if contract is successfully executed. After that everyone’s wallet balance is updated. Since receipt is one of the important concepts in Invoice to Cash cycle, an autogenerated barcode is provided as a transaction receipt to buyer. It includes contract address and buyer address if smart contract execution is done successfully. This barcode will be used to release an item from ware house operated by IoT. But before interaction with Ethereum network, IoT integrity should be validated.

4.5 IoT Registration with AWS-IoT Core

Before IoT interacts with Ethereum blockchain, IoT integrity should be validated. So, we have designed this on the top of AWS-IoT core platform which uses MQTT or http protocol to communicate between IoT and cloud services. The first step of communication between a device and AWS IoT is protected through the use of X.509 certificates. Certificates are embedded inside IoT devices. When a device communicates with AWS IoT, it presents the certificate signed with private key to AWS IoT as a credential. AWS IoT validate that certificate with the help of associated public key. It is advisable to use unique certificate for each of the multiple IoT devices. If certificate gets compromised, we could revoke the policy as attached with that particular certificate. Once IoT integrity is validated, it can start communication with blockchain [15].

4.6 Request Through Reverse Proxy Server

IoT initiates a connection by remote procedure call with Ethereum network. Since mostly IoT is exposed to the outside world. It is hard to send a request to the private blockchain node unless that node is running at privileged port with public IP address. Also, if we run our blockchain in private IP at different unprivileged port, we have to handle NAT and make a port to forward requests. This is also not secure approach. Mostly NAT process is operated in routers or firewalls which are the control of central network administration team in an organization. Thus, it is not easily accessible. Therefore, the idea of reverse proxy server [14] is adopted here. The RPC over http request comes to reverse proxy server which is normally running at privileged port 80. The proxy will pass the request to the private Ethereum end point after basic authentication. Also, if there is multiple IoT devices involved, reverse proxy server could handle the load balancing of the multiple requests.

4.7 Verification of Ownership

Based on the contract address, buyer address and JSON object of the contract, IoT send a request to the Ethereum network to invoke the contract function of the deployed contract. A ‘call’ is the local invocation of contract function which does not broadcast or publish anything on blockchain. It is a read-only operation and will not consume any Ether. It simulates what would happen in a transaction but discards all the state changes when it is done. It is synchronous, and a value is returned immediately from the contract function. If valid, it will return true. Otherwise an error message is returned. Then, the buyer gets messages as its ownership is validated. The response will be fast because the validation is done inside Ethereum network, not managed by IoT, assuming with steady network connection.

4.8 Receive Command from AWS-IoT

Once ownership is validated, a command is received from AWS-IoT service to release the item. This process is a combination of a Publish/Subscribe model, and shadow service which is used to control the state of the things. Overall message communication will be done through MQTT protocol[7]. After the synchronization with AWS, IoT devices will send publish requests to change their new states to a desired open state and trigger the actuator. Through message broker , an AWS - IoT client subscribes to that particular event request and changes the state. If there is a change between the current state and previous state stored in shadow, AWS-IoT will publish a response associated with delta event which will be again subscribed by the IoT device. Once response is received, the actuator will be triggered. Then the IoT device again publishes its latest state as ‘Reported’ state to the AWS-IoT. If there is no change between the shadow states, AWS will not send any command.

5 Implementation

5.1 Private Blockchain Configuration

Firstly, we install ethereum client to the individual system hosted in local server. Then “geth” command is executed to start and synchronize the client. In a private blockchain, every node has to fulfill some requirements where a distinct data directory has to be created to store database and the wallet. Furthermore, every node has to initialize the same genesis file and pair of private and public key to make a transaction. A folder called *keystore/* will be created for the account file which basically holds the information about the account address.

