Firearms tracking using Hyperledger Fabric – A Permissioned Blockchain Platform

Ashwath Venkataraman
CISE Department
Blockchain Optimization and Applications CIS6930
UFID: 5198-9461

Desikan Sundararajan
CISE Department
Blockchain Optimization and Applications CIS6930
UFID: 5615-9991

Vejay Mitun Venkatachalam Jayagopal
CISE Department
Blockchain Optimization and Applications CIS6930
UFID: 3106-5997

Mohammed Haroon Rasheed Kalilur Rahman
CISE Department
Blockchain Optimization and Applications CIS6930
UFID: 6751-2967

***Abstract*—Blockchain technology can be utilized to improvise gun control without changing existing laws. Firearm related mortality is at epidemic levels in the US and not only has a significant impact upon public health, it also creates a large financial burden. Gun ownership is a major factor affecting the number of suicides. Through better gun tracking and improved screening of high risk individuals, this technological advance in distributed ledger technology will advance background checks on individuals and fasten tracing of guns when crimes occur. This paper proposes a Hyperledger Fabric based solution to this problem and explains how this blockchain-based approach will help address the current issues. For this purpose, a prototype web application has been implemented with several key features. The role of each of the actors, the network model designed in Hyperledger Fabric and the method of smart contract formulation to address this use-case has been discussed in great detail, along with specifications regarding security aspects.**

I. Introduction

Blockchain as we all know is a highly disruptive technology, mainly due to its inherent capability of providing data immutability and decentralization of data. The problems of present centralized approaches are overcome by its highly distributed infrastructure. Centralized systems are vulnerable to collapse due to its single point of failure. As the brainchild of Bitcoin, blockchain exploded into the scene and it was only a matter of time, before we realized that it could be used for any asset and not only cryptocurrencies, i.e any entity of monetary value. Blockchain in simple terms is a secure, decentralized and a distributed ledger.

When applied in a holistic manner, blockchain can help solve firearm related issues. Gun tracking is a problematic situation in the United States (any country for that matter!). Privacy information important to gun-advocates and better legislation can be enforced through this approach. There are numerous parties involved behind the logistics of firearms manufacturing and sales. In this research the parties that we have identified that are most significant are listed in the system architecture section. A lot of data exchange and a notion of “trust” is involved in these exchanges. Hence we leverage the concept of permissioned/private blockchain, since besides being an enterprise blockchain application, delicate information is shared throughout and the usage of a private system would address this.

In order to replicate this use-case we have identified six main actors, which will act as the peers in the Fabric network. Each actor has a simple role in handling information which we have highlighted. A architectural overview of Hyperledger Fabric is presented in this paper,using which we have made the design of the Fabric network that contains these peers, and all connected to a single channel and the channel containing a single ordering service. In each of the peers, chaincodes have been installed based on smart contracts, the implementation details of which has been discussed in greater detail in the later sections of this paper. Finally we have addressed the security aspects and the analysis of this sample prototype.

II. Hyperledger Fabric - An Architectural Overview

For this research we leverage IBM/Linux’s Hyperledger Fabric. Hyperledger Fabric unlike other blockchain platforms, offers private/permissioned blockchain networks that allows only known participants to participate in the network by means of membership features. It comes with smart contracts, ledgers and protocols that is perfect for a private enterprise system. In reality enterprise systems have a lot of privacy issues and need to maintain data integrity between systems. We will present a brief overview about each of the significant components involved in Hyperledger Fabric.

A. Assets

Asset definitions enable the exchange of almost anything with monetary value over the network, from whole foods to antique cars to currency futures. Assets in Fabric are represented as key-value pairs. In this use-case it is pretty obvious that the asset is gun-related information.

B. Chaincode

Chaincode in Fabric is the actual business logic. It defines the asset and a transactional modification to that asset. Execution of a chaincode will change the state of the asset. Chaincode is nothing but the smart contract. It runs separately from transaction order which optimizes network security.

C. Channels

The very notion of privacy is Fabric is given through channels. It can be considered as a private subnet of communication between specific network members. Each channel has its own set of peers that are allowed to participate by proposing and receiving transactions. Each channel has a separate ledger which is stored in each peer.

D. Security and Membership Services

Fabric provides Membership Service Providers(MSP), which is used to identify which participant is allowed in the network and which is not. MSPs generally come in the form of X509 certificates i.e only identities with certificates of this specification is allowed to participate in the network. Through MSPs, access control and a trusted blockchain network is enforced.

E. Identities

In a Fabric network there are many identities involved like orderers, peers, administrators, clients. These identities get certified information saying that they are allowed in the network as specified in the Membership section. It is a digital version of certification of an actor in a blockchain network with various attributes.

F. Consensus

Fabric has a unique approach to consensus enables the flexibility and scalability needed for an enterprise application. Here there is something called "endorsement policy" that is used to validate transactions in the network. An endorsement policy can be as simple as two out of five participants have to accept a transaction for it to be complete.

G. Peer

Peers are the actual nodes in the blockchain network that host ledgers and smart contracts. An application connects with peers to interact with the ledger. Fabric has SDK that has APIs that makes it easy for applications to interact with peers provided they have a valid membership into the network. Peers in conjunction with orderers make sure the ledger state is kept upto date. Peers do transactions to update state of ledger or return ledger information (query) to the client.

H. Orderer

An orderer node is responsible for packaging a transaction received from a peer into blocks, order them properly and distribute/broadcast them to the peers for ledger update.

I. Ledger

The ledger in Fabric contains all state elements. One can think of it like a database that stores transaction logs in a chronological fashion.In Fabric the ledger is divided into two parts - world state and blockchain. Each represents facts about the business objects involved.There is one ledger per channel. Since this is a very important part of Fabric architecture, we have given the description of ledger in a separate section below.

III. Distributed Ledger

The distributed ledger in Hyperledger Fabric comprises of two parts - world State and the blockchain itself.In simple words a world state is a database that represents a set of facts relating to the current state of the blockchain. States are generally stored as key-value pairs, the blockchain is a transaction log that records changes in the world state. The world state can be modified, but the blockchain is immutable.

- World state: The ledger state that records facts about a business object in the form of key-value pairs.The world state is implemented as a database, which makes a lot of sense since it is useful for efficient storage and retrieval of states. The world state can also be modified. A simple operation like a get,put or delete can be invoked in a smart contract of an application. Update to the world state can be made only if there are sufficient endorsements to the transaction.

- Blockchain: The blockchain is a historical record of facts that is immutable. Each block in a blockchain contains contains a transaction representing a query or update made to the world state. The blockchain is typically implemented as a file. Each block in a blockchain consists of block header, transaction data and block metadata.

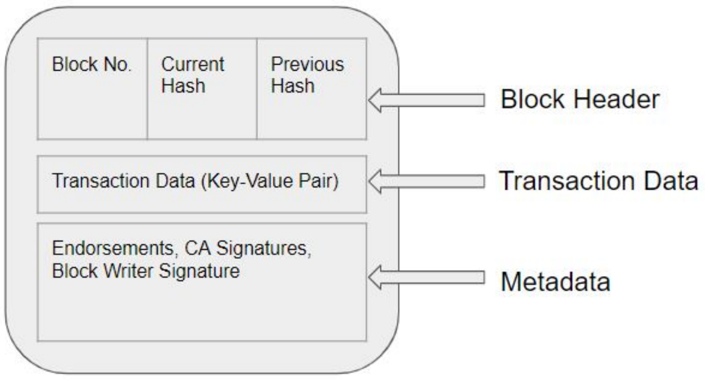

Block No.
Current Hash
Previous Hash
Block Header
Transaction Data (Key-Value Pair)
Transaction Data
Endorsements, CA Signatures, Block Writer Signature
Metadata

Fig. 1. Block Structure in Hyper-ledger Fabric

A client application (either one of the actors) will perform a transaction through a web interface. Every transaction done is added as a block (the size of a block can be varied in

Fabric) and is reflected in the ledger's world state of each of the other node's ledgers as well. Hyperledger provides a ledger that consists of two related parts - World State that stores the read-write sets of the transaction as key-value pairs and the blockchain itself which is a log of all transactions that enable us to see the state of the blockchain and is immutable. Since we are focused on the block itself, a high level view of what is contained in the block is presented in Fig 1. In this use-case a transaction (a firearm purchase with the key being the ID of the purchase and the value the details of the purchase) is added into a block. Block size related specifications can be configured in Fabric by varying batch sizes.

IV. Actor Functionalities

In this section, we have identified the various actors that we plan on implementing in this firearms tracking system. Each actor in this system is a peer that interacts with one or more peers whose specific roles and type of relationship with other actors are elaborated below.

A. Licensed Retailers

Licensed retailers have a direct relationship with almost all of the other peers. Licensed retailers interact with manufacturers in order to purchase firearms that are approved by the government for distribution and usage. A retailer can only purchase a gun if it has been approved by the government. Retailers also need to interact with users because they are the channel through which a customer can purchase firearms. This is possible only if the customer has no criminal history or medical illness in the past. This information is gathered by a retailer before issuing a firearm to a customer by doing a pre-background check. In order to perform this background check, it interacts with the other peers in the system such as the NICS, police and medical history team. The flow is described in greater detail in a later section of this proposal.

B. Firearms manufacturer

Whenever a manufacturer comes up with a new type of firearm, the government needs to be keep track of its license and add it to a list of licensed guns. Only after receiving an acknowledgement from the government can it sell its products to retailers. The legitimacy of manufacturers has been insured in our prototype through the use of certificates which have to be approved before a manufacturer is allowed to enter into the blockchain network. More security aspects have been illustrated in the Network and Privacy section of this proposal. Each peer also needs to maintain information regarding the retailer peers it does business with.

