UNIVERSITY OF THE AEGEAN

SCHOOL OF ENGINEERING

DEPARTMENT OF INFORMATION AND COMMUNICATION SYSTEMS ENGINEERING

POSTGRADUATE PROGRAM

INFORMATION AND COMMUNICATION SYSTEMS SECURITY

Blockchain-Based Decentralized Public Key Infrastructure (PKI) using Smart Contracts

DIPLOMA THESIS

of

Vasileios Zagoras

Professor : Dr. Panagiotis Rizomiliotis

Members of the examination committee:
Dr. Panagiotis Rizomiliotis, Dr. Maria Karida, Dr. Spiros Kokolakis

Samos, July 2021

This page is intentionally left blank.

To everyone and everything I have or have not met.

Z.

Thanks.

Here is to my teacher and supervisor Doctor Panagiotis Rizomiliotis, all of the other teachers, my fellow students and friends and everyone that lies behind the scenes and makes this department run.
In my time of studying here I have acquired tangible and fertile knowledge and for that I pledge my respect to the whole department by heart.
You have educated me pretty well both technically but also mentally so thank you!
Special thanks to the Aegean University Teachers Association for all the fights that they give every day so as to protect our University rights under a foul regime that only aims at privatization of our rightful public education by trying to downgrade everything.

I also pledge my respect to you Mother and Father for bringing me to life and raising me up by giving me whatever you could.
To you Sister for raising and rising up together with me.
To you my wider family by blood and not for assisting in me being what I am today.
To Gong Fu and my teachers for reminding and helping me stand up and fight.

I hope that I am worthy of all the gain and that by everything I do, I can make all of you smile proudly!
Thank you all for being parts of my life and myself!
All of my love to you!

Z.

This page is intentionally left blank.

Contents.

List of figures

List of tables

Acronyms

PKI	Public Key Infrastructure
DPKI	Decentralized Key Infrastructure
IoT	Internet of Things
HLF	Hyperledger Fabric
P2P	Peer-to-Peer
PoW	Proof of Work
PoS	Proof of Stake
DPoS	Delegated Proof of Stake
PBFT	Practical Byzantine Fault Tolerance
CA	Certification Authority
RA	Registration Authority
CDS	Certificate Distribution System
CMS	Certificate Management System
DLT	Distributed Ledger Technology
CFT	Crash Fault Tolerant
BFT	Byzantine Fault Tolerant
Chaincode	Smart Contract
CLI	Command Line Interface

Abstract

Public Key Infrastructure (PKI) is a set of roles, policies, hardware, software and procedures that can be used to create, manage, distribute, store and revoke digital certificates and manage public-key encryption. It's purpose is to facilitate the secure electronic transfer of information between entities inside a network. Where simple passwords are inadequate, for example a banking application, it is required to confirm the identity of the parties involved in the communication and to validate the information being transferred. There comes PKI to give the solution by providing the guarantees that an entity is the one it claims it is. PKI services are provided via third parties and this makes them central points of failure. So there is a need for alternate similar systems so as to avoid third parties while keeping the main PKI functionality. Decentralization is the key point here and blockchain technology seems that can help decentralize PKI. The emerging technology coming out of this union is called Decentralized Public Key Infrastructure and offers a great alternative to the classic PKI solution. With DPKI any user of the system can sign each other's public key certificate and keys are by design intended to have multiple signatures. If one signer is compromised and the signer's key is revoked, the impact on the trust network is limited. Decentralization of PKI enhances security, integrity, scalability and availability of the classic implementation, as there is no reliance on a central authority. Instead, a blockchain is deployed and utilized as the record-keeping of device identities, with smart contracts to interface with the blockchain network. A decentralized PKI solves the problem of trusting the authorities in the key technologies behind the Internet. The main idea for DPKIs is the ability to establish trust relations between all parties, in a web-of-trust model, avoiding centralized authorities and related root-of-trust certificates and are designed to address a fully decentralized ledger for managed certificates, providing data-replication with strong consistency guarantees, and fairly distributed trust management properties founded on a P2P trust model.

1 Introduction

Problem Statement

Public Key Infrastructure (PKI) establishes a relation between a user and a respective public key. Public Key Infrastructures (PKIs), which rely on digital signature technology to establish trust in a communication channel, allow entities to interoperate with authentication via X.509v3 digital certificates. Despite PKI technology being used by many applications for their security foundations (e.g. WEB/HTTPS/TLS, Cloud-Enabled Services, LANs/WLANs Security, VPNs, IP-Security), there are several concerns regarding the use of a centralized authority to handle all PKI's functionality. Nowadays there exist third party services that control identities online. This situation can result in security challenges Internet-wide. Not to mention that as the technology of Internet-of-Things (IoT) advances, the potential benefit of compromising such networks increases. Conventional security stacks like that of a classic PKI fails to address certain issues that are prevalent in these networks, as it lacks of a secure methodology of verifying the identities of devices without a central point of failure.

Objective

To avoid the aforementioned centralization drawbacks, a Decentralized Public Key Infrastructure (DPKI) can be the secure alternative. Decentralization of PKI enhances security, integrity, scalability and availability of the classic implementation, as there is no reliance on a central authority to do everything. Instead, a blockchain is deployed and utilized as the record-keeping (integrity through redundancy) of device identities, with smart contracts to interface with the blockchain network. A decentralized PKI solves the problem of trusting the authorities in the key technologies behind the Internet. The main idea for DPKIs is the ability to establish trust relations between all parties, in a web-of-trust model, avoiding centralized authorities and related root-of-trust certificates and are designed to address a fully decentralized ledger for managed certificates, providing data-replication with strong consistency guarantees, and fairly distributed trust management properties founded on a P2P trust model. Classic PKI functions are supported cooperatively, with validity agreement based on consistency criteria, for issuing, verification and revocation of X509v3 certificates. In our proposed DPKI framework model, X509v3 certificates are issued and managed following security invariants, processing rules, managing trust assumptions and establishing consistency metrics, defined and executed in a decentralized way by the Blockchain nodes, using Smart Contracts. So in this dissertation is about the implementation of a decentralized Public Key Infrastructure system using a Smart Contract and it is proposed so as to solve key-bootstrapping in IoT networks. [1 - 5].

2 Blockchain

Introduction

A blockchain is a nothing more than an ordered and continuously growing back-linked list of blocks of transactions data structure represented as a merkle-tree [18-19]. The back-linking property is achieved through the cryptographic SHA-256 hash of the previous block each block contains in it's block header. So we can say that each block references the previous one through it's hash. Also, each block carries transaction data in it's main body and after the block is filled then we say that it is "mined" and another one gets to be created and filled up with new transactions. So that is how this list of blocks gets to grow. And since these blocks are "tied" together by their cryptographic hashes, we can say that they form a chain and since it's parts are called blocks it is a chain of blocks aka blockchain.

Blockchain is usually called a public distributed ledger. That is because, the blockchain is managed by a peer-to-peer network, where all nodes adhere to a communication protocol and based on it, they validate new blocks. Blockchains can be considered secure in a way that they offer a distributed computing system with a high Byzantine fault tolerance, meaning that at least the two thirds of the system is functioning properly [23]. The blockchain can be stored as a plain file but also in a database. For example bitcoin's blockchain is stored in a levelDB. If we flip the blockchain vertically, we can see that it has a "height" marking the distance from the first block, a "top" to refer to the latest block added. So, each block is identified by it's hash and contains a hash of the previous one (parent). But what about the first block's parent? Well, the first block does not have a parent and it is called the genesis block.

Now, in blockchain, branching is a very common phenomenon. That means that a block can lead to more then one blocks, or have multiple children. That happens when the blockchain is "forked", meaning that more than one blocks were "mined" simultaneously. This is a temporary situation though as eventually only one block stays in the chain. As it is understood a reverse situation where a block has multiple parents cannot happen as each block contains a hash of the previous one and cannot contain multiple hashes aka multiple parenting situation does not exist.

Another great property of a blockchain is that it cannot be tampered. If let's say an intermediate block is tampered then the next block would immediately contain an invalid pointer of the previous one so from there and on, the blockchain gets to be invalid and so it cannot be changed.

History

1982

Computer scientist and Cryptographer David L. Chaum is the “thread of Ariadne” aka the beginning of everything regarding blockchain technology, as he first proposed a blockchain-like protocol in his dissertaion “ Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups ”. There, he visions: “a number of organizations who do not trust one another can build and maintain a highly-secured computer system that they can all trust (if they can agree on a workable design). ” as he mentions [24].

1991

Stuart Haber and physicist W. Scott Stornetta use cryptographic techniques to provide digital timestamping and to secure digital documents from data tampering. They produced a primitive immutable ledger. That is, an autonomous system of blocks cryptographically chained together. In other words they did create a primitive kind of blockchain technology [26, 30].

1992

S. Haber and W.S. Stornetta upgrade their system proposed in 1991 so as to make use of Merkle trees in order to be able to include whole document collections in just a single block [27, 30].

1996

Computer scientist Nick Szabo proposes “bit gold”, a decentralised digital currency which has an “unforgeable chain of digital signatures” that dictates each user’s bit gold holdings and transactions. He also intoduces for the fist time in computer science the consept of smart contracts and their uses [30].

1997

D. Bayer, S. Haber and W.S. Stornetta in [28] describe for the first time a chain of hashed blocks of data. In their paper back then they mention: “a method that provides a cryptographically verifiable label for any bit-string (for example, the content, in a particular format, of the document)” and also “naming a bit-string according to its position in a growing, directed acyclic graph of one-way hash values”.

1998

Computer engineer W. Dai, a specialist in cryptocurrencies and cryptography, proposes a decentalized money idea known as B-Money. Users can remain anonymous using digital pseudonyms and perform transactions with each other. Dai described a database protocol that would have a complete account of each pseudonym's money, enabling everyone on the network to know the money's ownership collectively [30].

2000

Stefan Konst publishes his theory of cryptographic secured chains, plus ideas for implementation.

2000

Satoshi Nakamoto releases a white paper which establishes the model for a blockchain structure and architecture.

2009

Nakamoto implements the first blockchain as the public ledger for transactions made using bitcoin.

2014

Blockchain technology is separated from the currency and its potential for other financial, interorganisational transactions is explored. Blockchain 2.0 is born, referring to applications beyond currency. The bitcoin blockchain file size, containing records of all transactions that have occurred on the network, reached 20 GB (gigabytes).

2015

Under the guidance of developer Vitalik Buterin, Ethereum public blockchain is lunched, having more functionality when compared to Bitcoin. The most innovative one is that Etherium introduces computer programs into the blocks to be used as financial bonds (aka smart contracts). At the same year, the Linux Foundation unveiled the open-source project Hyperledger, which is a collaborative development of distributed ledgers. Hyperledger seeks to advance cross-industry collaboration for the development of blockchain and distributed ledgers. Hyperledger is the first to initiate the concept of a private blockchain. It focuses on encouraging the use of blockchain technology to improve the performance and reliability of current systems to support global business transactions. Hyperledger has been mostly used by companies that want to run their business with the help of a distributed ledger, that is also private, in a way that no one can get inside withought the company’s permission, thus giving the company an insurance that sensitive data such us client data or any other sensitive information won’t go out in public [33].

2017

EOSIO is a leading open-source platform that is about blockchain innovation and performance. It is used from businesses to developers around the world so as to create secure, transparent, and deterministic digital infrastructures. It is one of the most performant, scalable, configurable and secure blockchain platforms. It offers great solutions regarding the development of blockchain applications and has a wide range of capabilities. It offers great tools to develop private, public and permissioned blockchains [34].

Block Structure

As we briefly mentioned earlier, a blockchain is a decentralized, distributed, and often public, digital ledger that consists of blocks that are used to record transactions. A block is a data structure that contains mostly transaction data. The block follows a certain structure and that is made of : a 4 bytes field called block size that dictates the size of each block, an 80 bytes header that contains all the block's metadata. That is, the previous block's hash, root's transaction hash (called merkle root), a nonce value (we will talk about that later on), a timestamp and more. Also, there is a transactions counter filed if 1-9 bytes of size, that is no more than an integer that contains the number of transactions that current block contains. Last but not least, the block contains a list of transactions that make up most of its size. Now, the average transaction is about 250 bytes and the average block contains more than 500 transactions. That make an average block to be more than 125000 bytes in size [20].

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields form the block header (see below)
1-9 bytes (VarInt)	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

Figure 1: The structure of a block.

Block Header

The block header contains the most important information about a block [35]. Following there is the description of it's parts:

Version: a number to track which software version the miner of the block used and so which set of block validation rules were followed.

Previous block hash: A field that contains the previous block's header hash and so it establishes an order throughout the chain of blocks and ensures that the blockchain cannot be tampered.

Merkle Root Hash: the last hash of current block's transactions, as the root of the merkle tree of the block's transactions. It is an aggregation of all previous transaction hashes. We will dig more into how the transactions are stored in merkle trees later on.

Timestamp: the Unix epoch time as the creation time of this block, aka the time that the miner started hashing the header.

Bits or nBits: represent the current difficulty of finding a new block.

Nonce - or number used once: the variable that miners change to modify the block headers hash for its value to meet the difficulty. We will analyze that later on in the mining chapter.

Size	Field
4 bytes	Version
32 bytes	Previous Block Hash
32 bytes	Merkle Root
4 bytes	Timestamp
4 bytes	Difficulty Target
4 bytes	Nonce

Figure 2: Block header structure [20]

Block Identifiers

The identifier of a block is its cryptographic hash, made by hashing the block header [20]. This is called block header hash and it is 32 bytes. The block hash is it's unique identifier and derived by any node by simply hashing the block's header. Block hash is not included inside each block's data structure, instead, the it is computed by each node as the block is received from the network. It could be saved in a seperate database though so that someone could easily navigate through the blockchain's blocks. Another identifier of a blocks position is the height, meaning the distance between the genesis block (zero height) and the i-th block (i height). Of course, height is not a unique identifier like the block's hash, as two blocks can have the same height, at least temporary. That is, when two blocks are mined simultaneously, as we discussed in the introduction and as we will analyze more later on.

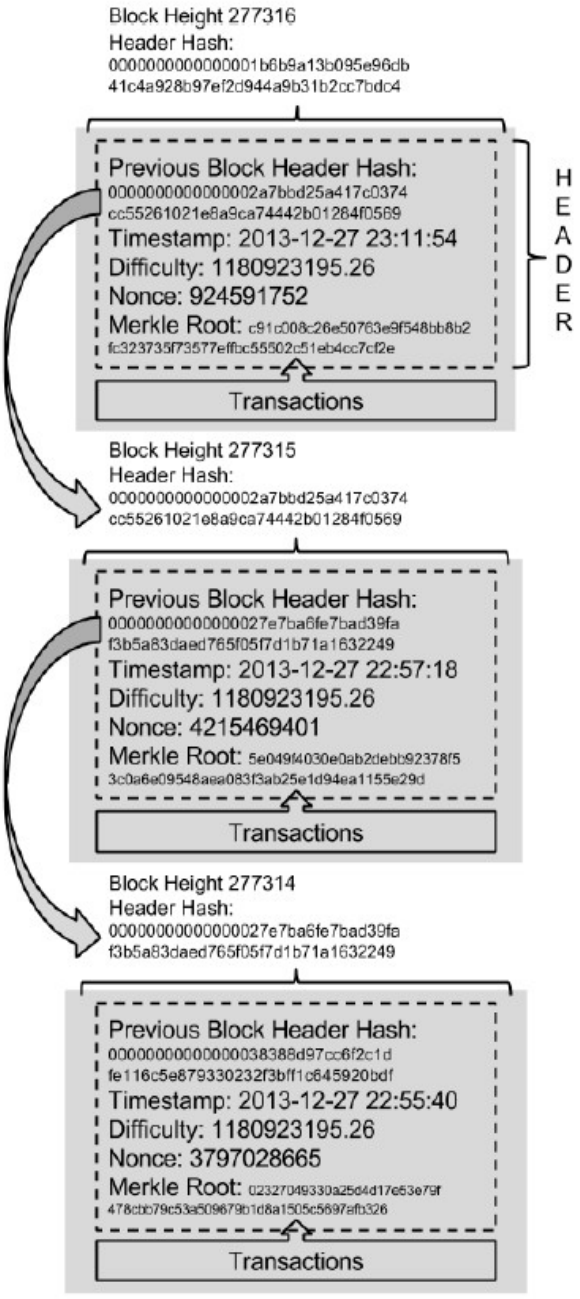


Figure 3: Blockchain in action [20].