Now as a part of p2p algorithm, we have to assume that there are certain nodes which will be always available. Those are called as bootstrap nodes in Ethereum. Boot nodes maintain a list of all nodes that connected to them in a period of predefined temporal value. The purpose of the boot node is to help other nodes discover each other. So, when a peer connects to the Ethereum network, it first connects with the boot nodes which shares the list of already connected peers. The newly connected ones then synchronize with other peers After initialization this creates a value called enode which uniquely identifies the boot node address. Once boot node address created, we can start the node as following way.

```
$bootnode –nodekey boot.key -verbosity 9 -addr :port no
```

Now we should start our seller and buyer nodes as presented with below command. There are few parameters associated with the command like port - the port through which nodes in a network communicate with each other and spread new transactions and blocks, Normally Ethereum uses default Port 30303 (TCP and UDP) as listener port and discovery ports respectively. Other parameters include rpc – enable HTTP-RPC server, rpcaddr – HTTP-RPC server listening interface, rpcport - HTTP-RPC server listening port, rpcapi - list APIs which can be accessed using RPC port, mine - makes the node a miner node, which mine ether and transactions, networkid - this is a unique network identifier that distinguishes one network from other networks and is used to join all the nodes of the same network.

```
$ geth --datadir node/ --syncmode 'full' –port num --rpc --
rpcaddr 'localhost' --rpcport --rpcapi
'personal,db,eth,net,web3,txpool,miner' --bootnodes
'enode://enode address@localhost:port_num' --networkid
--gasprice '1' -unlock 'seller account address' --password
password_file_location –mine
```

Now we have our own private blockchain with fully synchronized nodes.

5.2 Application Interface

Now as a part of order management module, the buyer or seller will interact with the blockchain through user friendly GUI. Application is involved with the creation of smart contract and deals with the recent transactions. Signing of the transaction and generation of public and private key pair could be handled inside application before sending the transaction object into the ethereum network. However, this application is designed as an interface to communicate with the ethereum network. Language specific Web3.js JavaScript API is used to pass the contract’s parameter into the ethereum network underlying JSON RPC. Now in order to connect through Web3, Provider is necessary to decide how web3 talks to the Blockchain. Providers take a JSON RPC request and submit it to HTTP or IPC socket based JSON- RPC server. Once provider is selected, web3 instance is ready to be initialized. Once web3 object is created, we can use the methods under web3.eth namespace to interact with RPC method which talks with EVM. Normally the Web3 object will look for Ethereum node in a default http://localhost:8545 address. In order to create new contract, compiled byte code along with JSON interface object is added into raw transaction and interfaced to ethereum network through web3 object. Normally web3.eth.Contract object is used to return contract.

5.3 Development of Smart Contract

Contracts are developed in Solidity language. Compilation, testing and deployment are done using Truffle framework. Once a contract is compiled, it generates JSON file to hold the information about contract unique byte code and Application

ISBN: 1-60132-508-8 | Copyright © 2019 CSREA Press | United States of America

Binary Interface object. Business logic of the contract is presented with the following Table 1.

Table 1 : Smart Contract Function Definitions

Function	Description	Param	Return to User
constructor	Constructor to create new contract	make, model, year, price, sold, owner	Contract's address
buy: Payable	Allows to transfer ether from contract address to seller upon successfully sold of item	buyer address, default gas value, price	Buyer get barcode contains contract's address and buyer's address
isSold: View	To check item sold or not	N/A	true/false
getItem: View	Allow user to retrieve item details	N/A	make, model, year, price, sold, owner, buyer
verifyOwnership: View	Allow user to verify the item status and ownership, used by IoT client	N/A	true/false

5.4 IoT Registration with AWS-IoT

AWS-IoT Core enables secure bi-directional communication between IoT and AWS cloud. Before IoT interact with blockchain, device integrity is ensured with AWS security mechanism. Communication between is protected through the use of X.509 certificates. It is associated with the registration of the things as shown Figure 3.