C. Police

This peer contains details of the criminal record of users, if any. This peer verifies that a potential customer has no criminal background before they can be issued a firearm license. This peer interacts with and provides information to the NICS who make their decision of issuing licenses based on the approval sent by this peer. In the future, we have plans of extending this peer's functionality to extract useful information quickly when crimes occur.

D. Medical Professionals

This peer interacts with the NICS as well and aids in the process of issuing licenses to cutomers. Surverys in the past indicate that some of the major causes of suicides is the possession of guns. Thus, care must be taken to ensure that people with mental illnesses and trauma shouldn't be issues firarms. This check is performed by the medical professional team who ensure that the potential customer is in stable mental health and can conduct any additional checks as well, if needed. This team's results and statistics on the health of a user will then be sent back to the NICS who will take this information into consideration before responding to the retailers.

E. NICS

NICS stands for National Instant Criminal Background Check System. This is a very important peer as it interacts with multiple peers. It contains details of the criminal records of potential customers, if any. It interacts with the retailer and performs the function of issuing licenses to potential firearms customers. For existing customers, it performs background checks to ensure that the customer is indeed legitimate. In case a user turns up in this database, then this peer also contacts the police for additional information and further review. It also interacts with medical professional peers to ensure that the approval from them are in order. Based on all these inputs, the NICS decides whether or not to approve the transaction and this information is passed on to the retailer.

F. Licensed Manufacturers

The government is responsible for maintaining a database of all licensed manufacturers and firearms as well as licensed firearms owners. It interacts with manufacturers to update its list of approved firearms. Only firearms that are approved by the government can be sold to retailers by the manufacturers. It also collaborates with the retailers and ensures that the potential customer has a valid license. If not, it takes care of the process of issuing a new license provided the customer has no criminal history or mental health conditions.

V. Transaction Function Flow

In this section, we would see the transaction function flows between the peers.

Flow 1: Customer - Retailer Flow

This flow is initiated when the customer buys the gun from the retailer, also known as a Federal Firearms Licensee (FFL). The retailer peer in this flow is a licensed retailer. At the retailer peer, the customer fills out an ATF form, customer's gun license, and all other details are forgathered at the retailer's peer. These details are forwarded to the NICS and the government peer. Where the details are verified, and the customer would be given authorization, if there are no holds against the customer. The NICS queries the Medical professionals peers, and Police Criminal control department peer to confirm that if there are any medical history or criminal records against the customer buying the gun. If the both Medical and the Police department peer acknowledge the queries, with a clear response, then the NICS peer sends the authorization for the customer. The government peer also verifies the customers gun license and sends authorization. After which the retailer makes the gun transaction to the user only when both the approval is received.

Flow 2: Manufacturer-Retailer flow

In this flow, the transaction is initiated when the manufacturer forges the gun. These guns are reported to the government peer, where it is tracked. On certifying the arms at the government peer, then the weapons are sold to the retailer. A reconfirmation is carried out at the retailer peer, in order to confirm if it is genuine consignment. Once the acknowledgment is received, the transaction is committed.

VI. Network Modelling

The previous section presented a holistic description of each of the actor's roles and some scenarios.The Fabric network is a technical infrastructure that provides ledger services to consumer and administrators. It is designed to support pluggable implementations of different components and accommodate the complexity and intricacies that exists across the economic ecosystem, in our case the actors and their roles defined previously. A network consists of ledgers, smart contracts, peers, orderers, channels, certificate authorities all of which were defined in the previous sections (Fig 2).

The blockchain network formulation that we have come up with, and implemented is as follows,

- There are six organizations in the network, i.e six peers for each of the actors in the ecosystem. Each peer is hosted in a separate port with a CA for security. Peers in Fabric are hosted as containers in Docker. The output of the peers and their container specifications of our implementation is given in the Fig 3. Thus we have six peers in our fire arms tracking system as shown in the overall architecture diagram.

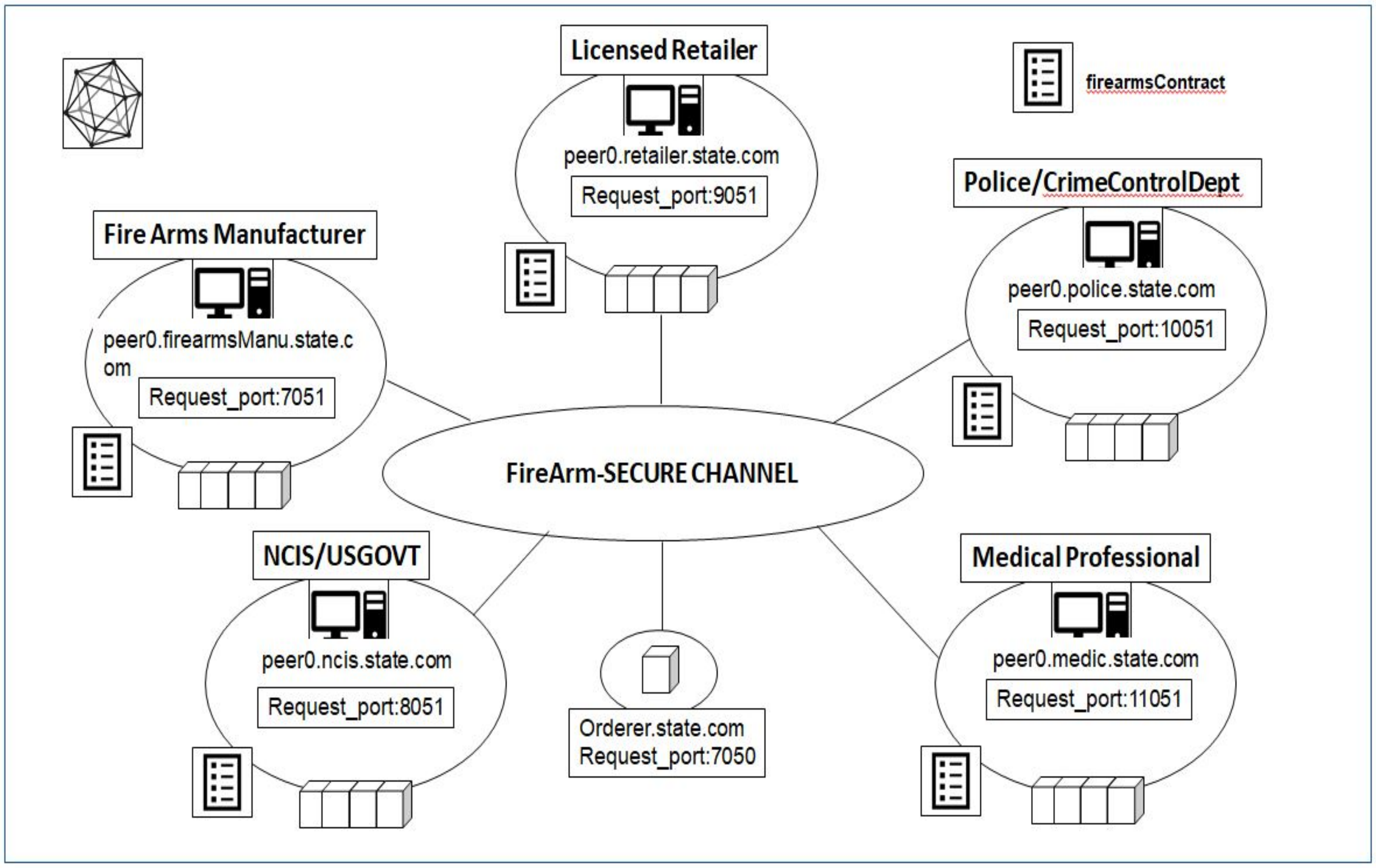
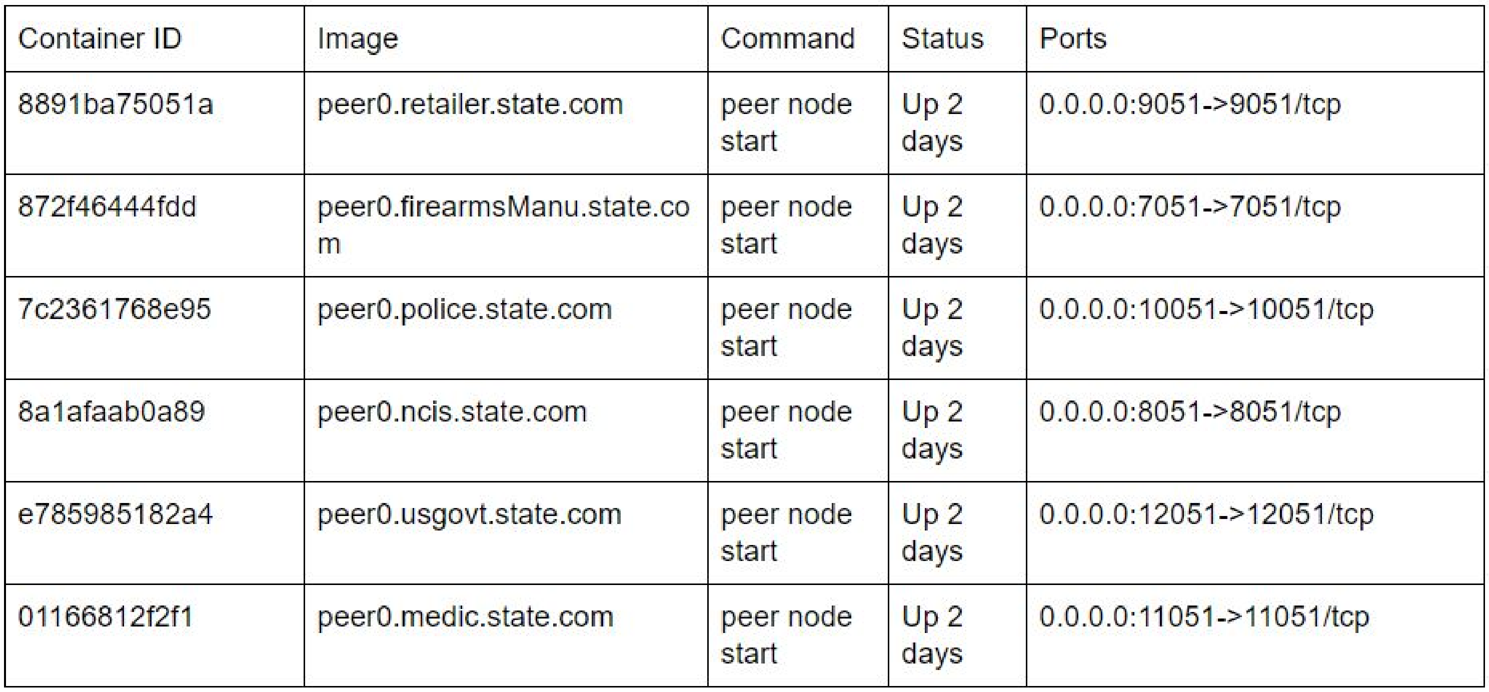