Genesis

Like everything in this world, from a tree, to a human, to an idea or to a whole universe, they all start with a genesis. Of course blockchain could not have escaped of that rule. So, blockchain's genesis is nothing more that a block, called the genesis block. From that time and after, that block is the "mother" of all preceding blocks in the blockchain, in a way that if we follow the line of blocks by their previous block hashes, we will always end up arriving at the genesis block. It is initially created from within each blockchain's code and represents a secure "root" whose details each block "knows" and can always use as a starting point reference. Regarding bitcoin, Satoshi Nakamoto embedded a hidden text inside bitcoin's genesis block that said: "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks". This message provides proof of the earliest date this block was created, as it is referencing the headline of the British newspaper: The Times [20].

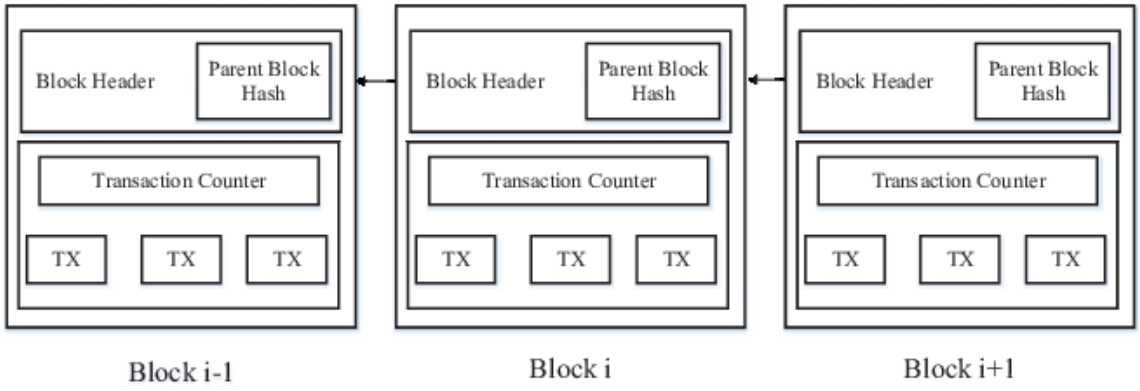


Figure 4: An intermediary blockchain part [36]

Consensus & Mining

Consensus is all about a common ruleset followed respectively by all blockchain nodes. In a decentralized system where all nodes are equal and there are no authorities present, achieving consensus is a difficult thing. So, the majority of nodes must agree on what is true, reaching consensus at regular intervals. As it is expected, some nodes will fail, misbehave or disagree with the consensus of the other nodes so the system must be designed in such a way that deals with this [49]. Consensus protocols currently used in most blockchain systems can be divided into two categories:

- **Probabilistic-finality:** Guarantee a consistency eventually

- **Absolute-finality:** Guarantee a common state between nodes and that they take consistent actions in each round of consensus

Property	PoW	PoS	DPoS	PBFT	Ripple
Type	Probabilistic-finality	Probabilistic-finality	Probabilistic-finality	Absolute-finality	Absolute-finality
Fault tolerance	50%	50%	50%	33%	20%
Power consumption	Large	Less	Less	Negligible	Negligible
Scalability	Good	Good	Good	Bad	Good
Application	Public	Public	Public	Permissioned	Permissioned

Figure 5: Consensus protocols comparison [51].

PoW (Proof of Work)

PoW is a well known and widely used consensus protocol. It selects one node to create a new block in each round of consensus based on the computational power competition [51]. In the competition, the participating nodes gather so as to solve a cryptographic puzzle. The first one to solve it can start creating a new block. A PoW puzzle is nothing more than finding a nonce for which the hash of the block has a certain number of leading zeros. This can only be done via trial and error, so there is the difficulty of the puzzle and that is why it is called proof of work. This process is also called mining. So, in PoW protocol a malicious attacker can overthrow one block in a chain, but since eventually the longest chain wins, the attacker has to have enormous computational power. PoW is adopted by Bitcoin, Ethereum etc and belongs to the probabilistic-finality consensus protocols since it guarantees eventual consistency.

PBFT (Practical Byzantine Fault Tolerance)

PBFT is a Byzantine Fault Tolerance protocol with low algorithmic complexity and is highly practical in distributed systems [51]. There are five phases in PBFT: request, pre-prepare, prepare, commit, reply. The primary node forwards the message sent by the client to a number of nodes. These nodes reply to the client to complete a round of consensus. PBFT guarantees that nodes have a common state. PBFT achieves the goal of strong consistency, so it is an absolute-finality consensus protocol.

Public versus Private blockchains

Public blockchains are open to the public and anyone can join without specific permission. All people who join the network can read, write, and participate in this network that is not controlled by anyone [50]. Public blockchains are immutable and decentralized and no one can change an entry once it has been validated. Bitcoin, Litecoin, Ethereum use public blockchains and get the most attention. But they can be also used by governments for a voting platform or keeping the healthcare records. In these platforms anonymity and transparency are the key features. Public blockchain is more a Business to Consumer solution.

Private blockchains are invitation-only and so permission from the governing body is needed so as to get access. They have different levels of access that define who can write, read or audit the blockchain. Private blockchains do not offer same levels of decentralized security as the public ones. They also require each user to have a verified identity that defines the type of access. Transactions are faster and more energy-efficient to maintain. In other words, private blockchains are best for business to business implementations.

Simultaneous Blocks

Branching is a common phenomenon in blockchain. That happens when a block leads to more than one paths. We then say that the blockchain is “forked” and we mean that more that one blocks were “mined” simultaneously. However, this is a temporary situation at last only one block will stay. The question here would be what happens with the rest of the blocks and which miner gets to have the PoW reward [22].

The thing is that although more than one miners have mined legitimate blocks, they will be different as each miner would have chosen different transactions to add to the block. So each miner would solve it’s own block with the nonce found by Proof of Work so as to pass the difficulty threshold set.

So we have two legitimate blocks waiting to be added to the chain. The decision on which block will be chosen depends on all of the rest miners in the network. And how will this happen? By internal announcements of the mined blocks of course. Both blocks will be spread over the network and the first one received by the miners will be added to their copy of the blockchain.

This means that for a short time, there will be two different, yet valid versions of the blockchain and so the validity of each blockchain will be diverse amongst miners. Each version of the blockchain will keep on growing differently for each mining circles. Now, depending on the mining speed one chain will grow larger than the other and by the time the longest chain is spread across the network, miners who had kept the smaller one will have to discard the one they were working on.

At that point, transactions that were in the discarded blocks will be dropped and if they were not included in the new blocks they will be dropped for good.

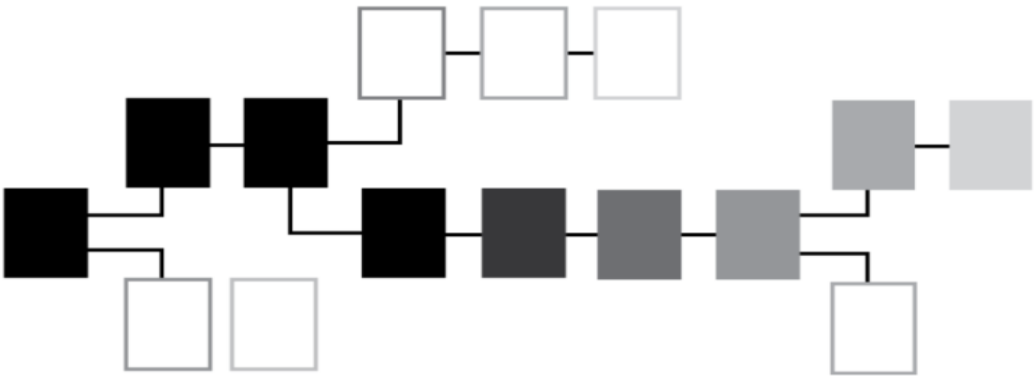


Figure 6: A blockchain fork.

Merkle Trees

As we discussed earlier, blocks in a blockchain contain transactions that are hashed and encoded into a Merkle Tree [18]. A Merkle tree or a binary hash tree, is a tree is "a data structure used for efficiently summarizing and verifying the integrity of large sets of data" [20]. Every node of such a tree contains the hash of it's children, in a way that: assuming that the leafs contain the data, the first-level nodes contain the hash of each leaf, the next-level ones the hash of the two previous nodes etc. Merkle trees are used in blockchain to summarize all block transactions and also provide a very efficient process to verify if a transaction is included in a block. Based on the aforementioned basic property, we can see if a transaction belongs to a block just by performing successive hashes from the transaction's position in some leaf up to the root. If a hash does not match the one that the block states, then the transaction is invalid and does not belong to the block. Now for a set of N transactions, the complexity of that will be $\log_2(N)$ aka O(log(N)) [37]. The concept of hash trees is named after Ralph Merkle, who patented it in 1979. A graphic representation of a Merkle Tree is shown bellow [19]:

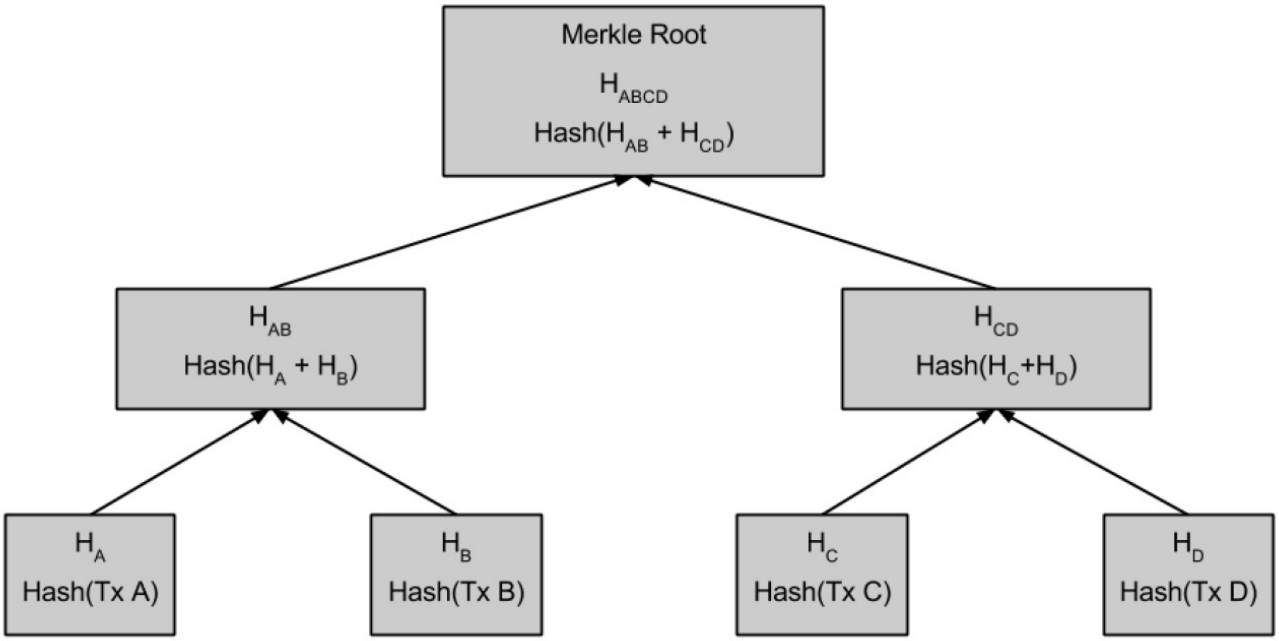


Figure 7: A representation of an encoded blockchain into a Merkle tree.

In the example above, we have transactions A, B, C and D whose hash values are stored as leaves of the Merkle Tree. To construct the root node HABCD, we first need to perform Hash(HA + HB) and Hash(HC + HD), store the values in the intermediary nodes, and then perform Hash(HAB + HCD) and save this value as the Merkle root.

3 Public Key Infrastructure

Introduction

The problem: Public key cryptography on its own is not enough to ensure the security of online transactions. Organizations need a framework that provides policies to generate and distribute keys [38].

Public Key Infrastructure (PKI) is a framework that solves the above. PKI contains security policies, encryption mechanisms and applications that generate, store, distribute, utilize and manage keys and certificates. It also describes the policies, standards and software that are used to regulate certificates, public keys, and private keys.

But what Is PKI?

A public key infrastructure (PKI) is a set of roles, policies, hardware, software and procedures that can be used to create, manage, distribute, store and revoke digital certificates and manage public-key encryption [40]. It's purpose is to facilitate the secure electronic transfer of information between entities inside a network. Where simple passwords are inadequate, for example a banking application, it is required to confirm the identity of the parties involved in the communication and to validate the information being transferred. There comes PKI to give the solution by providing the guarantees that an entity is the one it claims it is. In cryptography, PKI is an arrangement that binds public keys with respective identities of entities. This is achieved via certificates that are signed by a trusted authority, from now on CA. Communication is based in trust. In physical communication, face identification works as the source of truth. However, in case of electronic communication, this trust is quite difficult as the identity of the other entity remains concealed. Here, trust cannot be established unless both entities are sure about each others identities and that the information they are exchanging over a network is completely secure. When someone for example conducts a transaction over the Internet, is not sure about the legitimacy of the company, the product or identity of the owner. PKI addresses these underlying problems of trust, authentication and security in general over the network.

Setting up and running a PKI in the real world is not that simple. Issues such as identifying users for certificate issuing and trusted distribution of CA keys arise and have to be solved. There are also many more others and among the most complex are the existence of many different CAs and revocation of certificates [39].

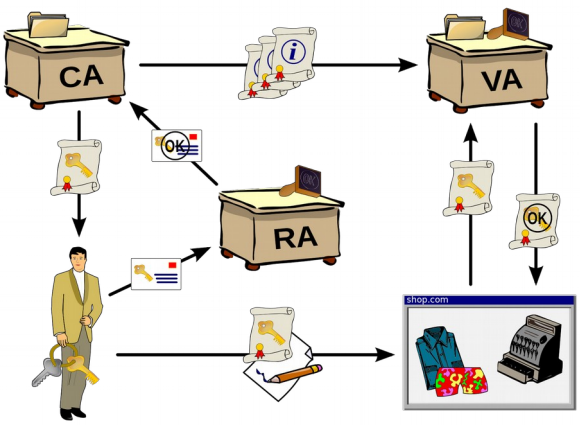


Figure 8: A PKI topology as a use case [41].

PKI Components

As we mentioned before PKI is a combination of hardware/software components, policies and different procedures. It's structural objects are called certificates and work as digital identifications. Certificates associate users to their public keys [42].

PKI consists of the following components:

■ **Certification Authority (CA):** PKI's trust basis. CA is the creator of certificates.

■ **Registration Authority (RA):** An interface between user and CA. Authenticates each user's identity and forwards requests for certificate issuing to the CA.

■ **Digital certificates:** Digital entities that associate users to their public keys.

■ **Certificate distribution/management system (CDS/CMS):** Distribution can be performed by users or a directory server such as LDAP. That is something that depends on the PKI environment.

■ **Security Policy:** Defines the level of security, the processes and the use of cryptography. It also includes information on how to handle keys and valuable information.

■ **PKI applications:** Every communication between server and browser.

Each PKI component mentioned above has a specific role to perform. Bellow we analyze further about each component.