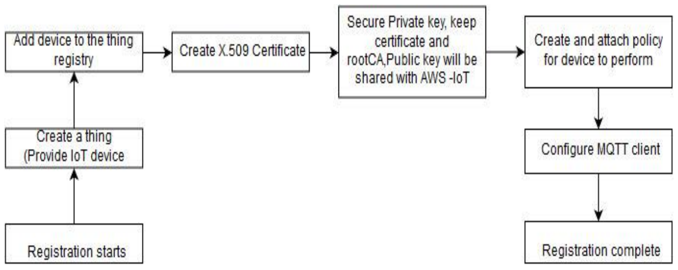

Registration starts
Create a thing (Provide IoT device
Add device to the thing registry
Create X.509 Certificate
Secure Private key, keep certificate and rootCA,Public key will be shared with AWS -IoT
Create and attach policy for device to perform
Configure MQTT client
Registration complete

Figure 3 IoT Registration

5.5 Verification of Ownership

5.5.1 Identity Validation

Communication between IoT and remote ethereum node will be established by submitting the request to HTTP based JSON RPC server. IoT application interface uses Web3.js to interact with remote blockchain nodes over RPC. The application is powered by Nodejs which creates a runtime environment to support this JavaScript application. Based on the contract address, deployed contract instance object is created which is used to invoke the verifyOwnership() function and validate the buyer's identity with the transactions stored in blockchain.

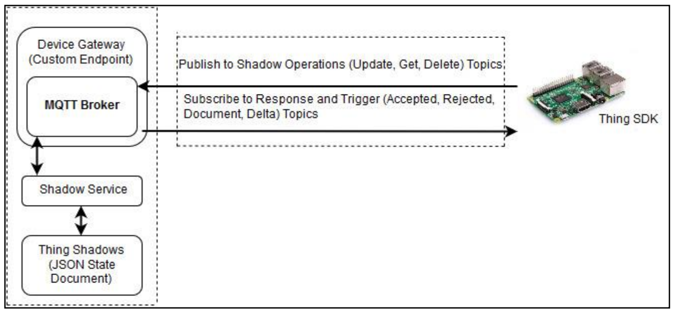
5.5.2 Request Transmission Process

Now the request transmission between IoT and private blockchain node is quite difficult as IoT belongs to different networks. It is not safe to expose Ethereum JSON-RPC API to public internet. Access protection to HTTP API is also important. So, reverse proxy server is introduced to sits between internal applications and external clients, forwarding client requests to the appropriate blockchain node. It acts as a line of defense for backend servers, protecting node servers from attacks such as DDoS. Also, reverse proxy can perform load balancing to distribute client requests evenly across backend servers. We have chosen Nginx [14] open source web server and **proxy_pass** directive are used to communicate with upstream geth that runs in localhost:8545.

5.6 AWS Shadow Service

In AWS, device shadow is a JSON document that stores and retrieves state information [15]. In Figure 4, we can see that devices report their states by publishing messages, in JSON format, on MQTT topics. These messages are sent to the MQTT message broker, which is responsible for sending all messages published on an MQTT topic to all clients subscribed to that topic. Now if AWS shadow service notices any change between the previous and the desired state stored in shadow documents, it publishes a response called Delta. Then IoT device subscribe to that topic and act accordingly. After that device again reports their new state to AWS shadow.

Figure 4 AWS Shadow Service


Device Gateway (Custom Endpoint)
MQTT Broker
Shadow Service
Thing Shadows (JSON State Document)
Publish to Shadow Operations (Update, Get, Delete) Topics
Subscribe to Response and Trigger (Accepted, Rejected, Document, Delta) Topics
Thing SDK

ISBN: 1-60132-508-8 | Copyright © 2019 CSREA Press | United States of America

6 Experimental Results

In this section, we show some transaction results based on the data set used to examine the functional behavior of the system. Input column of the Table 2 represents the parameters used to create the contract by seller. Once transaction is validated and added to blockchain, hash value 0x0ea2db81b9a4 for block 1 is generated which points genesis block. Additionally, contract address is populated. Buyer uses contract address to verify the transaction as shown in Table 3. After successful transaction, hash value 0x38af693b7b83 for block 2 is generated which points to hash value for block 1. Table 4 represents the data set used to examine the functionality of IoT with blockchain. We have tested with valid and invalid data for buyer's public key and received an expected result.