Fig. 2. Underlying Fabric Network



Container ID	Image	Command	Status	Ports
8891ba75051a	peer0.retailer.state.com	peer node start	Up 2 days	0.0.0.0:9051->9051/tcp
872f46444fdd	peer0.firearmsManu.state.com	peer node start	Up 2 days	0.0.0.0:7051->7051/tcp
7c2361768e95	peer0.police.state.com	peer node start	Up 2 days	0.0.0.0:10051->10051/tcp
8a1afaab0a89	peer0.ncis.state.com	peer node start	Up 2 days	0.0.0.0:8051->8051/tcp
e785985182a4	peer0.usgovt.state.com	peer node start	Up 2 days	0.0.0.0:12051->12051/tcp
01166812f2f1	peer0.medic.state.com	peer node start	Up 2 days	0.0.0.0:11051->11051/tcp

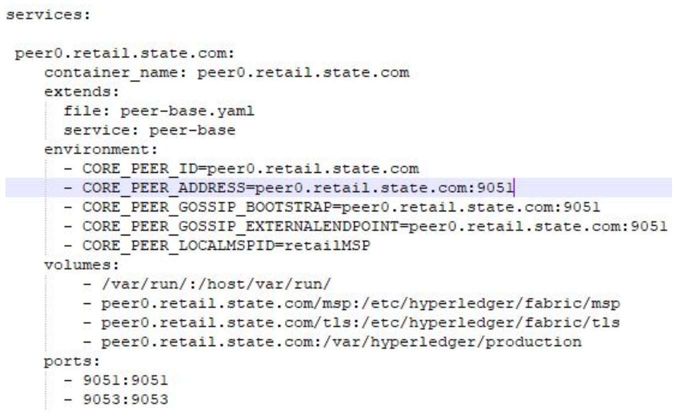
Fig. 3. Peer Container information

- We have one channel, "FireArm-Secure Channel", so one ledger that is distributed among the peers connected to the channel.
- Each peer is installed with a sample chaincode that contains the smart contracts called "firearmsContract" defined in Fig 5. The next sections of our research is development of the smart contracts for each of the defined functionalities of the nodes. Smart contracts is developed in GoLang.
- An orderer - orderer.example.com is defined in port 7050.
- The world state definition of our channel ledger will be represented as key value pairs, i.e a transaction is nothing but firearm related information represented as say , an ID of a purchase(as key) and its details as value represented in a JSON format. Again this is part of the smart contract development given in the next sections of our research

Thus our blockchain network has been modelled with the configuration as given above. The peers, channels, orderer and chaincode are setup and running as Docker containers as shown in the tables below. Using SDKs provided by Fabric one can access control over the network and manipulate it, provided they have the necessary cryptographic material to authenticate to the network.

VII. Network Implementation

We have defined our actor peers, their roles, some scenarios and modelled the network relative to our description in the previous sections. Here we present a brief specification of what we have implemented, ie. the underlying blockchain network that we have created. Output screens are given in figures 5,4



```
services:

 peer0.retail.state.com:
    container_name: peer0.retail.state.com
    extends:
      file: peer-base.yaml
      service: peer-base
    environment:
      - CORE_PEER_ID=peer0.retail.state.com
      - CORE_PEER_ADDRESS=peer0.retail.state.com:9051
      - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.retail.state.com:9051
      - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.retail.state.com:9051
      - CORE_PEER_LOCALMSPID=retailMSP
    volumes:
        - /var/run/:/host/var/run/
        - peer0.retail.state.com/msp:/etc/hyperledger/fabric/msp
        - peer0.retail.state.com/tls:/etc/hyperledger/fabric/tls
        - peer0.retail.state.com:/var/hyperledger/production
    ports:
      - 9051:9051
      - 9053:9053
```

Fig. 4. Peer config in YAML File

- In order to develop the network in Hyperledger Fabric the teechnology components that are needed are - 1) Docker 2) GoLang. Docker as mentioned is used to maintain Fabric components as containers. Go Language is used to develop chaincode for smart contracts.
- Configuration in Fabric is specified in YAML language. YAML is a human-readable data-serialization language. It is commonly used for configuration files and in applications where data is being stored or transmitted. YAML

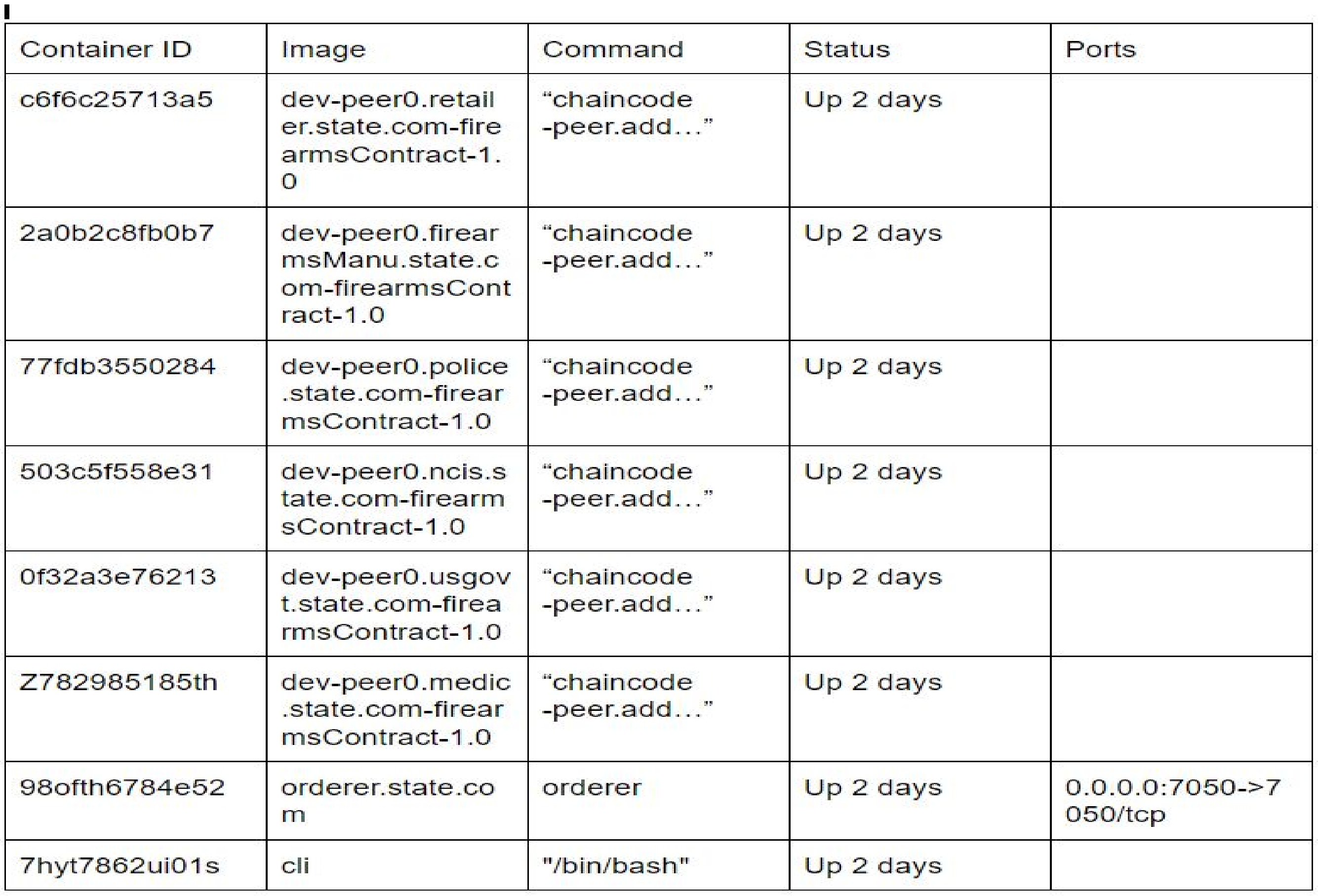
files have minimal syntax. Hyperledger Fabric requires us to put network specific configuration data in YAML files and call Docker commands on them to start them as containers.

We first setup the Fabric binaries provided as open-source (version 1.4). The binaries include peers,orderers,CAs,couchdb,cryptogen,configtxlator etc.