Certification Authority

A certificate authority (CA) or certification authority, is an organization whose purpose is to issue, revoke and validate the identities of entities. These, can be websites, email addresses, companies, individuals. In other words, CA is a trusted third party that authenticates entities. To authenticate an entity, the CA issues a digital certificate. This is a digital document that binds the credentials of the entities. The certificates issued, contain information such as the name of the subscriber, the cryptographic keys of the subscriber and the CA's public key. Note that depending on the policy of the issuing organization this information changes. Before issuing a digital certificate, the CA ought to verify the certificate request with a Registration Authority (RA). There, other organization rules apply. If the rules are met, then the certificate gets to be issued [38]. Now, during the authentication process, these identities are matched with their corresponding cryptographic keys through their digital certificates [43].

Registration Authority

A Registration Authority (RA) is responsible for receiving certificate signing or revoking requests and forwarding them to the CA for account of entities that need certifications [44]. A Registration Authority as a component entity, is usually separated from the Certificate Authority for accessibility and security reasons as the two components should communicate in order to process each request separately. Usually an RA is accessed via a user interface or an API. RAs are useful for scaling PKI applications, as a CA can delegate its responsibilities to multiple RAs and assign an area of operation to each RA. Last but not least, RA is used as a first filter as apart from it's other duties it performs all needed checks on the entity requesting the certificate so as to confirm their identity [45].

Digital Certificates

The security of a public key is important so as to avoid security breaches like impersonation and (key) tampering [38]. Here comes integrity. Integrity is needed so as tampered keys are detected and rejected immediately. Integrity mechanisms though are not enough so that it is guaranteed that the public key belongs to the claimed owner. So, somehow the identity of a public key should be validated and matched to an identity. There is the role of third party services, who at first ensure that the public key is valid and also they bind it some user's identity. Thus, the third party service that holds the user certificates, works as a middleware in each communication that shall be secured and can ensure that only the public key for a certificate authenticated by a CA matches with the private key possessed by the entity. This makes impersonation attacks useless in such channels. A digital certificate contains a serial number, the digital signature of the CA, the owner's public key, an expiration date, the CA name.

It provides:

- Authentication as it matches the identity with the public key.
- Encryption via the use of the user's public key.
- Integrity of the certificate-signed documents.

An entity can use a digital certificate so as to make communications as follows:

- Send digitally signed (with it's private key) messages so that integrity and authenticity are achieved.
- Digital signature verification capability through the use of the entity's public key. This is achieved by using the CA's public key, which lies inside the entity's digital certificate.
- There lies a global directory database which verifies if a digital certificate is valid or not. If not, then the action does not get to the next step so as to be executed.

Note that after a certificate has been issued, users and organizations get notified by a central mechanism called Certificate Distribution System.

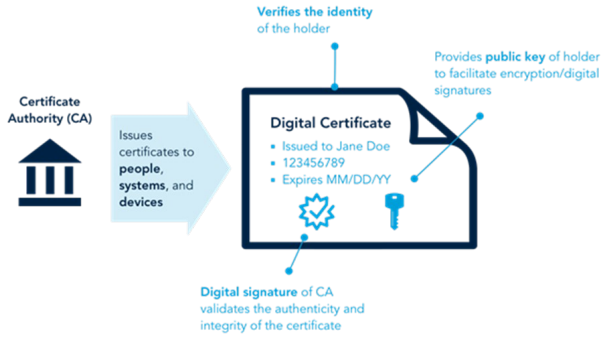


Figure 9: A digital certificate – visually [46].

Classes of Certificates

Digital certificates are divided into four classes [45]:

Class 1: Obtained by supplying an email address.

Class 2: Additional personal information needed so as to be supplied.

Class 3: Purchased after checks have been made about the requestor's identity.

Class 4: Used mostly by governments and financial organizations and need high level guarantees.

Certificate Management/Distribution System (CMS/CDS)

CMS is responsible for the publishing, suspending, renewing and revoking of certificates [45]. It distributes certificates to users and organizations. Depending on implementation of PKI in the organization, there are to ways to distribute certificates. By users

themselves or they can be distributed by a directory server that uses LDAP so as to query the user information stored in an X.500 compliant database [38]. The distribution system is used for the following tasks:

- Generate and issue key pairs
- Validate public keys
- Revoke expired/lost keys
- Publish public keys in the directory service server

Policies

PKI policies are set of rules that are configured by each organization. These, are mainly security principles that each organization needs to follow. Like how the organization manages public and private keys, the access rights of each user role etc [38].

PKI Clients

”The entities which request CAs or RAs to issue certificates are commonly referred to as PKI Clients” [38]. A PKI client in order to get a certificate from a CA first needs to perform these actions:

- Request a key pair from the CA.
- Request a CA certificate from the RA.
- Client can use the certificate so as to get authenticated.

Client – CA communications are encrypted. As it is understood the private key is kept at the client side, so the client needs to keep it safe and secure. A private key’s safety can be ensured by the use of hardware components like smart cards.

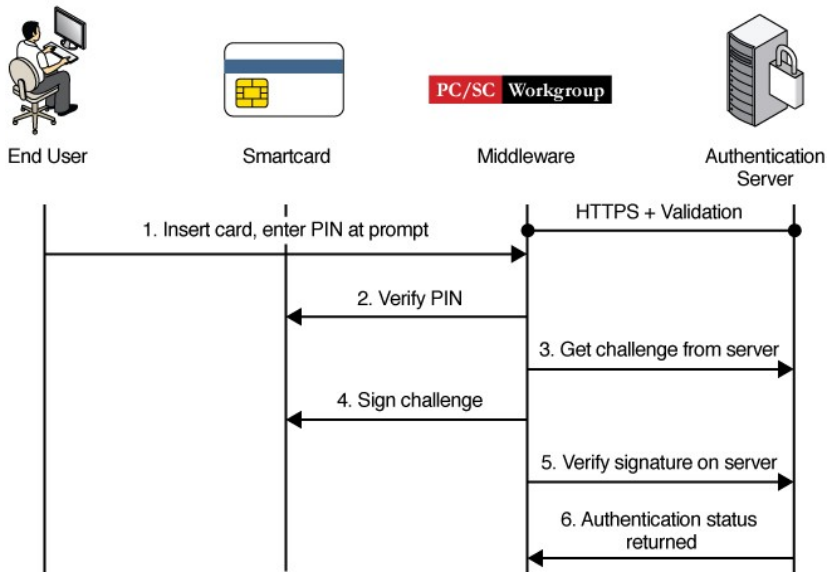


Figure 10: PKI authentication scheme by using smart cards [47].

Decentralized Public Key Infrastructure (DPKI)

As we discussed before, third-parties are central points of failure, each capable of compromising the integrity and security of the entire Internet. That requires designing a decentralized infrastructure where every identity is controlled not by a trusted third-party, but by its principal owner [3]. The main way to secure online communications is from the usage of public keys. Keys are used to verify the users identities. The entity these identities represent, called the principal, uses a corresponding secret private key to both decrypt messages sent to them, and to prove the authenticity of their sent messages. Traditionally, PKI systems are based on third party services for the secure delivery of public keys. So here comes DPKI to remove these third-parties that controls the communications. Production and the management of the keys that are necessary for the procedure is to complex to be left in the hands of users that don't have encryption skills, so the companies do this themselves. This means two things: Firstly that all keys are gathered in a central point of failure and above that, anyone with access to these servers can compromise their security and stay undetectable. As a result of conventional PKI's usability challenges, much of Web traffic today is unsigned and unencrypted. This is particularly evident on social networks. Because of PKI's complexity, social networks do not encrypt their user's communications in any way, other than relying on PKIX by sending them over HTTPS. So messages are not signed aka there is no way to be sure that a user really said what they said, or whether the text displayed is the result of a database compromise. This is where the DPKI systems come to introduce themselves as a solution to the problems above. The decentralized public key infrastructure uses technologies from which it is possible for disparate entities to reach consensus on the state of a shared database via decentralized key-value datastores which are called blockchains. In these systems validators still exist but their role is limited and mainly their role is to ensure for the integrity of the blockchain as a whole system and not for each transaction as a disparate entity.With DPKI any user of the system can sign each other's public key certificate and keys are by design intended to have multiple signatures. If one signer is compromised and the signer's key is revoked, the impact on the trust network is limited.

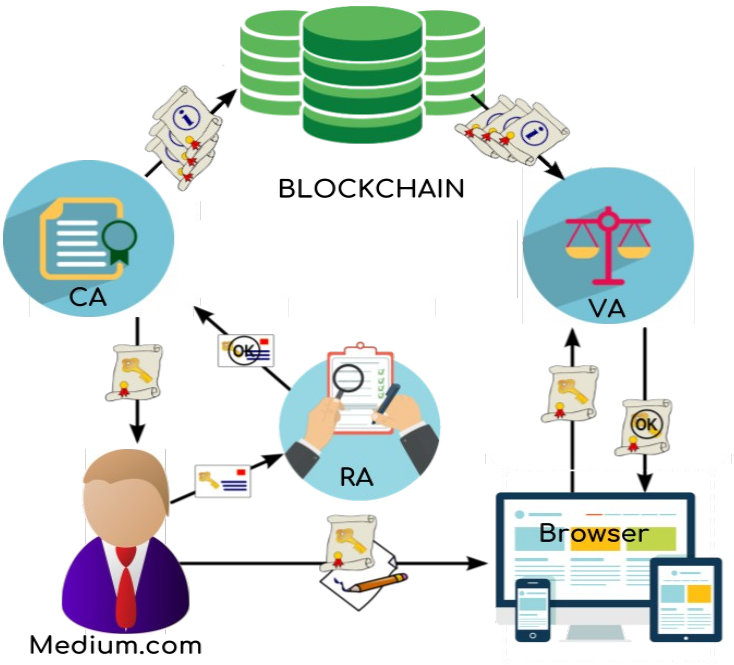


Figure 11: Decentralized Public Key Infrastructure scheme [52].

DPKI Core

The core of DPKI are decentralized key-value datastores that can serve as identifier registries, allowing a principal's public keys to become securely associated with their identifier. As long as registration remains valid and the principal is able to maintain control of it's private key, no third-party can take ownership of that identifier. A DPKI system must have some standard properties, such as the right to give permissionless writes. That means that the system can sign a confirmation form without the permission of a specific party. DPKI registration is different from that of DNS. Private keys must be generated in a decentralized manner so that principal control is ensured. Obviously private keys must never be stored or transmitted in an insecure way. So, whenever a user asks permission to contact with another user the system asks for a digital signing of the user's request. Then it uses the public key in order to confirm if that the private key was used without knowing the private key. Then everyone in the blockchain system will know that the user was approved by someone in the system without giving away his private key.

Registration

User's private key is used by the user to register and update the data which are associated with them and the public key for the communication process. The former is saved locally on user's machine, whereas public key can be stored directly or indirectly in the datastore. When a public key is stored indirectly on the DPKI system that means that a pointer is stored that contains the finger print of the public key. This is done so as to gain a noticable space in the datastore.

Security

In DPKI, the most common things that can go wrong are two. Either an outdated value for a lookup key is sent so as to identify a user or if the use case that someone sends wrong information in attempt to take control of the chain itself by overpowering the "honest" nodes of the system. However this problem is not a users problem because it is based in a larger security concern which the members of the system will have to solve in order to keep the control of the chain. The last one is the most heavy-waited. However, this can be surpassed with more disentralization. As much more decentralized a chain is, so much more secure it is. A user can understand the level of decentralization of a blockchain system by counting the number of parties that control the behavior of the chain which are called "Devs" and the number of the blockchain replicas that exist in the system. Every node in the blockchain has a replica of the current situation of the systems chain. Also there is another group, the validators which are the miners that create the new blocks. So we can understand that the more Nodes and Devs and Validators we have the more decentralized a system is, the more secure. Another concerns is the secure access of the blockchain data. Here come the thin-client protocols which download smaller portions of the blockchain, sufficient to provide security guarantees stronger than those provided by trusted intermediaries, but small enough to be used by any modern device.

4 System Architecture & Implementation

System Architecture - Hyperledger Fabric

"Hyperledger Fabric is an open source enterprise-grade permissioned distributed ledger technology (DLT) platform, designed for use in enterprise contexts, that delivers some key differentiating capabilities over other popular distributed ledger or blockchain platforms" [48].

Hyperledger was established under the Linux Foundation and is maintained by a diverse set of people from multiple organizations. It's architecture is modular and configurable and enables innovation, versatility and optimization for a full set of industries. Fabric is the first distributed ledger platform that supports smart contracts which can be written in programming languages like Go. The platform is also permissioned, meaning that the participants are known to each other, rather than anonymous and untrusted. Pluggable consensus protocols are also supported and enable customization so as to fit particular use cases and trust models.

Key features:

Modularity

Fabric's platform has been designed so as to meet the most enterprise requirements.

It is comprised of the following modular components:

- **Pluggable ordering service:** Establishes consensus on the order of transactions and broadcasts blocks to peers.
- **Pluggable membership service provider:** Association of entities with cryptographic identities.
- **Peer-to-peer gossip service (optional):** Disseminates the blocks output by ordering service to other peers.
- **Smart contracts ("chaincode"):** Written in standard programming languages having no direct access to the ledger state.
- Wide range of **DBMS** support.
- **Pluggable endorsement and validation policy enforcement:** Can be independently configured per application.

Smart Contracts

Smart contracts or "chaincode" is a blockchain's application business logic.

There are three key points that apply to smart contracts:

- They run concurrently
- Can deployed dynamically
- Should be treated as untrusted

Privacy and Confidentiality

In a public blockchain network that runs a PoW consensus, transactions are executed on every node. As a result, there is no confidentiality of the contracts, nor of the transaction data, so every transaction is visible to every node in the network.

Confidentiality is enabled through Fabric's channel architecture and private data feature. In channels, participants establish a sub-network where every member has visibility to a specific transactions subset. So, only participant nodes have access to the smart contract and transaction data, preserving the privacy and confidentiality of both. Now, the private data feature allows collections between members on a channel. That allows the same protection as channels more or less without the maintenance overhead of channels.

Pluggable Consensus

The consensus component, responsible for the transactions ordering, is decoupled from the peers that execute transactions and maintain the ledger. Since consensus is modular, it's implementation can be tailored to a particular solution. This modular architecture allows the platform to be flexible and rely on well-established toolkits for CFT (crash fault-tolerant) or BFT (byzantine fault-tolerant) ordering.

Performance and Scalability

Performance can be affected by limitations like transaction size, block size, network size, hardware etc. Fabric's Performance and Scale working group is currently working on a benchmarking framework called Hyperledger Caliper so as to increase performance capabilities. The latest Fabric's version is capable of making 20,000 transactions per second.

Implementation

In this part of this dissertation we will describe the implementation of the decentralized public key infrastructure system created and also how to set up the Hyperledger Fabric on a local Debian machine and run the implementation. Following you will see screenshots of the execution of each chaincode call performed. The use case that we examine is that of the initialization of the blockchain, the addition, registration and successful login of a user via issuing a user's certificate that is binded with the user's identity, as well as the revocation of the user's certificate, validation that the user cannot login to our system anymore and as a last step the re-enable of user's certificate and a final validation of a user's successful login.