Table 2 : Transaction data for Seller/Contract owner

Input	Block hash	Contract address	Remarks
Public key- 0xb4168EA9a4 Model-alpha Year-2019 Price-50ETH Number-4AA6	Block 0 0xe6f425940a67 Block 1 0x0ea2db81b9a4	0x3dD9 6E8772	Contract Created

Table 3: Transaction data for Customer/buyer

Input	Block hash	Remarks
Public key- 0xD5044AA5aE Contract address 0x3dD9 6E8772	Block 1 0x0ea2db81b9a4 Block 2 0x38af693b7b83	Item sold, Barcode generated

Table 4: Transaction data for IoT

IoT	Barcode input	Result
Valid data set	Buyer addr 0xD5044AA5aE Contract addr 0x3dD96E8772	Ownership verified
Invalid data set	Buyer addr 0xe5048AA5sE Contract addr 0x3dD96E8772	Not verified

7 Conclusions

Convergence of blockchain and IoT definitely benefit the industry to meet the security demands over traditional security mechanism. Moreover, smart contract built upon blockchains offers the opportunity to build reliable decentralized IoT applications. However, it's very hard to predict potential application areas when it comes to implementation. Some related work has been done. However, they are either cloud based or faces scalability issues. Although some sectors want to transform into blockchain based solution, there is no simplified design approach to make them responsive. We did our work on a part of sales order management system to provide hybrid approach with convergence of private blockchain, IoT and AWS cloud services. At the end, we can conclude that integrating blockchain with IoT can bring many benefits over traditional system. But at the same time, it introduces new challenges that should be addressed. There still need more research for investigation. Our work is an initiative in this journey to find a scalable solution.

8 References

[1] Satoshi Nakamoto. 2008. "Bitcoin: A peer-to-peer electronic cash system", www.Bitcoin.Org, (2008)

[2] Arijit Chakrabarti, Ashesh Kumar Chaudhuri, "Blockchain and its Scope in Retail", IRJET, Volume: 04 Issue: 07, July 2017

[3] Suruchi Mann, Vidyasagar Potdar, Raj Sekhar Gajavilli, Anulipt Chandan ,"Blockchain Technology for Supply Chain Traceability, Transparency and Data Provenance", International Conference on Blockchain Technology and Application, 2018

[4] Rui Zhang, Rui Xue, Ling Liu "Security and Privacy on Blockchain", www.arXiv.org , Cornell University, 2019

[5] Johan Alvebrink, Maria Jansson, "Thesis: Investigation of Blockchain Applicability to Internet of Things within Supply Chains", Uppsala Universitet, June 2018

[6] "IBM ADEPT Practitioner Perspective Pre Publication Draft", formally announced at CES in Las Vegas, 2015

[7] "Device democracy, Saving the Future of the Internet of Things", IBM Institute for Business Value, 2014

[8] Amitranjan Gantait, Joy Patra, Ayan Mukherjee, "Implementing Blockchain for Cognitive IoT Applications", IBM Developer Works, IBM Corporation, 2017

[9] Imran Bashir, "Mastering Blockchain", Packt Publishing Ltd, pages 210-353, March 2017

[10] Vitalik Buterin, "Ethereum's White Paper: A Next-Generation Smart Contract and Decentralized Application Platform", 2013

[11] Amit Taherkordi, Peter Herrmann, "Pervasive Smart Contracts for Blockchains in IoT Systems, International Conference on Blockchain Technology and Application, 2018

[12] Konstantinos Christidis, Michael Devetsikiotis "Blockchains and Smart Contracts for the Internet of Things", IEEE Access, 2016

[13] Ahmed Banafa, "IoT and Blockchain Convergence: Benefits and Challenges", IEEE Internet of Things, January 2017

[14] Art Stricek, " A Reverse Proxy Is A Proxy by Any Other Name, published in SANS Technology Institute, 2019

ISBN: 1-60132-508-8 | Copyright © 2019 CSREA Press | United States of America

[15] "Designing MQTT Topics for AWS IoT Core", published in Amazon Web Services, Inc, 2018