- In order to generate Crypto material required for the network entities, we use Cryptogen tool provided by Fabric as a binary, to create certificates, (in a real world scenario these certificates must come from proper certifying authorities). We specify configuration information in a file called 'crypto-config.yaml' for each of the peers and orderer for which crytographic material is generated. A sample configuration specific to what we implemented is given below.

```
" - Name: Nics
  Domain: nics.state.com
  EnableNodeOUs: true
  Template:
     Count: 1
  Users:
     Count: 1 "
```

  The above configuration will create Crypto material for peer0 in org nics.state.com
- The Configtxgen tool is used to generate four configuration artifacts - 1) Orderer genesis block(the 0th block) 2) channel configuration 3) anchor peer configuration by consuming configtx.yaml file.
- The peer and orderer configurations including environment variables are given in yaml files and started as containers by consuming the yaml files using the "docker-compose" command. Configuration specific to each peer is specified as environment variables, which is automatically parsed by Docker and started up as containers. A sample configuration specific to what we have implemented is given in Fig 4



Container ID	Image	Command	Status	Ports
c6f6c25713a5	dev-peer0.retailer.state.com-firearmsContract-1.0	"chaincode -peer.add..."	Up 2 days	
2a0b2c8fb0b7	dev-peer0.firearmsManu.state.com-firearmsContract-1.0	"chaincode -peer.add..."	Up 2 days	
77fdb3550284	dev-peer0.police.state.com-firearmsContract-1.0	"chaincode -peer.add..."	Up 2 days	
503c5f558e31	dev-peer0.ncis.state.com-firearmsContract-1.0	"chaincode -peer.add..."	Up 2 days	
0f32a3e76213	dev-peer0.usgovt.state.com-firearmsContract-1.0	"chaincode -peer.add..."	Up 2 days	
Z782985185th	dev-peer0.medic.state.com-firearmsContract-1.0	"chaincode -peer.add..."	Up 2 days	
98ofth6784e52	orderer.state.com	orderer	Up 2 days	0.0.0.0:7050->7050/tcp
7hyt7862ui01s	cli	"/bin/bash"	Up 2 days	

Fig. 5. Chain Code , Orderer, CLI Container information

- The next step involves having each peer have the chaincode installed and initiated. Applications interact with the blockchain ledger through chaincode. As mentioned before we need to have GO installed and chaincode install takes the relative path from GOPATH source. A

sample command specific to our implementation to install a chaincode is given below,
"peer chaincode install -n firearmsContract -v 1.0 -p github.com/chaincode/firearmsContract/go/"

As per the given configurations and model we have created our blockchain network whose specification is given in the figures.

VIII. Manipulating blockhain network using Fabric SDK

Fabric provides an API for interacting with the blockchain network through SDK. We are using Java SDK for simulation purposes. One can manage the underlying the Fabric network using the SDK provided. Using the SDK we will also be able to execute, install chaincodes on the respective peers and also define endorsement policies. The SDK works on behalf of a particular User Admin peer in our case we are considering peer0.retailer.state.com, given admin privileges to manage and access the channels, provided it is given the proper cryptographic material(specifics mentioned in section XI). Our chaincode development(next sections) is done in GoLang. A view of our flow is given in Appendix. The client, using the Java SDK will access the underlying blockchain network that we have implemented. In other words the SDK is used for an application level simulation (or rather a web interface).

IX. Chaincode - Smart Contract Development

Since we are focused on the smart contract development through chaincode in Fabric, we will see in the next few sections about the chaincode development and the application views. A small description about what a chaincode is given in one of the previous sections. We will see it now in more detail.

A chaincode is a program that is written to read and update from the ledger state. All the business logic is contained inside the chaincode and this is the program where we define our smart contracts. We saw two sample flows in the section Transaction Function Flow. All the business logic required to execute the scenarios mentioned in that section is defined in the chaincode. Chaincode is a programmable code that is written in GoLang, and instantiated on a channel. Application developers user chaincode to develop business contracts that define assets and for management of participants. It typically updates the world state of the ledger in Fabric. Therefore chaincode needs to be installed on every peer that will endorse a transaction and instantiated on the channel.

Two key APIs that are used when writing chaincode development are

- ChaincodeStub
- ChaincodeStubInterface

The chaincode stub provides functions that allows us interact with the ledger to put,get,update and delete state. The key functions in the chaincode that we have included are,

- **func (stub *ChaincodeStub) GetState(key string) ([]byte, error)** - This function returns a value of the specified key from the ledger. We have to note here that the GetState doesnt read data from the WriteSet that hasnt been commited to the ledger.
- **func (stub *ChaincodeStub) PutState(key string, value []byte) error** - Puts the specified key and value into the transaction's Write set as a data-write proposal. PutState doesn't affect the ledger until the transaction is validated and successfully committed.

There are two methods that are to be implemented,

- init is used to initialize the state of any application
- invoke is used to process transaction proposals.

We must create init and invoke transactions in a chaincode. One can do chaincode operations to a peer, by using the commands 'peer chaincode install', 'peer chaincode instantiate', 'peer chaincode invoke' or 'peer chaincode query'. As the commands clearly imply, we need to install chaincode on the peer, instantiate the chaincode during which we specify endorsements required, invoke the chaincode to do transactions on the world state and blockchain and query from the world state.

X. Chaincode Flow - Firearms Application

The chaincode section gave an overview of what the chaincode is, and the typical functions and interfaces required to model the smart contract for update to the ledger states. This section will present how we modelled our smart contract functions for the scenarios that we mentioned before along with the inherent flow involved. As explained before, the blockchain network in Fabric has six stakeholders for this particular Firearms usecase. We will also see briefly how we have used the JavaSDK to manipulate the underlying Firearms network from a client perspective.

To summarize the overall flow of the chaincode system, two types of information - the gun related information, and a customer related information (for a customer who purchases from a retailer) is mainly recorded in the ledger state for tracking. We initialize the ledger with these structures. The stakeholders in the process get access to the network with their MSP credentials through which they can do operations to the ledger. The definitions required for our use case scenarios are defined in the chaincode file written in GoLang. After this, the chaincode is installed and instantiated on the peers. The peers, i.e., the application clients then access this chaincode and call its functions. Endorsements are done as part of this execution, where in every endorsing peer executes the chaincode and makes sure that the signatures are verified. We recall that if even a single peer does not execute the transaction fully, then the state is committed into the Blockchain, that is there is no consensus. This essentially makes sure that the parties are bounded by a contract and are obligated towards it. The endorsed executions are given to the orderer and commited to the Blockchain. To query the ledger state we can issue commands provided we have the MSP credentials and get the state from the Blockchain. Note that a query to the Blockchain state (i.e the getState does not require services of the orderer, since there is not addition to the ledger). This summarizes the typical flow of the Firearms chaincode logic that we have used, further shown in Fig 6.

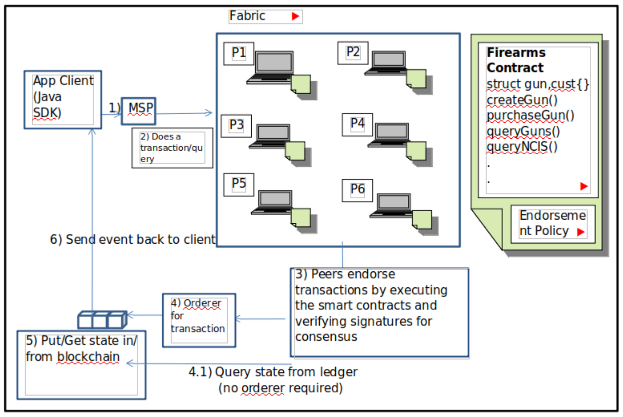

Fabric
P1
P2
P3
P4
P5
P6
Firearms Contract
struct gun,cust{}
createGun()
purchaseGun()
queryGuns()
queryNCIS()
Endorsement Policy
App Client (Java SDK)
1)
MSP
2) Does a transaction/query
6) Send event back to client
3) Peers endorse transactions by executing the smart contracts and verifying signatures for consensus
4) Orderer for transaction
5) Put/Get state in/ from blockchain
4.1) Query state from ledger (no orderer required)

Fig. 6. Chaincode Flow Logic

Let us analyze how the customer-retailer flow works in accordance with the technical aspects of the smart contract along with some code snippets. The flow starts with the retailer interface, gets the details of a customer and posts the details into the ledger state for verification from various other peers. A call to the smart contract execution is invoked from the Java SDK, by means of QueryProposalRequest class. The QueryProposalRequest queries the state for the respective values from the chaincode in the network. The code snippet below contains the call of the chaincode from the Java SDK client. Note that the smart contract's name is denoted by "facc", and the argument we pass is the social security number of the peer.

Listing 1: QueryProposal request for verification from peers

```
QueryByChaincodeRequest request = BlockChainHFClient.getInstance().getClient().
                                  newQueryProposalRequest();
Channel channel = BlockChainHFClient.getInstance().getCh();
ChaincodeID ccid = ChaincodeID.newBuilder().setName("facc").build();
request.setChaincodeID(ccid);
request.setFcn("queryNcis");
String args = null;
if (args != null)
        request.setArgs(args);
Collection<ProposalResponse> response = channel.queryByChaincode(request);
```

The Collection ⟨ProposalResponse ⟩response variable contains the response received back from the chaincode, and the stack trace from the chaincode installed on the respective peers. The entire function returns an "OK" response if the customer is eligible to purchase from the retailer. The proposal response payload in general contains the values (the read-write sets) of the ledger state. As per the terms of our smart contract, the details need to be verified by the NCIS for approval and confirmation. Therefore the smart contract function that implements this functionality is queryNcis(),for which a code snippet is given in Listing 2.