We will firstly run an instance of the Fabric test network by using the following commands [6]:

cd fabric-samples/test-network: navigate to the test-network's path

./network.sh down: kill any active or stale docker containers and remove previously generated artifacts.

```
zagoras@debian:~$ cd fabric-samples/test-network/
zagoras@debian:~/fabric-samples/test-network$ ./network.sh down
Stopping network
Stopping cli                    ... done
Stopping peer0.org1.example.com ... done
Stopping peer0.org2.example.com ... done
Stopping orderer.example.com    ... done
Removing cli                    ... done
Removing peer0.org1.example.com ... done
Removing peer0.org2.example.com ... done
Removing orderer.example.com    ... done
Removing network fabric_test
Removing volume docker_orderer.example.com
Removing volume docker_peer0.org1.example.com
Removing volume docker_peer0.org2.example.com
Removing network fabric_test
WARNING: Network fabric_test not found.
Removing volume docker_peer0.org3.example.com
WARNING: Volume docker_peer0.org3.example.com not found.
Removing remaining containers
Removing generated chaincode docker images
Untagged: dev-peer0.org2.example.com-dpki_0.07-69fafd2435aee9198c6ec5b28e2b73d05fda818a994351200b4dead2f4dd2b55-924246a29d9cc00fc5c0349206163f822c09f202894ee019f252ce554a75b8a1:latest
Deleted: sha256:f0c27608b376ad29db8447ca0c1f7c485ed391930f1913c560be42b0cea72cc3
Deleted: sha256:b950e21a8a6a4fdc512cadb5426bd0f21be1c715f7dce32dd429ac2edb236242
Deleted: sha256:eef59727fda13ebd6fa1f58d348aa60edc11da351f0ae92a361e0dfe13e87312
Deleted: sha256:30cfd0828395ad0084c5d320fb58a8e1a249b8fd527fce42d9a30493cfcda85b
Untagged: dev-peer0.org1.example.com-dpki_0.07-69fafd2435aee9198c6ec5b28e2b73d05fda818a994351200b4dead2f4dd2b55-8f96b9fa633cc58f503fb1557ab3465cee8cf12f7b73fddd3e606970fa6aa806:latest
Deleted: sha256:46343bcafdf35f51f2652d6a29aebd9cde60d7437cf909bf39bcbf87ac47f12f
Deleted: sha256:33e0ed536c547da9d59b097ff4d671366b3458081db67553f1b5d4aeee50e111
Deleted: sha256:29050489a9f350d56dc0cd1a146710840a10e84631f8a2eccfb04f40eab06d9e
Deleted: sha256:4e90a3dc9b9dafb5f88586f932f329c3967de5d8c517f7c2ba42ec84a98cc346
```

./network.sh up createChannel: start the test network

createChannel command: Creates a channel with two channel members.

```
zagoras@debian:~/fabric-samples/test-network$ ./network.sh up createChannel
Creating channel 'mychannel'.
If network is not up, starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb with crypto from 'cryptogen'
Bringing up network
Unable to find image 'hyperledger/fabric-tools:latest' locally
docker: Error response from daemon: manifest for hyperledger/fabric-tools:latest not found: manifest unknown: manifest unknown.
See 'docker run --help'.
LOCAL_VERSION=2.3.2
DOCKER_IMAGE_VERSION=
Local fabric binaries and docker images are out of  sync. This may cause problems.
/home/zagoras/fabric-samples/test-network/../bin/cryptogen
Generating certificates using cryptogen tool
Creating Org1 Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org1.yaml --output=organizations
org1.example.com
+ res=0
Creating Org2 Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org2.yaml --output=organizations
org2.example.com
+ res=0
Creating Orderer Org Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-orderer.yaml --output=organizations
+ res=0
Generating CCP files for Org1 and Org2
Creating network "fabric_test" with the default driver
Creating volume "docker_orderer.example.com" with default driver
Creating volume "docker_peer0.org1.example.com" with default driver
Creating volume "docker_peer0.org2.example.com" with default driver
Creating orderer.example.com    ... done
Creating peer0.org1.example.com ... done
Creating peer0.org2.example.com ... done
Creating cli                    ... done
CONTAINER ID        IMAGE                               COMMAND             CREATED             STATUS              PORTS                                            NAMES
198ae3ea1cbb        hyperledger/fabric-tools:2.3        "/bin/bash"         1 second ago        Up Less than a second                                                cli
9fc4629519e6        hyperledger/fabric-peer:2.3         "peer node start"   2 seconds ago       Up Less than a second   7051/tcp, 0.0.0.0:9051->9051/tcp                 peer0.org2.example.com
3397f09f8589        hyperledger/fabric-orderer:2.3      "orderer"           2 seconds ago       Up Less than a second   0.0.0.0:7050->7050/tcp, 0.0.0.0:7053->7053/tcp   orderer.example.com
00547f274597        hyperledger/fabric-peer:2.3         "peer node start"   2 seconds ago       Up Less than a second   0.0.0.0:7051->7051/tcp                           peer0.org1.example.com
Generating channel genesis block 'mychannel.block'
/home/zagoras/fabric-samples/test-network/../bin/configtxgen
+ configtxgen -profile TwoOrgsApplicationGenesis -outputBlock ./channel-artifacts/mychannel.block -channelID mychannel
2021-07-03 17:54:42.307 EEST [common.tools.configtxgen] main -> INFO 001 Loading configuration
2021-07-03 17:54:42.312 EEST [common.tools.configtxgen.localconfig] completeInitialization -> INFO 002 orderer type: etcdraft
2021-07-03 17:54:42.312 EEST [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 Orderer.EtcdRaft.Options unset, setting to tick_interval:"500ms" election_tick:10 heart
beat_tick:1 max_inflight_blocks:5 snapshot_interval_size:16777216
2021-07-03 17:54:42.312 EEST [common.tools.configtxgen.localconfig] Load -> INFO 004 Loaded configuration: /home/zagoras/fabric-samples/test-network/configtx/configtx.yaml
2021-07-03 17:54:42.314 EEST [common.tools.configtxgen] doOutputBlock -> INFO 005 Generating genesis block
```

```
2021-07-03 17:54:42.314 EEST [common.tools.configtxgen] doOutputBlock -> INFO 005 Generating genesis block
2021-07-03 17:54:42.314 EEST [common.tools.configtxgen] doOutputBlock -> INFO 006 Creating application channel genesis block
2021-07-03 17:54:42.314 EEST [common.tools.configtxgen] doOutputBlock -> INFO 007 Writing genesis block
+ res=0
Creating channel mychannel
Using organization 1
+ osnadmin channel join --channelID mychannel --config-block ./channel-artifacts/mychannel.block -o localhost:7053 --ca-file /home/zagoras/fabric-samples/test-network/organizations/ordererOr
ganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem --client-cert /home/zagoras/fabric-samples/test-network/organizations/ordererOrganizations/exam
ple.com/orderers/orderer.example.com/tls/server.crt --client-key /home/zagoras/fabric-samples/test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/ser
ver.key
+ res=0
Status: 201
{
        "name": "mychannel",
        "url": "/participation/v1/channels/mychannel",
        "consensusRelation": "consenter",
        "status": "active",
        "height": 1
}

Channel 'mychannel' created
Joining org1 peer to the channel...
Using organization 1
+ peer channel join -b ./channel-artifacts/mychannel.block
+ res=0
2021-07-03 17:54:48.553 EEST [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2021-07-03 17:54:48.644 EEST [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
Joining org2 peer to the channel...
Using organization 2
+ peer channel join -b ./channel-artifacts/mychannel.block
+ res=0
2021-07-03 17:54:51.730 EEST [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2021-07-03 17:54:51.796 EEST [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
Setting anchor peer for org1...
Using organization 1
Fetching channel config for channel mychannel
Using organization 1
Fetching the most recent configuration block for the channel
+ peer channel fetch config config_block.pb -o orderer.example.com:7050 --ordererTLSHostnameOverride orderer.example.com -c mychannel --tls --cafile /opt/gopath/src/github.com/hyperledger/fa
bric/peer/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
2021-07-03 14:54:51.964 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2021-07-03 14:54:51.966 UTC [cli.common] readBlock -> INFO 002 Received block: 0
2021-07-03 14:54:51.967 UTC [channelCmd] fetch -> INFO 003 Retrieving last config block: 0
2021-07-03 14:54:51.968 UTC [cli.common] readBlock -> INFO 004 Received block: 0
Decoding config block to JSON and isolating config to Org1MSPconfig.json
+ configtxlator proto_decode --input config_block.pb --type common.Block
```

```
+ configtxlator proto_decode --input config_block.pb --type common.Block
+ jq '.data.data[0].payload.data.config'
Generating anchor peer update transaction for Org1 on channel mychannel
+ jq '.channel_group.groups.Application.groups.Org1MSP.values += {"AnchorPeers":{"mod_policy": "Admins","value":{"anchor_peers": [{"host": "peer0.org1.example.com","port": 7051}]},"version":
 "0"}}' Org1MSPconfig.json
+ configtxlator proto_encode --input Org1MSPconfig.json --type common.Config
+ configtxlator proto_encode --input Org1MSPmodified_config.json --type common.Config
+ configtxlator compute_update --channel_id mychannel --original original_config.pb --updated modified_config.pb
+ configtxlator proto_decode --input config_update.pb --type common.ConfigUpdate
+ jq .
++ cat config_update.json
+ echo '{"payload":{"header":{"channel_header":{"channel_id":"mychannel", "type":2}},"data":{"config_update":{' '"channel_id":' '"mychannel",' '"isolated_data":' '{},' '"read_set":' '{' '"gr
oups":' '{' '"Application":' '{' '"groups":' '{' '"Org1MSP":' '{' '"groups":' '{},' '"mod_policy":' '"",' '"policies":' '{' '"Admins":' '{' '"mod_policy":' '"",' '"policy":' null, '"version"
:' '"0"' '},' '"Endorsement":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Readers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Writers"
:' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"MSP":' '{' '"mod_policy":' '"",' '"value":' null, '"version":' '"0"' '}' '},' '"version":' '"0"'
'}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '},' '"write_s
et":' '{' '"groups":' '{' '"Application":' '{' '"groups":' '{' '"Org1MSP":' '{' '"groups":' '{},' '"mod_policy":' '"Admins",' '"policies":' '{' '"Admins":' '{' '"mod_policy":' '"",' '"policy
":' null, '"version":' '"0"' '},' '"Endorsement":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Readers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '
"0"' '},' '"Writers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"AnchorPeers":' '{' '"mod_policy":' '"Admins",' '"value":' '{' '"anchor_peers"
:' '[' '{' '"host":' '"peer0.org1.example.com",' '"port":' 7051 '}' ']' '},' '"version":' '"0"' '},' '"MSP":' '{' '"mod_policy":' '"",' '"value":' null, '"version":' '"0"' '}' '},' '"version
":' '"1"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '}'
'}}}}'
+ configtxlator proto_encode --input config_update_in_envelope.json --type common.Envelope
2021-07-03 14:54:52.125 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2021-07-03 14:54:52.138 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
Anchor peer set for org 'Org1MSP' on channel 'mychannel'
Setting anchor peer for org2...
Using organization 2
Fetching channel config for channel mychannel
Using organization 2
Fetching the most recent configuration block for the channel
+ peer channel fetch config config_block.pb -o orderer.example.com:7050 --ordererTLSHostnameOverride orderer.example.com -c mychannel --tls --cafile /opt/gopath/src/github.com/hyperledger/fa
bric/peer/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
2021-07-03 14:54:52.283 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2021-07-03 14:54:52.285 UTC [cli.common] readBlock -> INFO 002 Received block: 1
2021-07-03 14:54:52.285 UTC [channelCmd] fetch -> INFO 003 Retrieving last config block: 1
2021-07-03 14:54:52.286 UTC [cli.common] readBlock -> INFO 004 Received block: 1
Decoding config block to JSON and isolating config to Org2MSPconfig.json
+ configtxlator proto_decode --input config_block.pb --type common.Block
+ jq '.data.data[0].payload.data.config'
Generating anchor peer update transaction for Org2 on channel mychannel
+ jq '.channel_group.groups.Application.groups.Org2MSP.values += {"AnchorPeers":{"mod_policy": "Admins","value":{"anchor_peers": [{"host": "peer0.org2.example.com","port": 9051}]},"version":
 "0"}}' Org2MSPconfig.json
+ configtxlator proto_encode --input Org2MSPconfig.json --type common.Config
+ configtxlator proto_encode --input Org2MSPmodified_config.json --type common.Config
+ configtxlator compute_update --channel_id mychannel --original original_config.pb --updated modified_config.pb
```

```
+ configtxlator proto_decode --input config_update.pb --type common.ConfigUpdate
+ jq .
++ cat config_update.json
+ echo '{"payload":{"header":{"channel_header":{"channel_id":"mychannel", "type":2}},"data":{"config_update":{' '"channel_id":' '"mychannel",' '"isolated_data":' '{},' '"read_set":' '{' '"gr
oups":' '{' '"Application":' '{' '"groups":' '{' '"Org2MSP":' '{' '"groups":' '{},' '"mod_policy":' '"",' '"policies":' '{' '"Admins":' '{' '"mod_policy":' '"",' '"policy":' null, '"version"
:' '"0"' '},' '"Endorsement":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Readers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Writers"
:' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"MSP":' '{' '"mod_policy":' '"",' '"value":' null, '"version":' '"0"' '}' '},' '"version":' '"0"'
'}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '},' '"write_s
et":' '{' '"groups":' '{' '"Application":' '{' '"groups":' '{' '"Org2MSP":' '{' '"groups":' '{},' '"mod_policy":' '"Admins",' '"policies":' '{' '"Admins":' '{' '"mod_policy":' '"",' '"policy
":' null, '"version":' '"0"' '},' '"Endorsement":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Readers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '
"0"' '},' '"Writers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"AnchorPeers":' '{' '"mod_policy":' '"Admins",' '"value":' '{' '"anchor_peers"
:' '[' '{' '"host":' '"peer0.org2.example.com",' '"port":' 9051 '}' ']' '},' '"version":' '"0"' '},' '"MSP":' '{' '"mod_policy":' '"",' '"value":' null, '"version":' '"0"' '}' '},' '"version
":' '"1"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '}'
'}}}}'
+ configtxlator proto_encode --input config_update_in_envelope.json --type common.Envelope
2021-07-03 14:54:52.444 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2021-07-03 14:54:52.455 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
Anchor peer set for org 'Org2MSP' on channel 'mychannel'
Channel 'mychannel' joined
```

We then need to package the chaincode. Before we do this, we need to install the chaincode dependencies.

cd fabric-samples/asset-transfer-basic/chaincode-go: navigate to the folder that contains the Go version of the asset-transfer (basic) chaincode

GO111MODULE=on go mod vendor: install the dependencies

On success, the go packages will be installed inside a vendor folder.

Now we can create the chaincode package.

cd ../../../test-network: navigate back to our working directory in the test-network folder

We can use the peer CLI to create a chaincode package in the required format. Peer binaries are located in the bin folder of the fabric-samples repository.

export PATH=${PWD}/../bin:$PATH: add peer binaries to CLI Path.

export FABRIC_CFG_PATH=$PWD/../config/: set the FABRIC_CFG_PATH to point to the core.yaml file in the fabric-samples repository.

peer version: confirm that we are able to use the peer CLI and check the version of the binaries.

peer lifecycle chaincode package dpki.tar.gz --path ../asset-transfer-basic/dpki/ --lang golang --label dpki_0.07: create the chaincode package in a tar.gz format.

```
zagoras@debian:~/fabric-samples/test-network$ cd ../asset-transfer-basic/dpki/
zagoras@debian:~/fabric-samples/asset-transfer-basic/dpki$ go build
zagoras@debian:~/fabric-samples/asset-transfer-basic/dpki$ cd chaincode/
zagoras@debian:~/fabric-samples/asset-transfer-basic/dpki/chaincode$ go build
zagoras@debian:~/fabric-samples/asset-transfer-basic/dpki/chaincode$ GO111MODULE=on go mod vendor
zagoras@debian:~/fabric-samples/asset-transfer-basic/dpki/chaincode$ cd ../../../test-network/
zagoras@debian:~/fabric-samples/test-network$ export PATH=${PWD}/../bin:$PATH
zagoras@debian:~/fabric-samples/test-network$ export FABRIC_CFG_PATH=$PWD/../config/
zagoras@debian:~/fabric-samples/test-network$ peer version
peer:
 Version: 2.3.2
 Commit SHA: 0022e8fd8
 Go version: go1.15.7
 OS/Arch: linux/amd64
 Chaincode:
  Base Docker Label: org.hyperledger.fabric
  Docker Namespace: hyperledger

zagoras@debian:~/fabric-samples/test-network$ peer lifecycle chaincode package dpki.tar.gz --path ../asset-transfer-basic/dpki/ --lang golang --label dpki_0.07
```

Now that we created the chaincode package, we can install the chaincode on the peers.