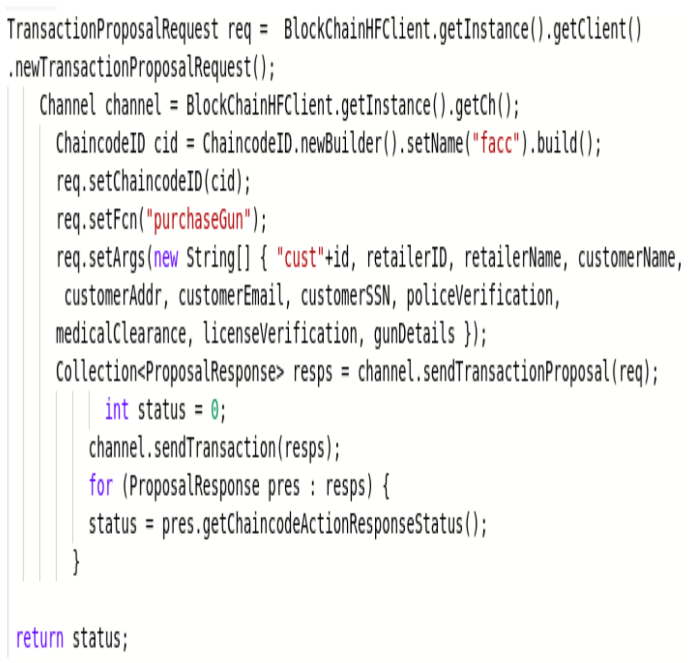
The chaincode from listing 2 takes in the SSN number of the customer, and sends it to the verifySSN() function to check if it is a person with a valid social security number. The verify function calls the NCIS peer, which will have its own contract set up (we recall that a peer can have one or more chaincodes). This contract at the NCIS peer in addition to our firearms contract, will get the customer data from its own private database, and sends it to the firearms contract. This is an ideal real world scenario. For the purpose of this simulation, we have given priority to the firearms smart contract in particular and have given some sample customer entries for verification.

Once the customer is verified, getCustAsBytes() returns the customer details that contains a Boolean flag along with it saying he or she exists. After this , three calls to three other different peers is made – government peer, police peer, medic peer. These three peers will reply with a boolean variable containing information confirming whether he/she has a valid license, no criminal background and whether the person is mentally stable respectively. This scenario is similar to the NCIS peer in a real world scenario. Each peer will have a separate contract to query their own private database to return these values to the firearms contract. Once verification is done from the three peers, we will append all the responses to a JSON object and return it back to the client SDK.

Endorsements in Fabric are a way of ensuring that consensus is maintained between the stakeholders that are involved. Endorsement policies define which peers need to agree on results or ledger states before adding in the ledger. Essentially endorsement involves execution of each peer's respective chaincode and signing transactions. In this particular use case every peer is an endorser peer and have to execute their smart contract, if it executes, successfully then state is committed or retrieved from the ledger, else it is cancelled. So the flow starts with the client who assembles all endorsements from the peers into transactions and broadcasts them to the ordering service which adds it as a block. This is then committed into the blockchain state. Endorsement policies are specified during substantiation of chaincode.

Listing 2: Verification through NCIS and various parties Query Smart contract code

```
func (s *SmartContract) queryNcis(APIstub shim.ChaincodeStubInterface, args []string)
sc.Response {
  if len(args) != 1 {
      return shim.Error("Incorrect␣number␣of␣arguments.␣Expecting␣1")
}
ssnNumber = args[0]
getCustAsBytes, err := s.verifySSN(args[0])
//getCustAsBytes has bool variable indicating if customer data is
//there or not from the NCIS peer
if err != nil {
     jsonResp := "{\"Error\":\"Failed␣to␣get␣state␣for␣" + ssnNumber + "\"}"
     return nil, errors.New(jsonResp)
}
if getCustAsBytes == nil {
     jsonResp := "{\"Error\":\"Nil␣info␣for␣" + ssnNumber + "\"}"
     return nil, errors.New(jsonResp)
}
hasValidLicense := s.queryGovernment(ssnNumber)
//verifying if user has valid license
isBackgroundVerified := s.queryPolice(ssnNumber)
//verifying the user doesn't have any criminal background
isHealthy := s.queryMedicalProfessional(ssnNumber)
//ensuring the user is mentally stable
jsonResp := "{\"hasValidLicense\":" + hasValidLicense +",\"isBackgroundVerified\":"
+ isBackgroundVerified +",\"isHealthy\":" + isHealthy"}"
return jsonResp
}
```



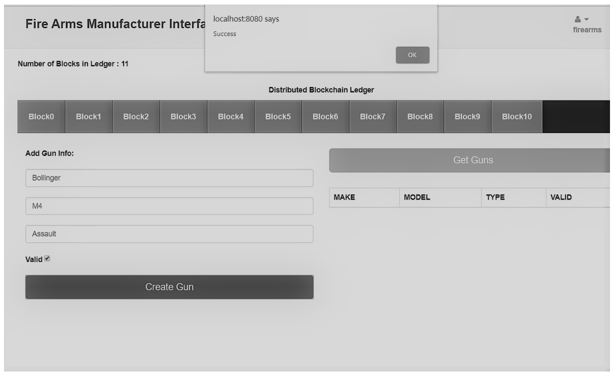
```
TransactionProposalRequest req =  BlockChainHFClient.getInstance().getClient()
.newTransactionProposalRequest();
    Channel channel = BlockChainHFClient.getInstance().getCh();
      ChaincodeID cid = ChaincodeID.newBuilder().setName("facc").build();
      req.setChaincodeID(cid);
      req.setFcn("purchaseGun");
      req.setArgs(new String[] { "cust"+id, retailerID, retailerName, customerName,
       customerAddr, customerEmail, customerSSN, policeVerification,
      medicalClearance, licenseVerification, gunDetails });
      Collection<ProposalResponse> resps = channel.sendTransactionProposal(req);
          int status = 0;
        channel.sendTransaction(resps);
        for (ProposalResponse pres : resps) {
        status = pres.getChaincodeActionResponseStatus();
      }

 return status;
```

Fig. 7. TransactionProposal Request for purchase gun once verification is done

After successful verifications obtained from the peers , the gun can be purchased or in blockchain terms, a transaction can be made with a state addition to the ledger. Once the purchase button is clicked in the client interface the SDK takes in the data, and we invoke a transaction through a TransactionProposalRequest. This request is sent to all the peers. Once the peers execute the transactions and endorses them, we get it back as a ProposalResponse. This contains the response payload and the status. If it is fine, this gets sent to the channel's orderer service which organizes the transaction execution result as a block and adds it to the blockchain. The code snippet is showing in Fig 7.

After receiving the response objects, the flow of control goes back to the smart contract functionality. The smart contract, as shown in Fig 14, takes in customer details and the gun details as arguments to the function. It then unmarshals the gun details and gets the state from the world state, to check if the gun is indeed valid. If it is true (valid), then the customer details are added in the World State using putState(). This response is sent back to the client SDK and then sent to the other peers through the channel and orderer service.

The second scenario that we implemented in our application is that of the manufacturer-retailer relationship. As elaborated before, this flow ensures that everytime the manufacturer issues a new gun, its records are sent to the government and the retailer peers. The smart contracts are used to add a gun to the world state and query all guns from the world state. When the firearms manufacturer creates a new gun, it adds it through the createGun() function in the chaincode. The core logic is similar to the previous scenario. At the retailer peer's end, we do an additional reconfirmation to ensure that the gun is genuine and has a proper license.



localhost:8080 says

Success

Number of Blocks in Ledger : 11

Add Gun Info:

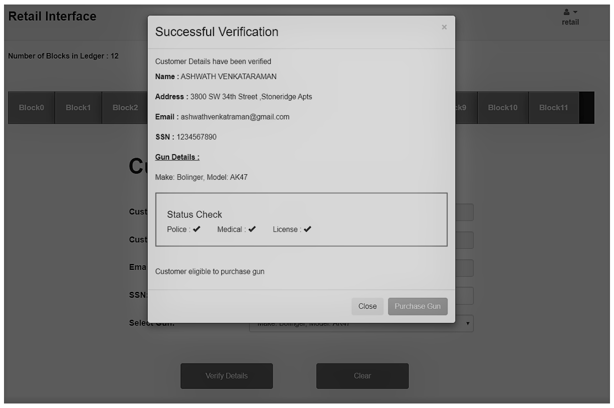
Bollinger

M4

Assault

Valid ☑

Fig. 8. Manufacturer interface for issuing new guns



Number of Blocks in Ledger : 12

Successful Verification

Customer Details have been verified

Name : ASHWATH VENKATARAMAN

Address : 3800 SW 34th Street ,Stoneridge Apts

Email : ashwathvenkatraman@gmail.com

SSN : 1234567890

Gun Details :

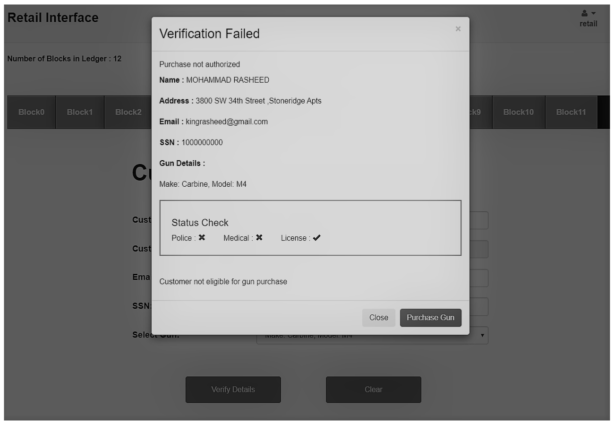
Make: Bolinger, Model: AK47

Status Check

Police : ✓ Medical : ✓ License : ✓

Customer eligible to purchase gun

Fig. 9. Successful Verification of customer details



Number of Blocks in Ledger : 12

Verification Failed

Purchase not authorized

Name : MOHAMMAD RASHEED

Address : 3800 SW 34th Street ,Stoneridge Apts

Email : kingrasheed@gmail.com

SSN : 1000000000

Gun Details :

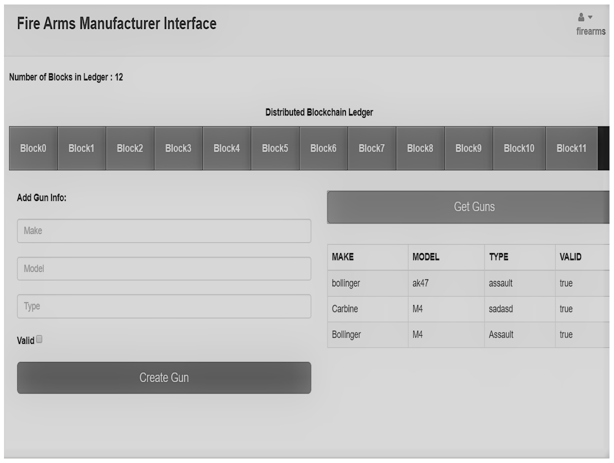
Make: Carbine, Model: M4

Status Check

Police : ✗ Medical : ✗ License : ✓

Customer not eligible for gun purchase

Fig. 10. Failed Verification of customer details



Number of Blocks in Ledger : 12

Add Gun Info:

Make

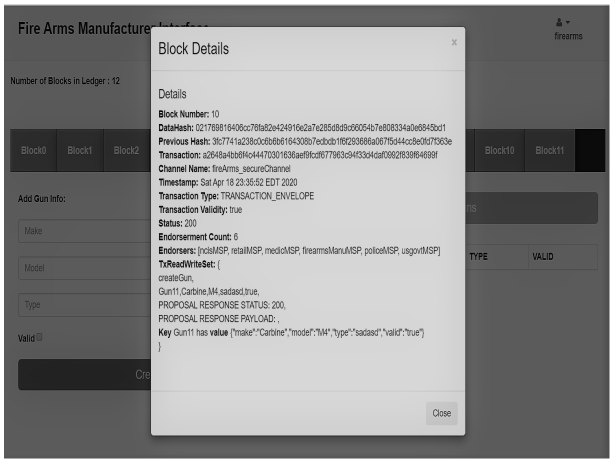
Model

Type

Valid ☐

MAKE	MODEL	TYPE	VALID
bollinger	ak47	assault	true
Carbine	M4	sadasd	true
Bollinger	M4	Assault	true

Fig. 11. Manufacturer interface for viewing existing guns



Number of Blocks in Ledger : 12

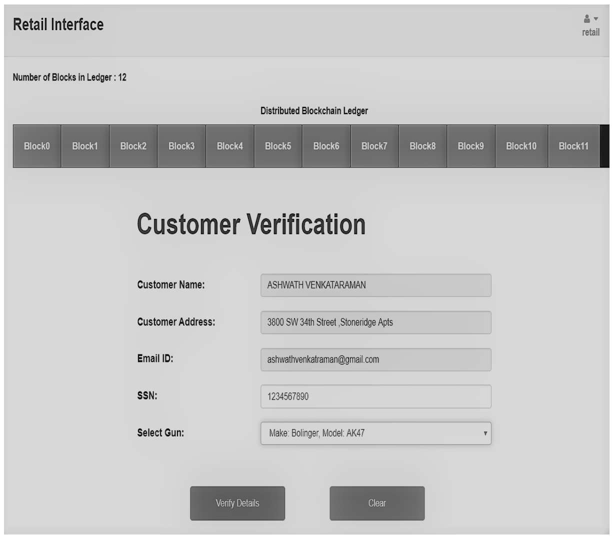
Add Gun Info:

Block Details

Details

```
Block Number: 10
DataHash: 021769816406cc76fa82e424916e2a7e285d8d9c66054b7e808334a0e6845bd1
Previous Hash: 3fc7741a238c0c6b6b6164308b7edbdb1f6f293686a067f5d44cc8e0fd7f363e
Transaction: a2648a4bb6f4c44470301636aef9fcdf677963c94f33d4daf0992f839f64699f
Channel Name: fireArms_secureChannel
Timestamp: Sat Apr 18 23:35:52 EDT 2020
Transaction Type: TRANSACTION_ENVELOPE
Transaction Validity: true
Status: 200
Endorserment Count: 6
Endorsers: [ncisMSP, retailMSP, medicMSP, firearmsManuMSP, policeMSP, usgovtMSP]
TxReadWriteSet: {
createGun,
Gun11,Carbine,M4,sadasd,true,
PROPOSAL RESPONSE STATUS: 200,
PROPOSAL RESPONSE PAYLOAD: ,
Key Gun11 has value {"make":"Carbine","model":"M4","type":"sadasd","valid":"true"}
}
```

Fig. 12. Block details of a firearm



Number of Blocks in Ledger : 12

Customer Verification

Customer Name: ASHWATH VENKATARAMAN

Customer Address: 3800 SW 34th Street ,Stoneridge Apts

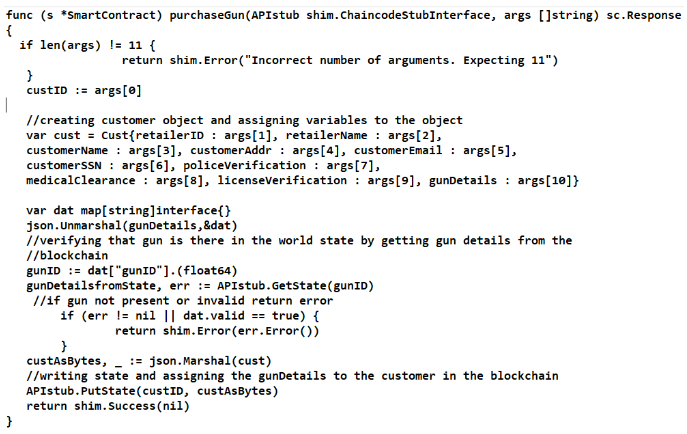
Email ID: ashwathvenkatraman@gmail.com

SSN: 1234567890

Select Gun: Make: Bolinger, Model: AK47

Fig. 13. Customer interface for purchase of firearms

Fig 8 through 13 show screenshots of the application in action. Screenshots depicting the various features of the application implemented by us has been showcased.



```
func (s *SmartContract) purchaseGun(APIstub shim.ChaincodeStubInterface, args []string) sc.Response
{
  if len(args) != 11 {
                return shim.Error("Incorrect number of arguments. Expecting 11")
   }
   custID := args[0]

   //creating customer object and assigning variables to the object
   var cust = Cust{retailerID : args[1], retailerName : args[2],
   customerName : args[3], customerAddr : args[4], customerEmail : args[5],
   customerSSN : args[6], policeVerification : args[7],
   medicalClearance : args[8], licenseVerification : args[9], gunDetails : args[10]}

   var dat map[string]interface{}
   json.Unmarshal(gunDetails,&dat)
   //verifying that gun is there in the world state by getting gun details from the
   //blockchain
   gunID := dat["gunID"].(float64)
   gunDetailsfromState, err := APIstub.GetState(gunID)
    //if gun not present or invalid return error
        if (err != nil || dat.valid == true) {
                return shim.Error(err.Error())
        }
   custAsBytes, _ := json.Marshal(cust)
   //writing state and assigning the gunDetails to the customer in the blockchain
   APIstub.PutState(custID, custAsBytes)
   return shim.Success(nil)
}
```

Fig. 14. Smartcontract code for purchaseGun after transaction proposal request

XI. Privacy and Security in Implementation

In the blockchain setting, every actor who wishes to interact with the network needs an identity. Hyperledger Fabric is a permissioned blockchain and offers a Membership Service Provider (MSP) feature that helps to ascertain whether the identity is reliable or not. Public Key Infrastructure (PKI) is a technology for authenticating users and a Digital Certificate is provided to the identity by a certificate issuer known as Certificate Authority (CA) in this model. MSP determines which of these identities provided by CA can participate in the network. Some of the most popular CAs are GoDaddy, Symantec, GeoTrust, and DigiCert among others. Fabric provides a built-in CA component (we have used Cryptogen tool to generate certificates for our network components) that allows us to create CAs in the blockchain networks formed.

This component known as Fabric CA can be used to provide and manage certificates. A Certificate Revocation List (CRL) consists of a list of certificates that a CA knows has to be revoked for various reasons. Hence CRL is used by the CA to check if a certificate is revoked or not. Hence this way when the CAs let other members enter the platform, they are automatically trusted. We will see how we secure data that is stored as plaintext in the world state, how and one can access the underlying network given the cryptographic material in through the SDK as a client.

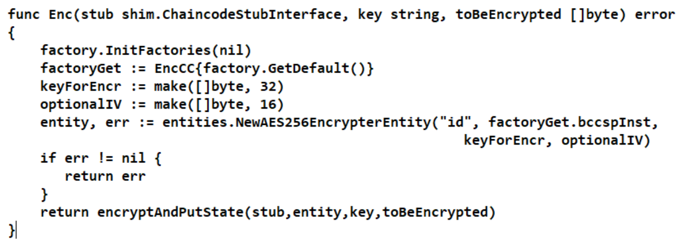
-> Although the network implementation considers the privacy and security of data in the network as a priority, all of the data that is stored in the state is plain-text. It is not a good practice to store the critical data as plain text. Therefore we are using a certain cryptographic algorithm provided by fabric to implement encryption. In particular we are using the BCCSP (Blockchain Cryptographic Service Provider) package that offers the implementation of cryptographic standards and algorithms. BCCSP uses the AES256 algorithm for encryption and will be imported in our chaincode that will be implemented in GO as part of smart contract development. We will see how this is done in detail. Also note that data stored in the blockchain itself is encrypted, however data stored in the state is plain text and can be prone to attacks.