Install the chaincode package

The chaincode needs to be installed on every peer of the test network. First we install the chaincode on the Org1 peer. For this purpose, we set the following environment variables. The CORE_PEER_ADDRESS will be set to point to the Org1 peer, peer0.org1.example.com.

export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051

peer lifecycle chaincode install dpki.tar.gz: issue the peer lifecycle chaincode install command so that chaincode is installed on the peer. If the command is successful, the peer will generate and return the package identifier.

Likewise, we can install the chaincode on the Org2 peer as shown bellow:

export CORE_PEER_LOCALMSPID="Org2MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
export CORE_PEER_ADDRESS=localhost:9051

peer lifecycle chaincode install dpki.tar.gz: install the chaincode to the org2 peer.

```
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_TLS_ENABLED=true
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_LOCALMSPID="Org1MSP"
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_ADDRESS=localhost:7051
zagoras@debian:~/fabric-samples/test-network$ peer lifecycle chaincode install dpki.tar.gz
2021-07-03 17:57:10.753 EEST [cli.lifecycle.chaincode] submitInstallProposal -> INFO 001 Installed remotely: response:<status:200 payload:"\nJdpki_0.07:69fafd2435aee9198c6ec5b28e2b73d05fda818a994351200b4dead2f4dd2b55\022\tdpki_0.07" >
2021-07-03 17:57:10.753 EEST [cli.lifecycle.chaincode] submitInstallProposal -> INFO 002 Chaincode code package identifier: dpki_0.07:69fafd2435aee9198c6ec5b28e2b73d05fda818a994351200b4dead2f4dd2b55
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_LOCALMSPID="Org2MSP"
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_ADDRESS=localhost:9051
zagoras@debian:~/fabric-samples/test-network$ peer lifecycle chaincode install dpki.tar.gz
2021-07-03 17:57:32.334 EEST [cli.lifecycle.chaincode] submitInstallProposal -> INFO 001 Installed remotely: response:<status:200 payload:"\nJdpki_0.07:69fafd2435aee9198c6ec5b28e2b73d05fda818a994351200b4dead2f4dd2b55\022\tdpki_0.07" >
2021-07-03 17:57:32.334 EEST [cli.lifecycle.chaincode] submitInstallProposal -> INFO 002 Chaincode code package identifier: dpki_0.07:69fafd2435aee9198c6ec5b28e2b73d05fda818a994351200b4dead2f4dd2b55
```

Approve a chaincode definition

Now we need to approve the chaincode definition for our organization. The definition includes the important parameters of chaincode governance such as the name, version, and the chaincode endorsement policy.

For this purpose, the majority of channel members need to approve the chaincode before it can be used on our channel. Because we have only two organizations on the channel both org1 and org2 should approve it.

If an organization has installed the chaincode on their peer, they need to include the packageID in the chaincode definition approved by their organization.

peer lifecycle chaincode queryinstalled: this command will return the package id needed so as to approve the chaincode.

export CC_PACKAGE_ID=dpki_0.07:69fafd2435aee9198c6ec5b28e2b73d05fda818a994351200b4dead2f4dd2b55: we then save it as an environment variable.

peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --channelID mychannel --name dpki --version 0.07 --package-id $CC_PACKAGE_ID --sequence 1 --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem": now we approve the chaincode definition using the peer lifecycle chaincode approveformyorg command.

The command above uses the --package-id flag to include the package identifier in the chaincode definition. The --sequence parameter is an integer that keeps track of the number of times a chaincode has been defined or updated.

Now we can approve the chaincode definition as Org1. So we set the following environment variables:

export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_ADDRESS=localhost:7051

peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --channelID mychannel --name dpki --version 0.07 --package-id $CC_PACKAGE_ID --sequence 1 --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem": now approve the chaincode definition as Org1.

Same applies for org2.

```
zagoras@debian:~/fabric-samples/test-network$ peer lifecycle chaincode queryinstalled
Installed chaincodes on peer:
Package ID: dpki_0.07:69fafd2435aee9198c6ec5b28e2b73d05fda818a994351200b4dead2f4dd2b55, Label: dpki_0.07
zagoras@debian:~/fabric-samples/test-network$ export CC_PACKAGE_ID=dpki_0.07:69fafd2435aee9198c6ec5b28e2b73d05fda818a994351200b4dead2f4dd2b55
zagoras@debian:~/fabric-samples/test-network$ peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --channelID mychannel --name dpki --
version 0.07 --package-id $CC_PACKAGE_ID --sequence 1 --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert
.pem"
2021-07-03 17:58:28.340 EEST [chaincodeCmd] ClientWait -> INFO 001 txid [5a2c36bd9db97c387a5ff95bdd078bd50a27ada515eb243fc9a7b2a2e233f122] committed with status (VALID) at localhost:9051
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_LOCALMSPID="Org1MSP"
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
zagoras@debian:~/fabric-samples/test-network$ export CORE_PEER_ADDRESS=localhost:7051
zagoras@debian:~/fabric-samples/test-network$ peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --channelID mychannel --name dpki --
version 0.07 --package-id $CC_PACKAGE_ID --sequence 1 --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert
.pem"
2021-07-03 17:59:10.008 EEST [chaincodeCmd] ClientWait -> INFO 001 txid [9642fabab6d7c7327c91c9abd3bcd86e6908d8073cddb04f967bf6318440eb1d] committed with status (VALID) at localhost:7051
```

Committing the chaincode definition to the channel

After a sufficient number of organizations have approved a chaincode definition, one organization can commit the chaincode definition to the channel.

For that purpose we can use the peer lifecycle chaincode checkcommitreadiness command so as to check whether channel members have approved the same chaincode definition as follows:

peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name dpki --version 0.07 --sequence 1 --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" --output json

The above command will produce a JSON map that displays if a channel member has approved the parameters that were specified in the checkcommitreadiness command:

```
{
        "Approvals": {
                "Org1MSP": true,
                "Org2MSP": true
        }
}
```

Now the chaincode definition is ready to be committed to the channel. So, we use the peer lifecycle chaincode commit command to commit the chaincode definition to the channel.

peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --channelID mychannel --name dpki --version 0.07 --sequence 1 --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt"

peer lifecycle chaincode querycommitted --channelID mychannel --name dpki --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem": use the peer lifecycle chaincode querycommitted command to confirm that the chaincode definition has been committed to the channel.

If the chaincode was successfully committed to the channel, the querycommitted command will return the sequence and version of the chaincode definition as shown bellow:

```
zagoras@debian:~/fabric-samples/test-network$ peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name dpki --version 0.07 --sequence 1 --tls --cafile "${PWD}/organizations
/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" --output json
{
        "approvals": {
                "Org1MSP": true,
                "Org2MSP": true
        }
}
zagoras@debian:~/fabric-samples/test-network$ peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --channelID mychannel --name dpki --version 0
.07 --sequence 1 --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" --peerAddresses localhost:7051
--tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/p
eerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt"
2021-07-03 18:00:10.443 EEST [chaincodeCmd] ClientWait -> INFO 001 txid [3239986984efcdc57f58080eed83bb8ca687f3875ba4f32def3caf3bd0496fc9] committed with status (VALID) at localhost:7051
2021-07-03 18:00:10.448 EEST [chaincodeCmd] ClientWait -> INFO 002 txid [3239986984efcdc57f58080eed83bb8ca687f3875ba4f32def3caf3bd0496fc9] committed with status (VALID) at localhost:9051
zagoras@debian:~/fabric-samples/test-network$ peer lifecycle chaincode querycommitted --channelID mychannel --name dpki --cafile "${PWD}/organizations/ordererOrganizations/example.com/ordere
rs/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem"
Committed chaincode definition for chaincode 'dpki' on channel 'mychannel':
Version: 0.07, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true]
```

Invoking the chaincode

After the chaincode definition has been committed to a channel, the chaincode will be up and running on the peers joined to the channel where the chaincode was installed. The chaincode is now ready to be invoked by client applications.

At first we initialize our ledger by calling InitDPKILedger which will publish a genesis record.

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"InitDPKILedger","Args":[]}'

Then by using GetAllDPKIAssets we can retrieve our newly created dpki asset.

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"GetAllDPKIAssets","Args":[]}'

```
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"InitDPKILedger","Args":[]}'
2021-07-03 18:01:31.726 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"GetAllDPKIAssets","Args":[]}'
2021-07-03 18:01:50.658 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200 payload:"[{\"id\":\"GenesisDPKIAsset\",\"username\":\"NaN\",\"public_key\":\"NaN\",\"challenge\":\"NaN\",\"issued_on\":\"2021-07-03\",\"revoked_on\":\"\",\"revoked\":false}]"
```

Now, we are ready to start our dpki proof of concept. We start by creating our client's dpki asset. Here, we make a call to CreateDPKIAsset as follows:

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"CreateDPKIAsset","Args":["z4398rw0rwvr34jv743", "icsdAegean", "-----BEGIN PUBLIC KEY-----\nMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAsOwZWhPHn8AzYRK7nvoQ\n5dZQ5XtRDEW0KjbJbEYnXwKjmyD1aMid1ud+PmsRbSYfGh+DC5zDWnwL/MBaF742\n7VASCNmZZ1CDyJTB32dGf1Hghn2bnwQUjOJeWIttFfCz7K3G1DwRVAwwqw3yR1oq

nUYjIhVJkRk2/1Tp7DvTJKCMGvo+CB/EbVBNC00FSSNiLxPp+nCPHQdLGHut9Q7z5
nD9IIRBlanMNswohHiH79HPIyHxS9K+dJ/jEQyIWFJGLNLCo58GXxSK+Sr5uG5gOk
nJ+qNI9UjFRdzE1oBSbQ3K+gz/wGl9TxXVqBhkhcTXpOUCixtcakdSTE0ctIVEnOu
ndIyBpUnMCkhHwzDBP7vHh8qyFq5QPDpZpfEI8VaVfzl20Vl7YyuhisKHzDeWrVGX
nNMRU60Qk01vC4DrvITIantgrJGltdivclEO1H5MceTi7oov60nH26Dx8HlAxOGZb
ntW8/9RklZYTM87OULypUHLadSfWGdl8ZywWruUg8U8zGxV5DKQfELmUHeKh9ZMKK
nux6HsOYUNx67MOUEBNiBvOYKCXLqD0eJib5aQwVTzGfWZ1OkW3iGEQFa3cPTJjSk
nSj3MUf7JPNAiyRzr3Wuo+NAmbF1o4sVx0HaDXcDR1EOX4Tc4Wd/JHSLEJQdjoOtT
nWEPgL3wfIlndPaVXlehHfaUCAwEAAQ==\n-----END PUBLIC KEY-----\n"]}'

Please notice the three arguments used here: id, username and publicKey. The id is a random alphanumeric value, username is our client's username and then our client's publicKey given as a one line format having new lines as \n characters.

```
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizatio
ns/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrgani
zations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.
example.com/tls/ca.crt" -c '{"function":"CreateDPKIAsset","Args":["z4398rw0rwvr34jv743", "icsdAegean", "-----BEGIN PUBLIC KEY-----\nMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAsOwZWhPHn8AzYR
K7nvoQ\n5dZQ5XtRDEW0KjbJbEYnXwKjmyD1aMid1ud+PmsRbSYfGh+DC5zDWnwL/MBaF742\n7VASCNmZZ1CDyJTB32dGf1Hghn2bnwQUjOJeWIttFfCz7K3G1DwRVAwwqw3yR1oq\nUYjIhVJkRk2/1Tp7DvTJKCMGvo+CB/EbVBNC00FSSNiLxPp+nC
PHQdLGHut9Q7z5\nD9IIRBlanMNswohHiH79HPIyHxS9K+dJ/jEQyIWFJGLNLCo58GXxSK+Sr5uG5gOk\nJ+qNI9UjFRdzE1oBSbQ3K+gz/wGl9TxXVqBhkhcTXpOUCixtcakdSTE0ctIVEnOu\ndIyBpUnMCkhHwzDBP7vHh8qyFq5QPDpZpfEI8VaVfzl
l20Vl7YyuhisKHzDeWrVGX\nNMRU60Qk01vC4DrvITIantgrJGltdivclEO1H5MceTi7oov60nH26Dx8HlAxOGZb\ntW8/9RklZYTM87OULypUHLadSfWGdl8ZywWruUg8U8zGxV5DKQfELmUHeKh9ZMKK\nux6HsOYUNx67MOUEBNiBvOYKCXLqD0eJib
5aQwVTzGfWZ1OkW3iGEQFa3cPTJjSk\nSj3MUf7JPNAiyRzr3Wuo+NAmbF1o4sVx0HaDXcDR1EOX4Tc4Wd/JHSLEJQdjoOtT\nWEPgL3wfIlndPaVXlehHfaUCAwEAAQ==\n-----END PUBLIC KEY-----\n"]}'
2021-07-03 18:02:53.397 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
```

Then by calling GetAllDPKIAssets we can again retrieve all dpki assets in our ledger:

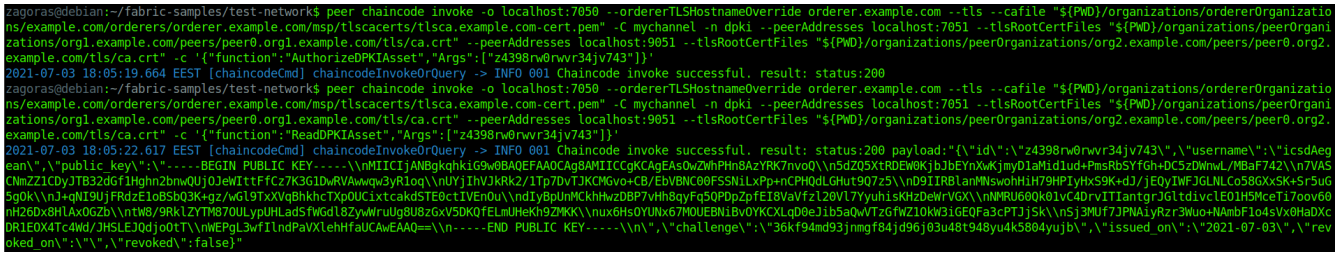
```
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizatio
ns/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrgani
zations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.
example.com/tls/ca.crt" -c '{"function":"GetAllDPKIAssets","Args":[]}'
2021-07-03 18:03:45.182 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200 payload:"[{\"id\":\"GenesisDPKIAsset\",\"username\":\"NaN\",\"p
ublic_key\":\"NaN\",\"challenge\":\"NaN\",\"issued_on\":\"2021-07-03\",\"revoked_on\":\"\",\"revoked\":false},{\"id\":\"z4398rw0rwvr34jv743\",\"username\":\"icsdAegean\",\"public_key\":\"---
--BEGIN PUBLIC KEY-----\\nMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAsOwZWhPHn8AzYRK7nvoQ\\n5dZQ5XtRDEW0KjbJbEYnXwKjmyD1aMid1ud+PmsRbSYfGh+DC5zDWnwL/MBaF742\\n7VASCNmZZ1CDyJTB32dGf1Hghn2bnw
QUjOJeWIttFfCz7K3G1DwRVAwwqw3yR1oq\\nUYjIhVJkRk2/1Tp7DvTJKCMGvo+CB/EbVBNC00FSSNiLxPp+nCPHQdLGHut9Q7z5\\nD9IIRBlanMNswohHiH79HPIyHxS9K+dJ/jEQyIWFJGLNLCo58GXxSK+Sr5uG5gOk\\nJ+qNI9UjFRdzE1oBSbQ
3K+gz/wGl9TxXVqBhkhcTXpOUCixtcakdSTE0ctIVEnOu\\ndIyBpUnMCkhHwzDBP7vHh8qyFq5QPDpZpfEI8VaVfzl20Vl7YyuhisKHzDeWrVGX\\nNMRU60Qk01vC4DrvITIantgrJGltdivclEO1H5MceTi7oov60nH26Dx8HlAxOGZb\\ntW8/9RklZYTM87OULypUHLadSfWGdl8ZywWruUg8U8zGxV5DKQfELmUHeKh9ZMKK\\nux6HsOYUNx67MOUEBNiBvOYKCXLqD0eJib5aQwVTzGfWZ1OkW3iGEQFa3cPTJjSk\\nSj3MUf7JPNAiyRzr3Wuo+NAmbF1o4sVx0HaDXcDR1EOX4Tc4Wd/JHSLEJQdjoOtT
\\nWEPgL3wfIlndPaVXlehHfaUCAwEAAQ==\\n-----END PUBLIC KEY-----\\n\",\"challenge\":\"\",\"issued_on\":\"2021-07-03\",\"revoked_on\":\"\",\"revoked\":false}]"
```

And we can also retrieve only our newly created dpki asset by calling ReadDPKIAsset as follows:

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"ReadDPKIAsset","Args":["z4398rw0rwvr34jv743"]}'

```
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizatio
ns/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrgani
zations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.
example.com/tls/ca.crt" -c '{"function":"ReadDPKIAsset","Args":["z4398rw0rwvr34jv743"]}'
2021-07-03 18:04:45.182 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200 payload:"{\"id\":\"z4398rw0rwvr34jv743\",\"username\":\"icsdAeg
ean\",\"public_key\":\"-----BEGIN PUBLIC KEY-----\\nMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAsOwZWhPHn8AzYRK7nvoQ\\n5dZQ5XtRDEW0KjbJbEYnXwKjmyD1aMid1ud+PmsRbSYfGh+DC5zDWnwL/MBaF742\\n7VAS
CNmZZ1CDyJTB32dGf1Hghn2bnwQUjOJeWIttFfCz7K3G1DwRVAwwqw3yR1oq\\nUYjIhVJkRk2/1Tp7DvTJKCMGvo+CB/EbVBNC00FSSNiLxPp+nCPHQdLGHut9Q7z5\\nD9IIRBlanMNswohHiH79HPIyHxS9K+dJ/jEQyIWFJGLNLCo58GXxSK+Sr5uG
5gOk\\nJ+qNI9UjFRdzE1oBSbQ3K+gz/wGl9TxXVqBhkhcTXpOUCixtcakdSTE0ctIVEnOu\\ndIyBpUnMCkhHwzDBP7vHh8qyFq5QPDpZpfEI8VaVfzl20Vl7YyuhisKHzDeWrVGX\\nNMRU60Qk01vC4DrvITIantgrJGltdivclEO1H5MceTi7oov60
nH26Dx8HlAxOGZb\\ntW8/9RklZYTM87OULypUHLadSfWGdl8ZywWruUg8U8zGxV5DKQfELmUHeKh9ZMKK\\nux6HsOYUNx67MOUEBNiBvOYKCXLqD0eJib5aQwVTzGfWZ1OkW3iGEQFa3cPTJjSk\\nSj3MUf7JPNAiyRzr3Wuo+NAmbF1o4sVx0HaDXc
DR1EOX4Tc4Wd/JHSLEJQdjoOtT\\nWEPgL3wfIlndPaVXlehHfaUCAwEAAQ==\\n-----END PUBLIC KEY-----\\n\",\"challenge\":\"\",\"issued_on\":\"2021-07-03\",\"revoked_on\":\"\",\"revoked\":false}"
```

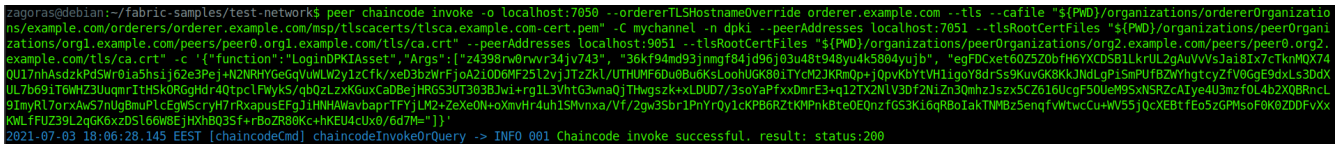
Now that we have saved our client's dpki asset having it's public key we can make an authorization request so as to update our dpki asset with a challenge string from the system for it to sign it later on with it's private key. We then retrieve again the dpki asset so to see the update in the challenge field.



```
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizatio
ns/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrgani
zations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.
example.com/tls/ca.crt" -c '{"function":"AuthorizeDPKIAsset","Args":["z4398rw0rwvr34jv743"]}'
2021-07-03 18:05:19.664 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizatio
ns/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrgani
zations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.
example.com/tls/ca.crt" -c '{"function":"ReadDPKIAsset","Args":["z4398rw0rwvr34jv743"]}'
2021-07-03 18:05:22.617 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200 payload:"{\"id\":\"z4398rw0rwvr34jv743\",\"username\":\"icsdAeg
ean\",\"public_key\":\"-----BEGIN PUBLIC KEY-----\\nMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAsOwZWhPHn8AzYRK7nvoQ\\n5dZQ5XtRDEW0KjbJbEYnXwKjmyDlaMid1ud+PmsRbSYfGh+DC5zDWnwL/MBaF742\\n7VAS
CNmZZ1CDyJTB32dGf1Hghn2bnwQUjOJeWIttFfCz7K3G1DwRVAwwqw3yR1oq\\nUYjIhVJkRk2/1Tp7DvTJKCMGvo+CB/EbVBNC00FSSNiLxPp+nCPHQdLGHut9Q7z5\\nD9IRBlanMNswohHiH79HPIyHxS9K+dJ/jEQyIWFJGLNLCo58GXxSK+Sr5uG
5gOk\\nJ+qNI9UjFRdzE1oBSbQ3K+gz/wGl9TxXVqBhkhcTXpOUCixtcakdSTE0ctIVEnOu\\ndIyBpUnMCkhHwzDBP7vHh8qyFq5QPDpZpfEI8VaVfzl20Vl7YyuhisKHzDeWrVGX\\nNMRU60Qk01vC4DrvITIantgrJGltdivclEO1H5MceTi7oov60
nH26Dx8HlAxOGZb\\ntW8/9RklZYTM870ULypUHLadSfWGdl8ZywWruUg8U8zGxV5DKQfELmUHeKh9ZMkK\\nux6HsOYUNx67MOUEBNiBvOYKCXLqD0eJib5aQwVTzGfWZ1OkW3iGEQFa3cPTJjSk\\nSj3MUf7JPNAiyRzr3Wuo+NAmbF1o4sVx0HaDXc
DR1EOX4Tc4Wd/JHSLEJQdjoOtT\\nWEPgL3wfIlndPaVXlehHfaUCAwEAAQ==\\n-----END PUBLIC KEY-----\\n\",\"challenge\":\"36kf94md93jnmgf84jd96j03u48t948yu4k5804yujb\",\"issued_on\":\"2021-07-03\",\"rev
oked_on\":\"\",\"revoked\":false}"
```


As a final step for our client's dpki registration a call to LoginDPKIAsset should be performed. Note that this call takes three arguments: id (already known), challenge (value given at the previous step) and signature. The signature here is the string value of the digital signature of the challenge value with our client's dpki asset. If the signature is correct, then login function will return a code of 200 and finelly our client would be registered and successfully loged in to the system:

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/ tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"LoginDPKIAsset","Args":["z4398rw0rwvr34jv743", "36kf94md93jnmgf84jd96j03u48t948yu4k5804yujb", "egFDCxet6OZ5ZObfH6YXCDSB1LkrUL2gAuVvVsJai8Ix7cTknMQX74QU17nhAsdzkPdSWr0ia5hsij62e3Pej+N2NRH YGeGqVuWLW2y1zCfk/xeD3bzWrFjoA2iOD6MF25l2vjJTzZkl/ UTHUMF6Du0Bu6KsLoohUGK80iTYcM2JKRmQp+jQpvKbYtVH1igoY8drSs9KuvGK8KkJNdLgPiSmPUfBZWYhgtcy ZfV0GgE9dxLs3DdXUL7b69iT6WHZ3UuqmrItHSkORGgHdr4QtpclFWykS/ qbQzLzxKGuxCaDBejHRGS3UT303BJwi+rg1L3VhtG3wnaQjTHwgszk+xLDUD7/3soYaPfxxDmrE3+q12TX2NlV3Df2Ni Zn3QmhzJszx5CZ616UcgF5OUeM9SxNSRZcAIye4U3mzfOL4b2XQBRncL 9ImyRl7orxAwS7nUgBmuPlcEgWScryH7rRxapusEFgJiHNHAWavbaprTFYjLM2+ZeXeON+oXmvHr4uh1SMvnxa/Vf/ 2gw3Sbr1PnYrQy1cKPB6RZtKMPnkBteOEQnzfGS3Ki6qRBoIakTNMBz5enqfvWtwcCu+WV55jQcXEBtfEo5zGPMsoF 0K0ZDDFvXxKWLfFUZ39L2qGK6xzDSl66W8EjHXhBQ3Sf+rBoZR80Kc+hKEU4cUx0/6d7M="]}'



```
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizatio
ns/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrgani
zations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.
example.com/tls/ca.crt" -c '{"function":"LoginDPKIAsset","Args":["z4398rw0rwvr34jv743", "36kf94md93jnmgf84jd96j03u48t948yu4k5804yujb", "egFDCxet6OZ5ZObfH6YXCDSB1LkrUL2gAuVvVsJai8Ix7cTknMQX74
QU17nhAsdzkPdSWr0ia5hsij62e3Pej+N2NRHYGeGqVuWLW2y1zCfk/xeD3bzWrFjoA2iOD6MF25l2vjJTzZkl/UTHUMF6Du0Bu6KsLoohUGK80iTYcM2JKRmQp+jQpvKbYtVH1igoY8drSs9KuvGK8KkJNdLgPiSmPUfBZWYhgtcyZfV0GgE9dxLs3DdX
UL7b69iT6WHZ3UuqmrItHSkORGgHdr4QtpclFWykS/qbQzLzxKGuxCaDBejHRGS3UT303BJwi+rg1L3VhtG3wnaQjTHwgszk+xLDUD7/3soYaPfxxDmrE3+q12TX2NlV3Df2NiZn3QmhzJszx5CZ616UcgF5OUeM9SxNSRZcAIye4U3mzfOL4b2XQBRncL
9ImyRl7orxAwS7nUgBmuPlcEgWScryH7rRxapusEFgJiHNHAWavbaprTFYjLM2+ZeXeON+oXmvHr4uh1SMvnxa/Vf/2gw3Sbr1PnYrQy1cKPB6RZtKMPnkBteOEQnzfGS3Ki6qRBoIakTNMBz5enqfvWtwcCu+WV55jQcXEBtfEo5zGPMsoF0K0ZDDFvXx
KWLfFUZ39L2qGK6xzDSl66W8EjHXhBQ3Sf+rBoZR80Kc+hKEU4cUx0/6d7M="]}'
2021-07-03 18:06:28.145 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
```


Now, in order to simulate a full functional pki alternative with ou proposed dpki solution we have to be able to revoke client certifications as well. For this purpose, we have the UpdateDPKIAsset function in which we can give a revoked value of true so to denote that this dpki asset is no longer valid. Then, the system marks also to the dpki asset the revoked date as we can see later on by calling ReadDPKIAsset function again.

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/

tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"UpdateDPKIAsset","Args":["z4398rw0rwvr34jv743", "36kf94md93jnmgf84jd96j03u48t948yu4k5804yujb", "true"]}'

```
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"UpdateDPKIAsset","Args":["z4398rw0rwvr34jv743", "36kf94md93jnmgf84jd96j03u48t948yu4k5804yujb", "true"]}'
2021-07-03 18:07:41.458 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"ReadDPKIAsset","Args":["z4398rw0rwvr34jv743"]}'
2021-07-03 18:07:44.153 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200 payload:"{\"id\":\"z4398rw0rwvr34jv743\",\"username\":\"icsdAegean\",\"public_key\":\"-----BEGIN PUBLIC KEY-----\\nMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAsOwZWhPHn8AzYRK7nvoQ\\n5dZQ5XtRDEW0KjbJbEYnXwKjmyD1aMid1ud+PmsRbSYfGh+DC5zDWnwL/MBaF742\\n7VASCNmZZ1CDyJTB32dGf1Hghn2bnwQUjOJeWIttFfCz7K3G1DwRVAwwqw3yRloq\\nUYjIhVJkRk2/1Tp7DvTJKCMGvo+CB/EbVBNC00FSSNiLxPp+nCPHQdLGHut9Q7z5\\nD9IIRBlanMNswohHiH79HPIyHxS9K+dJ/jEQyIWFJGLNLCo58GXxSK+Sr5uG5gOk\\nJ+qNI9UjFRdzE1oBSbQ3K+gz/wGl9TxXVqBhkhcTXpOUCixtcakdSTE0ctIVEnOu\\ndIyBpUnMCkhHwzDBP7vHh8qyFq5QPDpZpfEI8VaVfzl20Vl7YyuhisKHzDeWrVGX\\nNMRU60Qk01vC4DrvITIantgrJGltdivclEOlH5MceTi7oov60nH26Dx8HlAxOGZb\\ntW8/9RklZYTM87OULypUHLadSfWGdl8ZywWruUg8U8zGxV5DKQfELmUHeKh9ZMKK\\nux6HsOYUNx67MOUEBNiBvOYKCXLqD0eJib5aQwVTzGfWZ1OkW3iGEQFa3cPTJjSk\\nSj3MUf7JPNAiyRzr3Wuo+NAmbF1o4sVx0HaDXcDR1EOX4Tc4Wd/JHSLEJQdjoOtT\\nWEPgL3wfIlndPaVXlehHfaUCAwEAAQ==\\n-----END PUBLIC KEY-----\\n\",\"challenge\":\"36kf94md93jnmgf84jd96j03u48t948yu4k5804yujb\",\"issued_on\":\"2021-07-03\",\"revoked_on\":\"2021-07-03\",\"revoked\":true}"
```

Now, if we try to login with our client's dpki asset we can see that the system does not let us anymore as it's certificate is revoked.

```
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"LoginDPKIAsset","Args":["z4398rw0rwvr34jv743", "36kf94md93jnmgf84jd96j03u48t948yu4k5804yujb", "egFDCxet6OZ5ZObfH6YXCDSB1LkrUL2gAuVvVsJai8Ix7cTknMQX74QUl7nhAsdzkPdSWr0ia5hsij62e3Pej+N2NRHYGeGqVuWLW2y1zCfk/xeD3bzWrFjoA2iOD6MF25l2vjJTzZkl/UTHUMF6Du0Bu6KsLoohUGK80iTYcM2JKRmQp+jQpvKbYtVH1igoY8drSs9KuvGK8KkJNdLgPiSmPUfBZWYhgtcyZfV0GgE9dxLs3DdXUL7b69iT6WHZ3UuqmrItHSkORGgHdr4QtpclFWykS/qbQzLzxKGuxCaDBejHRGS3UT303BJwi+rg1L3VhtG3wnaQjTHwgszk+xLDUD7/3soYaPfxxDmrE3+q12TX2NlV3Df2NiZn3QmhzJszx5CZ616UcgF5OUeM9SxNSRZcAIye4U3mzfOL4b2XQBRncL9ImyRl7orxAwS7nUgBmuPlcEgWScryH7rRxapusEFgJiHNHAWavbaprTFYjLM2+ZeXeON+oXmvHr4uh1SMvnxa/Vf/2gw3Sbr1PnYrQy1cKPB6RZtKMPnkBteOEQnzfGS3Ki6qRBoIakTNMBz5enqfvWtwcCu+WV55jQcXEBtfEo5zGPMsoF0K0ZDDFvXxKWLfFUZ39L2qGK6xzDSl66W8EjHXhBQ3Sf+rBoZR80Kc+hKEU4cUx0/6d7M="]}'
Error: endorsement failure during invoke. response: status:500 message:"Sorry, your dPKI asset's certificate has already been revoked on 2021-07-03"
```