Since data manipulation with respect to the world state is done through the chaincode, to ensure chaincode-level encryption, Fabric provides 3 such packages namely,

- BCCSP
- BCCSPFactory
- Entities

The **B**lock**c**hain **C**ryptographic **S**ervice **P**rovider is the package that offers services cryptographic algorithms, BCCSPFactory is used to get object instances of the BCCSP. Entities provides interfaces and packages to all crypto entities for data level encryption in the chaincode. The primary functionalities achieved through this are encryption of data and storing to state, and decryption of data after getting it from the state.

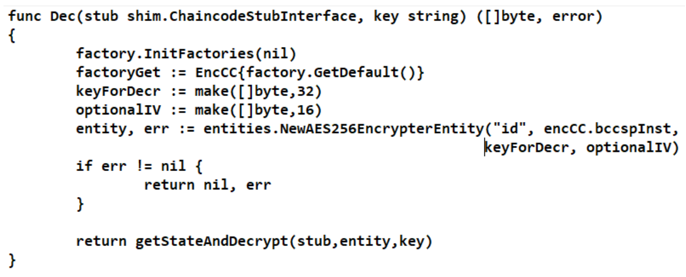
The **encryption** wrapper works like this. Once we import all packages mentioned above, we initialize the encryption signature algorithm. We then create a key that will be passed as a 32 byte array (AES 256). We also use an intialization vector for the blockcipher which is optional. The key string is then encrypted using these parameters and then added to the state using the usual putState() (Fig 15).



```
func Enc(stub shim.ChaincodeStubInterface, key string, toBeEncrypted []byte) error
{
    factory.InitFactories(nil)
    factoryGet := EncCC{factory.GetDefault()}
    keyForEncr := make([]byte, 32)
    optionalIV := make([]byte, 16)
    entity, err := entities.NewAES256EncrypterEntity("id", factoryGet.bccspInst,
                                                           keyForEncr, optionalIV)
    if err != nil {
       return err
    }
    return encryptAndPutState(stub,entity,key,toBeEncrypted)
}
```

Fig. 15. Encryption Wrapper

The **decryption** wrapper works in similar way by getting the key for decryption, the optional IV and calls the factory method to decrypt data which is got from the state. The two methods encryptAndPutState() and getStateAndDecrypt() are utility methods that contain the original getState() and putState() functions and use the entity to invoke Encrypt() and Decrypt() correspondingly (Fig 16).

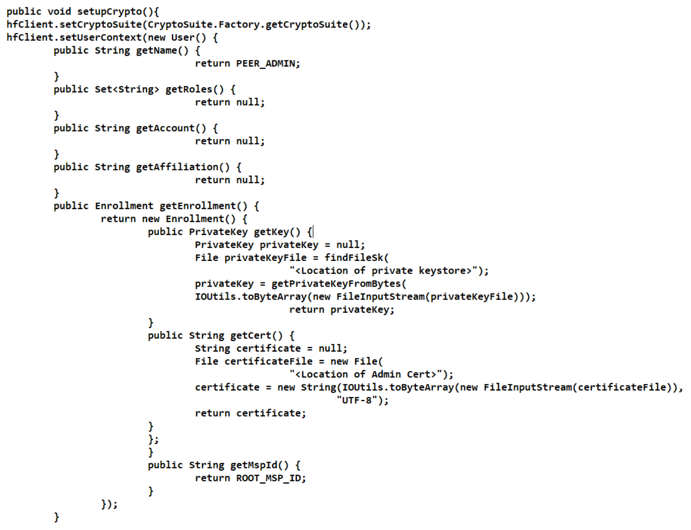


```
func Dec(stub shim.ChaincodeStubInterface, key string) ([]byte, error)
{
        factory.InitFactories(nil)
        factoryGet := EncCC{factory.GetDefault()}
        keyForDecr := make([]byte,32)
        optionalIV := make([]byte,16)
        entity, err := entities.NewAES256EncrypterEntity("id", encCC.bccspInst,
                                                          keyForDecr, optionalIV)
        if err != nil {
                return nil, err
        }

        return getStateAndDecrypt(stub,entity,key)
}
```

Fig. 16. Decryption Wrapper

Calling these functions from the original chaincode contract 'firearmsSmartContract' encrypts and decrypts plaint-text data to ensure further security.

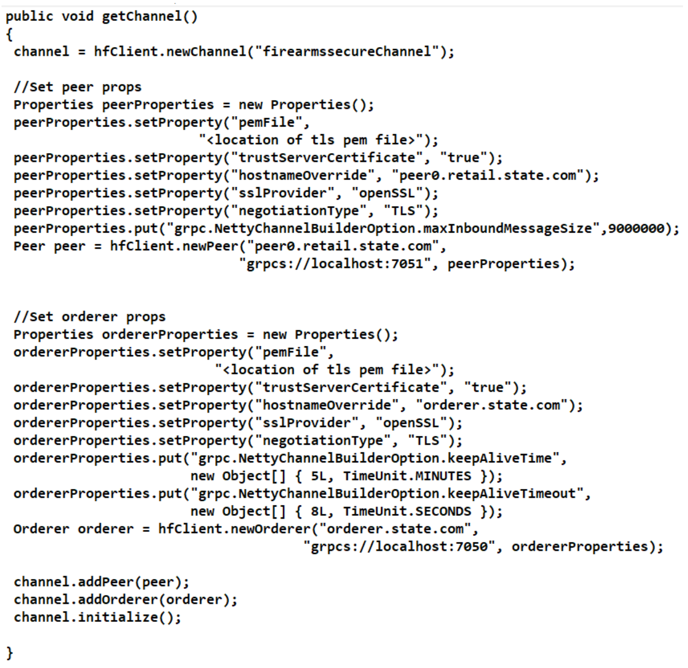
-> The term MSP is frequently mentioned throughout this report. The Membership Service Provider is nothing but a certificate of a specific format (x509) format that constitutes a root of trust. The MSP defines a set of certificates for the role it plays (for example admin). The certicates are typically pem extension files that is of format —BEGIN CERTIFICATE— *** —END CERTIFICATE— These credentials once issued and used are verified by Fabric to make sure that it is of proper format. Now to access the underlying blockchain network through the SDK (Java in our case), one needs to set the context of the user accessing, for example the user has to be an admin to access the network and the channel containing the respective peers and orderer. The code snippet to access the network as a client is given in Fig 17.



```
public void setupCrypto(){
hfClient.setCryptoSuite(CryptoSuite.Factory.getCryptoSuite());
hfClient.setUserContext(new User() {
        public String getName() {
                                return PEER_ADMIN;
        }
        public Set<String> getRoles() {
                                return null;
        }
        public String getAccount() {
                                return null;
        }
        public String getAffiliation() {
                                return null;
        }
        public Enrollment getEnrollment() {
                return new Enrollment() {
                        public PrivateKey getKey() {
                                PrivateKey privateKey = null;
                                File privateKeyFile = findFileSk(
                                        "<Location of private keystore>");
                                privateKey = getPrivateKeyFromBytes(
                                IOUtils.toByteArray(new FileInputStream(privateKeyFile)));
                                        return privateKey;
                        }
                        public String getCert() {
                                String certificate = null;
                                File certificateFile = new File(
                                        "<Location of Admin Cert>");
                                certificate = new String(IOUtils.toByteArray(new FileInputStream(certificateFile)),
                                             "UTF-8");
                                return certificate;
                        }
                        };
                        }
                        public String getMspId() {
                                return ROOT_MSP_ID;
                        }
                });
        }
```

Fig. 17. Setting up user context in SDK to access Network as client

The above code snippet gets the Fabric client object, sets up the user context. Using the certificate provided we then get an enrollment into the network. If the enrollment is successful, we get access to manipulate the network. Now that we have the necessary cryptographic material we can now access the channel and its components - peers and orderer. To this we need to supply the pemFile, hostname and invoke the peer or the orderer through its hostname that was given during network modelling. We add the specifications in a property object and initialize the channel. The code snippet is shown in Fig 18. We access the Docker containers through a grpcs call.



```
public void getChannel()
{
 channel = hfClient.newChannel("firearmssecureChannel");

 //Set peer props
 Properties peerProperties = new Properties();
 peerProperties.setProperty("pemFile",
                        "<location of tls pem file>");
 peerProperties.setProperty("trustServerCertificate", "true");
 peerProperties.setProperty("hostnameOverride", "peer0.retail.state.com");
 peerProperties.setProperty("sslProvider", "openSSL");
 peerProperties.setProperty("negotiationType", "TLS");
 peerProperties.put("grpc.NettyChannelBuilderOption.maxInboundMessageSize",9000000);
 Peer peer = hfClient.newPeer("peer0.retail.state.com",
                             "grpcs://localhost:7051", peerProperties);


 //Set orderer props
 Properties ordererProperties = new Properties();
 ordererProperties.setProperty("pemFile",
                        "<location of tls pem file>");
 ordererProperties.setProperty("trustServerCertificate", "true");
 ordererProperties.setProperty("hostnameOverride", "orderer.state.com");
 ordererProperties.setProperty("sslProvider", "openSSL");
 ordererProperties.setProperty("negotiationType", "TLS");
 ordererProperties.put("grpc.NettyChannelBuilderOption.keepAliveTime",
                   new Object[] { 5L, TimeUnit.MINUTES });
 ordererProperties.put("grpc.NettyChannelBuilderOption.keepAliveTimeout",
                   new Object[] { 8L, TimeUnit.SECONDS });
 Orderer orderer = hfClient.newOrderer("orderer.state.com",
                                    "grpcs://localhost:7050", ordererProperties);

 channel.addPeer(peer);
 channel.addOrderer(orderer);
 channel.initialize();

}
```

Fig. 18. Initializing channel through SDK after getting crypto material

Thus our Firearms application provides essentially a three-layer security from the underlying blockchain network level (consisting of the participants), channel and the chaincode data level.