As a final step we revert our client's dpki asset revocation and then login again to the system.

```
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"UpdateDPKIAsset","Args":["z4398rw0rwvr34jv743", "36kf94md93jnmgf84jd96j03u48t948yu4k5804yujb", "false"]}'
2021-07-03 18:08:52.377 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"LoginDPKIAsset","Args":["z4398rw0rwvr34jv743", "36kf94md93jnmgf84jd96j03u48t948yu4k5804yujb", "egFDCxet6OZ5ZObfH6YXCDSB1LkrUL2gAuVvVsJai8Ix7cTknMQX74QUl7nhAsdzkPdSWr0ia5hsij62e3Pej+N2NRHYGeGqVuWLW2y1zCfk/xeD3bzWrFjoA2iOD6MF25l2vjJTzZkl/UTHUMF6Du0Bu6KsLoohUGK80iTYcM2JKRmQp+jQpvKbYtVH1igoY8drSs9KuvGK8KkJNdLgPiSmPUfBZWYhgtcyZfV0GgE9dxLs3DdXUL7b69iT6WHZ3UuqmrItHSkORGgHdr4QtpclFWykS/qbQzLzxKGuxCaDBejHRGS3UT303BJwi+rg1L3VhtG3wnaQjTHwgszk+xLDUD7/3soYaPfxxDmrE3+q12TX2NlV3Df2NiZn3QmhzJszx5CZ616UcgF5OUeM9SxNSRZcAIye4U3mzfOL4b2XQBRncL9ImyRl7orxAwS7nUgBmuPlcEgWScryH7rRxapusEFgJiHNHAWavbaprTFYjLM2+ZeXeON+oXmvHr4uh1SMvnxa/Vf/2gw3Sbr1PnYrQy1cKPB6RZtKMPnkBteOEQnzfGS3Ki6qRBoIakTNMBz5enqfvWtwcCu+WV55jQcXEBtfEo5zGPMsoF0K0ZDDFvXxKWLfFUZ39L2qGK6xzDSl66W8EjHXhBQ3Sf+rBoZR80Kc+hKEU4cUx0/6d7M="]}'
2021-07-03 18:08:54.690 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
zagoras@debian:~/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n dpki --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"ReadDPKIAsset","Args":["z4398rw0rwvr34jv743"]}'
2021-07-03 18:09:03.930 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200 payload:"{\"id\":\"z4398rw0rwvr34jv743\",\"username\":\"icsdAegean\",\"public_key\":\"-----BEGIN PUBLIC KEY-----\\nMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAsOwZWhPHn8AzYRK7nvoQ\\n5dZQ5XtRDEW0KjbJbEYnXwKjmyD1aMid1ud+PmsRbSYfGh+DC5zDWnwL/MBaF742\\n7VASCNmZZ1CDyJTB32dGf1Hghn2bnwQUjOJeWIttFfCz7K3G1DwRVAwwqw3yRloq\\nUYjIhVJkRk2/1Tp7DvTJKCMGvo+CB/EbVBNC00FSSNiLxPp+nCPHQdLGHut9Q7z5\\nD9IIRBlanMNswohHiH79HPIyHxS9K+dJ/jEQyIWFJGLNLCo58GXxSK+Sr5uG5gOk\\nJ+qNI9UjFRdzE1oBSbQ3K+gz/wGl9TxXVqBhkhcTXpOUCixtcakdSTE0ctIVEnOu\\ndIyBpUnMCkhHwzDBP7vHh8qyFq5QPDpZpfEI8VaVfzl20Vl7YyuhisKHzDeWrVGX\\nNMRU60Qk01vC4DrvITIantgrJGltdivclEOlH5MceTi7oov60nH26Dx8HlAxOGZb\\ntW8/9RklZYTM87OULypUHLadSfWGdl8ZywWruUg8U8zGxV5DKQfELmUHeKh9ZMKK\\nux6HsOYUNx67MOUEBNiBvOYKCXLqD0eJib5aQwVTzGfWZ1OkW3iGEQFa3cPTJjSk\\nSj3MUf7JPNAiyRzr3Wuo+NAmbF1o4sVx0HaDXcDR1EOX4Tc4Wd/JHSLEJQdjoOtT\\nWEPgL3wfIlndPaVXlehHfaUCAwEAAQ==\\n-----END PUBLIC KEY-----\\n\",\"challenge\":\"36kf94md93jnmgf84jd96j03u48t948yu4k5804yujb\",\"issued_on\":\"2021-07-03\",\"revoked_on\":\"\",\"revoked\":false}"
```

4 Conclusions & Future Work

This dissertation addressed the design and implementation of a Blockchain Based Decentralized Public Key Infrastructure solution, based on a private blockchain platform Hyperledger Fabric (HLF) Blockchain platform. In our proposed solution for the DPKI solution, users access to the system is managed via the use of X.509v3 digital certificates. The smart contract executed by the DPKI peers dictates the issuance, authentication and revocation of users access in the system created. The system is a proof of concept for a DPKI implementation in HLF. We have implemented the proposed solution by producing an available prototype and that can be used for research and experimental studies by interested researchers and developers. The prototype implements the design assumptions and all the components discussed in the dissertation. As a future work we would further expand current smart contract implementation so as to include more functionality. We would also deploy current implementation in a cloud based environment where the blockchain peers can run in different virtual or dedicated machines from different service providers. Last but not least, after the testing in the virtual machines was performed we could also implement this to an Internet of Things network and run all the necessary performance testing evaluations and benchmarks to compare the two testbeds (cloud-based vs physical) make our observations and any optimizations needed.

Bibliography.

1. Privacy based decentralized Public Key Infrastructure (PKI) implementation using Smart contract in Blockchain Sivakumar and Dr. Kunwar Singh, National Institute of Technology, Trichy, Tamil Nadu 620015 shibv3@gmail.com

2. DPKI: A Blockchain-Based Decentralized Public Key Infrastructure System, Alexander Papageorgiou Inlecom Innovation Athens, Greece alex.papageorgiou@inlecomsystems.com Antonis Mygiakis Inlecom Innovation Athens, Greece antonis.mygiakis@inlecomsystems.com Konstantinos Loupos Inlecom Innovation Athens, Greece konstantinos.loupos@inlecomsystems.com Thomas Krousarlis Inlecom Innovation Athens, Greece thomas.krousarlis@inlecomsystems.com

3. Decentralized Public Key Infrastructure: A White Paper from Rebooting the Web of Trust by (alphabetical by last name) Christopher Allen, Arthur Brock, Vitalik Buterin, Jon Callas, Duke Dorje, Christian Lundkvist, Pavel Kravchenko, Jude Nelson, Drummond Reed, Markus Sabadello, Greg Slepak, Noah Thorp, and Harlan T Wood

4.Blockchain-based PKI for Crowdsourced IoT Sensor Information Guilherme Pinto, Jo~ao Pedro Dias, and Hugo Sereno Ferreira, Faculty of Engineering, University of Porto, Portugal, INESC TEC and Department of Informatics Engineering, Faculty of Engineering, University of Porto, Portugal

5. Blockchain-Enabled DPKI Framework - Miguel Pires Egídio Reis

6. https://hyperledger-fabric.readthedocs.io/en/latest/deploy_chaincode.html

7. https://cryptotools.net/rsagen

8. https://www.devdungeon.com/content/working-files-go

9. https://hyperledger-fabric.readthedocs.io/en/latest/deploy_chaincode.html

10. https://medium.com/@Raulgzm/export-import-pem-files-in-go-67614624adc7

11. https://play.golang.org/

12. https://golang.org/pkg/crypto/x509/

13. https://golang.org/pkg/crypto/rsa/

14. https://gist.github.com/diorahman/274e667b0bb99d854806

15. https://golang.org/pkg/crypto/ecdsa/

16.https://golang.hotexamples.com/site/file?hash=0x78ebbcaeb905bf955b9f8d12969dd70c9bb7f45e111730fa409833566f4a00cc&fullName=ztools/tls.go&project=xtalentfeng/zgrab_

17. https://stackoverflow.com/questions/44230634/how-to-read-an-rsa-key-from-file/44231740

18. https://en.wikipedia.org/wiki/Blockchain

19. https://en.wikipedia.org/wiki/Merkle_tree

20. Mastering Bitcoin, Second Edition Andreas M. Antonopoulos 2017

21. https://criptomo.com/two-simultaneous-blocks/

22. https://www.matsherman.com/p/2020-hard-fork

23. https://decrypt.co/resources/byzantine-fault-tolerance-what-is-it-explained

24. https://nakamotoinstitute.org/static/docs/computer-systems-by-mutually-suspicious-groups.pdf

25. Bitcoin: A Peer-to-Peer Electronic Cash System, Satoshi Nakamoto

26. S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In Journal of Cryptology, vol 3, no 2, pages 99-111, 1991.

27. D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In Sequences II: Methods in Communication, Security and Computer Science, pages 329-334, 1993.

28. S. Haber, W.S. Stornetta, "Secure names for bit-strings," In Proceedings of the 4th ACM Conference on Computer and Communications Security, pages 28-35, April 1997.

29. Secure Property Titles with Owner Authority, Nick Szabo, Originally published in 1998

30. https://unenumerated.blogspot.com/2005/12/bit-gold.html

31. https://itchronicles.com/what-is-blockchain/

32. Ethereum White Paper: A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM, Vitalik Buterin

33. https://101blockchains.com/history-of-blockchain-timeline/

34. https://eos.io/

35. https://academy.horizen.io/technology/expert/blockchain-as-a-data-structure/

36. https://www.researchgate.net/profile/Hong-Ning-Dai/publication/318131748/figure/fig1/AS:613940996345865@1523386348329/An-example-of-blockchain-which-consists-of-a-continuous-sequence-of-blocks.png

37. PROTOCOLS FOR PUBLIC KEY CRYPTOSYSTEMS, Ralph C. Merkle, ELXSi International, Sunnyvale, Ca.

38. Public Key Infrastructure, Implementation and Design, Suranjan Choudhury with Kartik Bhatnagar, Wasim Haque, & NIIT

39. Understanding Cryptography, a textbook for students and practitioners, Christof Paar, Jan Pelsi

40. https://en.wikipedia.org/wiki/Public_key_infrastructure

41. https://3.bp.blogspot.com/-SdDDFVbo5Fk/VGKICEq1JYI/AAAAAAAAAgQ/RPRWbXszkX0/s1600/x.png

42. PKI Fundamentals, iQ.Suite Crypt Pro Basics

43. https://www.ssl.com/faqs/what-is-a-certificate-authority/

44. https://www.primekey.com/wiki/what-is-a-registration-authority/

45. https://www.tutorialspoint.com/cryptography/public_key_infrastructure.htm

46. https://id4d.worldbank.org/sites/id4d-ms8.extcc.com/files/inline-images/18%20digital%20certificates.png

47. https://docs.oracle.com/cd/E53394_01/html/E54787/figures/L10n_scpki.auth.jpg

48. https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html

49. https://decrypt.co/resources/byzantine-fault-tolerance-what-is-it-explained

50. https://www.e-zigurat.com/innovation-school/blog/public-vs-private-blockchain-whats-the-difference/

51. Analysis of the main consensus protocols of blockchain. Shijie Zhanga, Jong-Hyouk Leeb, a Protocol Engineering Lab., Sangmyung University, Republic of Korea Sejong University, Republic of Korea

52. https://medium.com/hackergirl/decentralized-public-key-infrastructure-4e7ea9173bac

53. https://miro.medium.com/max/1400/1*z2REkO4RIp73p4wESu_0IA.png

Appendix A

Chaincode:

1. Main function

```
/*
SPDX-License-Identifier: Apache-2.0
*/

package main

import (
	"log"

	"dpki/chaincode"

	"github.com/hyperledger/fabric-contract-api-go/contractapi"
)

func main() {
	dPKIAssetChaincode, err := contractapi.NewChaincode(&chaincode.DPKISmartContract{})
	if err != nil {
		log.Panicf("Error creating dPKI chaincode: %v", err)
	}

	if err := dPKIAssetChaincode.Start(); err != nil {
		log.Panicf("Error starting dPKI chaincode: %v", err)
	}
}
```