- We have used a private/permissioned blockchain platform. In Hyperledger Fabric, in order to do any operation to the network, one needs to have a membership service provider (MSP). This is used to identify which participant or entity is allowed in the network and which is not. MSP is a set of cryptographic algorithms (certificates). The peers and orderers that are part of the system are issued certificates. This provides the first layer of security at the network level.
- The very notion of "channels" in Fabric provide privacy in that only a particular subnet of members can be part of the network; that is only only a private subnet of communication between specific network members (in our case the parties involved in the firearm tracking ecosystem). This provides the second layer of security at the channel level
- We observed that world state stores records in plain text format. In order to provide encryption to the data (key-value pairs), we use the BCCSP (Blockchain Cryptographic Service Provider) that will encrypt the data using AES256 algorithm. This provides the third layer of security at the chaincode level.

This ensures that the overall system is secure as such in pretty much every way we access it.

XII. Analysis and Future Scope

The implementation of Hyperledger fabric as a blockchain based approach into firearms tracking systems will definitely facilitate and help streamline existing gun tracking processes. It can also result in conducting more thorough background checks on people. Being a computerized approach, this will reduce the human error of entering data, checking, and verification. As shown in our prototype, the whole process has been automated with the help of chaincodes.Another advantage is that with this kind of system in place, in case any newer government regulations are imposed, all one has to do is add a new peer into the existing framework.

Hyperledger fabric helps to create and maintain a database that is accurate, resistant to hacking, and easy to access. The security aspects of this system has been already been explained in greater detail in the previous section and it goes to show that this implementation is quite secure.

Tracking the transfer of guns from the manufacturer to retailer, as well as reatiler to customer is done effectively and, in a time-efficient manner. This approach can help prevent illegal firearms from being forged in the market, as the firearms application would simply reject the firearm that has not been approved by the government, and would never allow the transaction to happen in the first place between a retailer and a manufacturer.

The NICS system currently used is a centralized, hub and spoke modelled database. A single organization maintains all inquiries that are made to the central database. The disadvantage with this kind of structure is that it has a low fault tolerance, and it also possesses a single point of attack for a hacker or malicious adversary. Thus, this can compromise data security. This drawback is taken care of in our Hyperledger based approach through the use of chaincode. Currently, due to the imperfect system in the NCIS, the FBI has recognized over 3000 people passing the background checks every year, despite being prohibited from buying guns. This can avoid by replacing the existing system with the Hyperledger, which would make the process 98 percent more efficient.

Our current version of the prototype has shown the capability and potential of Hyperledger Fabric by the implementation of two major use cases. We believe our application can be extended further to support a wide variety of use-cases in the future, such as providing live tracking and assistance during a crime investigation that involves these firearms. When such an incident occurs, it becomes way easier to track the weapon used to its owner. Even if the owner is not available or deceased, it can be tracked to the issuing retailer after which the police may be able to gather valuable information to conduct their investigation. The usage of channels, cryptogenic certificate, and chaincode have helped encrypt the data and improved the security of the system by a great margin and helped hide all the confidential data from malicious adversaries.

XIII. Conclusion

This report starts with describing the purpose of using Hyperledger Fabric for our application. We defined the components and the architecture of the Hyperledger Fabric and how those components would be beneficial in building our prototype. We defined and formulated the roles of the various actors for our prototype, the relationships between each other and the mode of interaction with other actors. Using this information, we implemented the underlying blockchain network using the binaries provided by Fabric. We also described the flows that we showed that have been implemented in our prototype and identified the scenarios that we plan on showing. We then described the chaincode functionality and the kind of smart contracts we used for developing our application with code snippets for a more concise and clear understanding of the methods used. We also showed the various features of our application and showed our application in action, supported by screenshots. Further the security aspects and how this is provably very secure was shown as well. We believe that this Hyperledger Fabric based solution for tracking of firearms is a feasible and viable one and we hope to take it to a larger scale in the near future.

Citations

[1] Hyperledger fabric: a distributed operating system for permissioned blockchains, Androulaki et al., Proceedings of the Thirteenth EuroSys Conference. ACM, 2018.

[2] Bauchner H, Rivara FP, Bonow RO, et al. (2017) Death by Gun Violence-A Public Health Crisis. JAMA. doi: 10.1001/jama.2017.16446

[3] Heston, Thomas. (2018). A Blockchain Solution to Gun Control. International Journal of Scientific Research.

[4] Pongnumkul, S.; Siripanpornchana, C.; Thajchayapong, S. Performance analysis of private blockchainplatforms in varying workloads. In Proceedings of the 26th International Conference on ComputerCommunication and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017; pp. 1–6.

[5] "Hyperledger blockchain technologies for business,"https://www.hyperledger.org/, accessed: 2017-03-10.

[6] Hu, Yining Liyanage, Madhusanka Manzoor, Ahsan Thilakarathna, Kanchana Jourjon, Guillaume Seneviratne, Aruna. (2019). Blockchain-based Smart Contracts - Applications and Challenges.

[7] Goldfeder S., Bonneau J., Gennaro R., Narayanan A. (2017) Escrow Protocols for Cryptocurrencies: How to Buy Physical Goods Using Bitcoin. In: Kiayias A. (eds) Financial Cryptography and Data Security. FC 2017. Lecture Notes in Computer Science, vol 10322. Springer, Cham

[8] Haseeb et al. "A fabric architecture towards block chain application using hyper ledger." (2018).

[9] Ma, Chaoqun Kong, Xiaolin Lan, Qiujun Zhongding, Zhou. (2019). The privacy protection mechanism of Hyperledger Fabric and its application in supply chain finance.

[10] Cachin, Christian. "Architecture of the Hyperledger Blockchain Fabric." (2016)

[11] F. Benhamouda, S. Halevi and T. Halevi, "Supporting Private Data on Hyperledger Fabric with Secure Multiparty Computation," 2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, 2018, pp. 357-363.

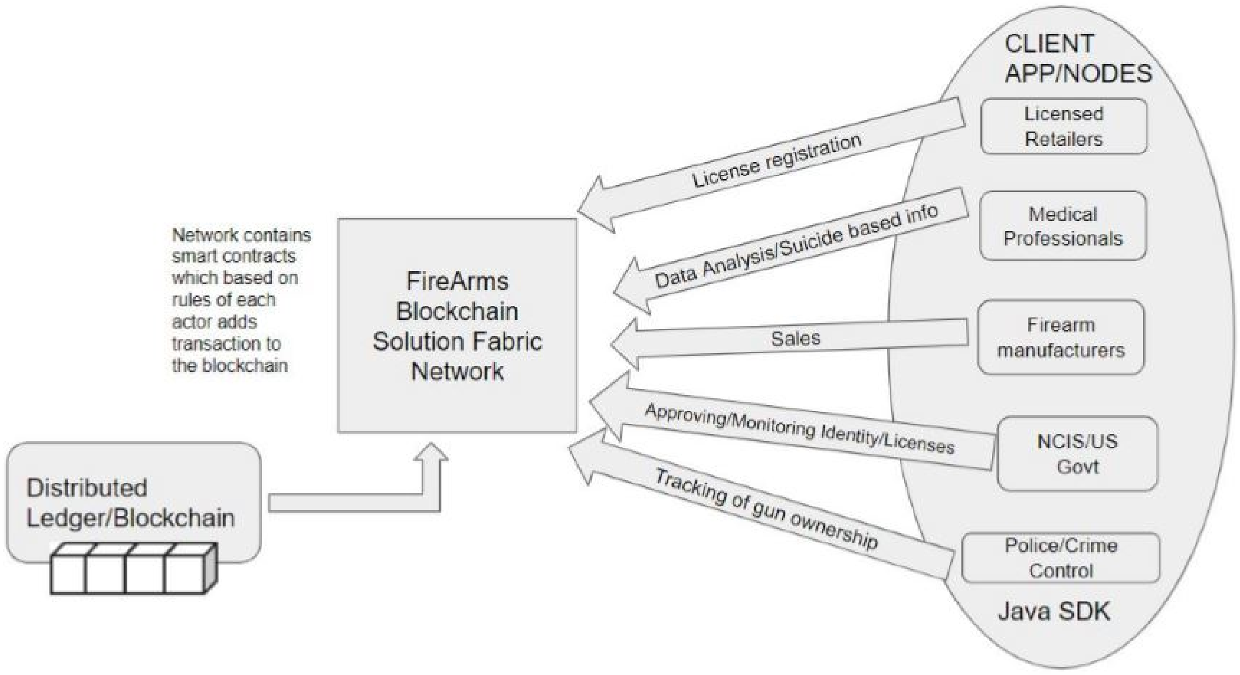
[12] Beyond Theory: Getting Practical With Blockchain, Federal Reserve Bank of Boston" (2019)

[13] Kariuki, D. (2018, February 24). Blockchain Technology Can Improve Gun Control (2019).

[14] Price, D. Blockchain Problems: Security, Privacy, Legal, Regulatory, and Ethical Issues (2019)

Appendix (Terms for reference):

- High level flow diagram of implementation:


CLIENT APP/NODES
Licensed Retailers
Medical Professionals
Firearm manufacturers
NCIS/US Govt
Police/Crime Control
Java SDK
License registration
Data Analysis/Suicide based info
Sales
Approving/Monitoring Identity/Licenses
Tracking of gun ownership
FireArms Blockchain Solution Fabric Network
Network contains smart contracts which based on rules of each actor adds transaction to the blockchain
Distributed Ledger/Blockchain

- MSP: Membership Service Provider
- CA: Certificate Authority
- PKI: Public Key Infrastructure
- AES 256: Advanced Encryption Standard
- Consortium: Group of orgs that transact
- NICS: National Instant Criminal Background Check System
- SDK: Software Development Kit
- facc: fire arms contract code (smart contract)