2. Main implementation code

```
package chaincode

import (
	"crypto"
	"crypto/rsa"
	"crypto/sha256"
	"crypto/x509"
	"encoding/base64"
	"encoding/json"
	"encoding/pem"
	"fmt"
	"io"
	"log"
	"os"
	"time"

	"github.com/hyperledger/fabric-contract-api-go/contractapi"
)

// SmartContract provides functions for managing an Asset
type DPKISmartContract struct {
	contractapi.Contract
}

// Asset describes basic details of what makes up a simple asset
type DPKIAsset struct {
	ID        string `json:"id"`
	Username  string `json:"username"`
	PublicKey string `json:"public_key"`
	Challenge string `json:"challenge"`
	IssuedOn  string `json:"issued_on"`
	RevokedOn string `json:"revoked_on"`
	Revoked   bool   `json:"revoked"`
}

// DPKIInitLedger adds a base set of dpki assets to the ledger.
func (s *DPKISmartContract) InitDPKILedger(ctx contractapi.TransactionContextInterface) error {

	// Create and populate values to dPKIAssets slice.
	dPKIAssets := []DPKIAsset{
		{ID: "GenesisDPKIAsset", Username: "NaN", PublicKey: "NaN", Challenge: "NaN", IssuedOn:
time.Now().UTC().Format("2006-01-02")},
	}

	// Iterate through dPKIAssets slice and perform a PutState action for each dPKI asset.
	for _, dPKIAsset := range dPKIAssets {
		dPKIAssetJSON, err := json.Marshal(dPKIAsset)
		if err != nil {
			return err
		}

		err = ctx.GetStub().PutState(dPKIAsset.ID, dPKIAssetJSON)
		if err != nil {
			return fmt.Errorf("failed to put to world state. %v", err)
		}
	}

	return nil
}

// DPKIAssetExists returns true when asset with given ID exists in world state
func (s *DPKISmartContract) DPKIAssetExists(ctx contractapi.TransactionContextInterface, id string) (bool, error) {
	DPKIAssetJSON, err := ctx.GetStub().GetState(id)
	if err != nil {
```

```
                return false, fmt.Errorf("failed to read from world state: %v", err)
        }

        return DPKIAssetJSON != nil, nil
}

// CreateDPKIAsset issues the client's dPKI asset so that this is later used as a login token.
func (s *DPKISmartContract) CreateDPKIAsset(ctx contractapi.TransactionContextInterface, id string, username string, publicKey
string) error {

        // Enable logging.
        log.SetOutput(io.MultiWriter(os.Stdout))

        // Check if the generated id is already used.
        exists, err := s.DPKIAssetExists(ctx, id)
        if err != nil {
                return err
        } else if exists {
                fmt.Errorf("dPKI Asset id given is already used. Please try again with a new id.")
        }

        // Parse and validate client's public key.
        block, _ := pem.Decode([]byte(publicKey))
        if block == nil {
                return fmt.Errorf("Invalid PEM format given.")
        }

        // Create the dPKI Asset model.
        dPKIAsset := DPKIAsset{
                ID:       id,
                Username:  username,
                PublicKey: publicKey,
                IssuedOn:  time.Now().UTC().Format("2006-01-02"),
        }

        // Marshal the afore mentioned asset to it's json form.
        dPKIAssetJSON, err := json.Marshal(dPKIAsset)
        fmt.Println("dPKIAssetJSON")
        fmt.Println(dPKIAssetJSON)
        if err != nil {
                return err
        }

        return ctx.GetStub().PutState(id, dPKIAssetJSON)
}

// ReadDPKIAsset returns the dPKI asset stored in the world state with given id.
func (s *DPKISmartContract) ReadDPKIAsset(ctx contractapi.TransactionContextInterface, id string) (*DPKIAsset, error) {

        // Enable logging.
        log.SetOutput(io.MultiWriter(os.Stdout))

        // Retrieve dPKI asset.
        dPKIAssetJSON, err := ctx.GetStub().GetState(id)
        if err != nil {
                return nil, fmt.Errorf("failed to read from world state: %v", err)
        }
        if dPKIAssetJSON == nil {
                return nil, fmt.Errorf("the dPKI asset %s does not exist", id)
        }
        log.Println("Found dPKI asset:\n", dPKIAssetJSON)

        var dPKIAsset DPKIAsset
        // Marshal the afore mentioned asset to it's json form.
        err = json.Unmarshal(dPKIAssetJSON, &dPKIAsset)
        if err != nil {
```

```
                return nil, err
        }

        return &dPKIAsset, nil
}

// GetAllDPKIAssets returns all dpki assets found in world state.
func (s *DPKISmartContract) GetAllDPKIAssets(ctx contractapi.TransactionContextInterface) ([]*DPKIAsset, error) {
        // range query with empty string for startKey and endKey does an
        // open-ended query of all assets in the chaincode namespace.
        resultsIterator, err := ctx.GetStub().GetStateByRange("", "")
        if err != nil {
                return nil, err
        }
        defer resultsIterator.Close()

        var dPKIAssets []*DPKIAsset
        for resultsIterator.HasNext() {
                queryResponse, err := resultsIterator.Next()
                if err != nil {
                        return nil, err
                }

                var dPKIAsset DPKIAsset
                err = json.Unmarshal(queryResponse.Value, &dPKIAsset)
                if err != nil {
                        return nil, err
                }
                dPKIAssets = append(dPKIAssets, &dPKIAsset)
        }

        return dPKIAssets, nil
}

// UpdateDPKIAsset updates an existing asset in the world state with provided parameters.
func (s *DPKISmartContract) UpdateDPKIAsset(ctx contractapi.TransactionContextInterface, id string, challenge string, revoked
bool) error {

        // Enable logging.
        log.SetOutput(io.MultiWriter(os.Stdout))
        var revokedOn string

        // Validate dPKI asset's existance.
        exists, err := s.DPKIAssetExists(ctx, id)
        if err != nil {
                return err
        }
        if !exists {
                return fmt.Errorf("the asset %s does not exist", id)
        }

        // Find client's dPKI asset.
        dPKIAsset, err := s.ReadDPKIAsset(ctx, id)

        if revoked {
                revokedOn = time.Now().UTC().Format("2006-01-02")
        }

        // Overwriting original asset with new asset.
        dPKIAssetNew := DPKIAsset{
                ID:        dPKIAsset.ID,
                Username:  dPKIAsset.Username,
                PublicKey: dPKIAsset.PublicKey,
                Challenge: challenge,
                IssuedOn:  dPKIAsset.IssuedOn,
                RevokedOn: revokedOn,
```

```
		Revoked:   revoked,
	}
	dPKIAssetNewJSON, err := json.Marshal(dPKIAssetNew)
	if err != nil {
		return err
	}

	return ctx.GetStub().PutState(id, dPKIAssetNewJSON)
}

// AuthorizeDPKIAsset set's the challenge attribute of the client's dPKI asset.
// This is later signed with the client's private key and id fed to the login function so that the client gets to be authenticated.
func (s *DPKISmartContract) AuthorizeDPKIAsset(ctx contractapi.TransactionContextInterface, id string) error {

	// Enable logging.
	log.SetOutput(io.MultiWriter(os.Stdout))

	// Find client's dPKI asset.
	dPKIAsset, err := s.ReadDPKIAsset(ctx, id)
	if dPKIAsset.Revoked {
		return fmt.Errorf("Sorry, your dPKI asset's certificate has already been revoked on %v", dPKIAsset.RevokedOn)
	}

	// Parse and validate client's public key.
	block, _ := pem.Decode([]byte(dPKIAsset.PublicKey))
	if block == nil {
		return fmt.Errorf("Invalid PEM format given.")
	}

	// Parse the client's public RSA key given.
	_, err = x509.ParsePKIXPublicKey(block.Bytes)
	if err != nil {
		return fmt.Errorf("Invalid x509 public key format found while parsing dPKI asset: ", err)
	}

	// Set a hardcoded challenge so as to avoid peers response diversity. For testing purposes.
	challenge := "36kf94md93jnmgf84jd96j03u48t948yu4k5804yujb"

	// Update client's dPKI asset with the challenge attribute.
	err = s.UpdateDPKIAsset(ctx, id, challenge, false)
	if err != nil {
		return fmt.Errorf("An error occured while updating the client's dPKI asset: ", err)
	}

	return nil
}

// LoginDPKIAsset is the last step so as to validate that the client is registered to the system, by succesfully logging in.
func (s *DPKISmartContract) LoginDPKIAsset(ctx contractapi.TransactionContextInterface, id string, challenge string, signature string) error {

	// Enable logging.
	log.SetOutput(io.MultiWriter(os.Stdout))

	// Find client's dPKI asset.
	dPKIAsset, err := s.ReadDPKIAsset(ctx, id)
	if dPKIAsset.Revoked {
		return fmt.Errorf("Sorry, your dPKI asset's certificate has already been revoked on %v", dPKIAsset.RevokedOn)
	}

	// Check that the challenge given matches the one in client's dPKI asset.
	if challenge != dPKIAsset.Challenge {
		return fmt.Errorf("Challenge given does not match the one in client's dPKI asset.")
	}
```

```
	// Parse and validate client's public key.
	block, _ := pem.Decode([]byte(dPKIAsset.PublicKey))
	if block == nil {
		return fmt.Errorf("Invalid PEM format given.")
	}

	// Parse the client's public RSA key given.
	publicKey, err := x509.ParsePKIXPublicKey(block.Bytes)
	if err != nil {
		return fmt.Errorf("Invalid x509 public key format found while parsing dPKI asset: ", err)
	}

	// Decode client's signature.
	signatureDecoded, err := base64.StdEncoding.DecodeString(signature)
	if err != nil {
		return fmt.Errorf("An error occured while parsing client's signature: ", err)
	}

	// Verify client's signature.
	hash := sha256.Sum256([]byte(challenge))
	err = rsa.VerifyPKCS1v15(publicKey.(*rsa.PublicKey), crypto.SHA256, hash[:], signatureDecoded)
	if err != nil {
		return fmt.Errorf("An error occured while verifying client's signature: ", err)
	}

	log.Println("Congratulations! You have succesfully logged in to the system!")

	return nil
}
```

3. Client's public key:

-----BEGIN PUBLIC KEY-----\nMIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAsOwZWhPHn8AzYRK7nvoQ\
n5dZQ5XtRDEW0KjbJbEYnXwKjmyD1aMid1ud+PmsRbSYfGh+DC5zDWnwL/MBaF742\
n7VASCNmZZ1CDyJTB32dGf1Hghn2bnwQUjOJeWIttFfCz7K3G1DwRVAwwqw3yR1oq\
nUYjIhVJkRk2/1Tp7DvTJKCMGvo+CB/EbVBNC00FSSNiLxPp+nCPHQdLGHut9Q7z5\
nD9IIRBlanMNswohHiH79HPIyHxS9K+dJ/jEQyIWFJGLNLCo58GXxSK+Sr5uG5gOk\nJ+qNI9UjFRdzE1oBSbQ3K+gz/
wGl9TxXVqBhkhcTXpOUCixtcakdSTE0ctIVEnOu\
ndIyBpUnMCkhHwzDBP7vHh8qyFq5QPDpZpfEI8VaVfzl20Vl7YyuhisKHzDeWrVGX\
nNMRU60Qk01vC4DrvITIantgrJGltdivclEO1H5MceTi7oov60nH26Dx8HlAxOGZb\
ntW8/9RklZYTM87OULypUHLadSfWGdl8ZywWruUg8U8zGxV5DKQfELmUHeKh9ZMKK\
nux6HsOYUNx67MOUEBNiBvOYKCXLqD0eJib5aQwVTzGfWZ1OkW3iGEQFa3cPTJjSk\
nSj3MUf7JPNAiyRzr3Wuo+NAmbF1o4sVx0HaDXcDR1EOX4Tc4Wd/JHSLEJQdjoOtT\
nWEPgL3wfllndPaVXlehHfaUCAwEAAQ==\n-----END PUBLIC KEY-----\n

4. Client's private key:

-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAACFwAAAAdzc2gtcn
NhAAAAAwEAAQAAAgEAufMsPCGyr8zQbSs1f7u3bF6XO+TKyySa5yUGXLZzqNOQc33tURQU
XSh3SzlImTZYo/XWFNSfrThuuJp//1zUESbZ6yhzMDSnmr6qcFzqaM4jwNapsXiptPc+m0
/nqkEmeRaQeK7ZlJ5AFk0fsvK7nl9esY1NpgQkk3/Rfh7ygD5eX9/8fLIofJZshkzHTVrQ
fXbhP/O2xq2h8zKK362Q6y8TAjprdDkeeONP0iAztTUoWVQEmxmt9ziQ+g0AFBPMwITS8o
gqoZIHdRXYHG1Ps4YV2WjpNMx6LGhn6KnoZ1bVZamAw126PX3IbhscTW7IewKix850p46/
c/o7icVE1ReC9RaCOQk9iUvhyr/WTR5RaEe2k0+0HI9gBS6A0gFzZEeFQcfUDVEbscjYYt
M0y5mQi+M2kaDqu9te3fw796N9quYpmaKd0WB/+tquwYF1uP/R5lFAnGYQ6N4Uicbhg5fn
yjptG76FCgRljU46BD5EQLHFbul+x2MsSQKUhlRHUOe3tp0/BDlOJSk8FWpEwYMJ83v9d7
qiCRWR1KMZ7i/WOhrHRj90Fj9j8DHtTtqNYG7bR2mygmNFph5SguWfh3UFTsNf86vs4u/L
zU+KBoitiLQYqJJ9rylBmujUAyZiqAe9nOCtWBAzXAeoYq2iR3EugaC2kcY+Ub5EXi2RwQ
EAAAdQlAM9+pQDPfoAAAAHc3NoLXJzYQAAAgEAufMsPCGyr8zQbSs1f7u3bF6XO+TKyySa
5yUGXLZzqNOQc33tURQUXSh3SzlImTZYo/XWFNSfrThuuJp//1zUESbZ6yhzMDSnmr6qcF
zqaM4jwNapsXiptPc+m0/nqkEmeRaQeK7ZlJ5AFk0fsvK7nl9esY1NpgQkk3/Rfh7ygD5e
X9/8fLIofJZshkzHTVrQfXbhP/O2xq2h8zKK362Q6y8TAjprdDkeeONP0iAztTUoWVQEmx
mt9ziQ+g0AFBPMwITS8ogqoZIHdRXYHG1Ps4YV2WjpNMx6LGhn6KnoZ1bVZamAw126PX3I
bhscTW7IewKix850p46/c/o7icVE1ReC9RaCOQk9iUvhyr/WTR5RaEe2k0+0HI9gBS6A0g
FzZEeFQcfUDVEbscjYYtM0y5mQi+M2kaDqu9te3fw796N9quYpmaKd0WB/+tquwYF1uP/R
5lFAnGYQ6N4Uicbhg5fnyjptG76FCgRljU46BD5EQLHFbul+x2MsSQKUhlRHUOe3tp0/BD
lOJSk8FWpEwYMJ83v9d7qiCRWR1KMZ7i/WOhrHRj90Fj9j8DHtTtqNYG7bR2mygmNFph5S
guWfh3UFTsNf86vs4u/LzU+KBoitiLQYqJJ9rylBmujUAyZiqAe9nOCtWBAzXAeoYq2iR3
EugaC2kcY+Ub5EXi2RwQEAAAADAQABAAACADWIJxzsipeHvB0hilZKgUSmep1YgXU4BYwz
oYBBWkypE3XcM9zajf4/rxAt3pOVDLGvDUGT8KAale15NQ4b+EUzzRqg24BuOC4ZY9HTPf
yME8jpQ5L+7MRXRWPK/iOKe/vyHgdQNv1NDaAd9dk7ff3a8NTmnAsGOE1qR7IqUJ/Q1LR/
NB3l4Df1yipjoiDH2yuzdKdHjz/lLrFnxIkeW5EJTUyf6t1bsEUXE5ynfptVQVduE6FfJD
D+zOXfpj0hchmeY2hFARjghFsVvdB5O6+wE0wyoqh7kVSFfhyCBZDyc56/zHzx17LqKjBg
h6ZXFUoseRZSfkzakI1f5ZIFA4pjBZvJJ8qESwB5jMzikgr0lKLJljQwoQEUSdBYEXLcaz
o8Ywkm+TEooS13iQU5mWgG7h6yB8jXpx5FoHVGTKX9pA6aNaY1yvFCVzSOF91XcswKllUX
Ay0XjJFhC2JfluH6VX1KSxwAAQm9Wa5W5gFs37RfwH4hD5zpi/tDApt9/5keD5l8+SkNw+
MyJb6MFrakneHTQQdCdGHclmfi7mQkB/uY7UJ/aiCiUPUk/qPGi8r1UrZuNy0HLw+73+Nf
U5EhIUsGMYX5qfNzKJy1GWp5khMMyIGxaj94tyegC/hXccdVI9PC2/A+FAswZelOdBWtxU
iv2pFyfGaLKL7ujFIRAAABAEpU0/NLQarmETpgXcZTmupCLuKY/dUk3My/GhrcCaXwRmlZ
229uKMxxuzI1X8/CDHLrlE3mM+j0tkm9TnDR4rBSOu8Kv9oi7g4HA5jokW7M5ltCjf/tjV
aSRZa4hBT8mpAoOWizVK3jcJWug1pDJlweSzXeWaJKtSCEAa+qiDeVY2+OVdDaMwh3DwJi
q3UgJtfjhdJk5krPEoWJDEgst5Ox7fMJuVwtcjbb50t8p++NvP2b5kDvE/uWKcSTUsLiMD
gF1z+mMWgFn3Ut0FXLhzv1/4EI/JP5H9HEOYN27cXhERr83NnJCZe8GGjcZFAiOtm03R5+
RjwEZWwY7whpHiAAAAEBAO7hLdyfYa1zqr8OeU7D8wKZIq7QZHmCrrH8oVQqjBLdn0lGrM
Hf2/XksDX8PuTr6fXKce1RprG/Xt9ixoVV20XMXihyjpSpSKkEBGweXZzXwQxn1LyrMbGN
k1fg4tN2NrfZRNO0Ya4rY8KHfgRjGqfEqnQZir3GnpfPwNaWa5BmwIG2WKZoYDEqxehzmJ
aPR4T+EXaHFdvWDoMezTVY3zFhOKj2KVYTEW71CHnrxqr6jzt9EC76H/AKLuV+vekgg7hP
8RBA1/vfg59yu7bV2iXcyHSCtK1UGfAk7tmEFeItmUp//apWSTB4wYHA/UBtX9G9+RB+vb
lMTBPUX0H60bUAAAEBAMdG3uuS0VU6VF92MMNoRnPQKUo94VkrNIGOlYeERZut5jbnTRh7
56BpFjcresaBoXEbP/bdWR6ULqMgkMkA//1vkeIkugC8vS7T8XmrywK5W2doY/ptK7Wk0p
CSSbUUytgBEOrOn2ZgAMg7cYgBpjwTNeknb4NxvQQsxddvOD3bt+25FE5EID2IGzudQxFE
MgUKbNA4kfABuFESZm6T+4RnvZNK7TjiA/QOd2XkL6JbUIOMu4LiPZe+jb8wdCQr99roFW
Y09wlEZ8CIZecML35PyTSkUMKalZONXmrMMmuYcAAhzrhtS7cyCAeedH8VuOF3GC+QpQxI

WHe9s8ldsZ0AAAAaWNzZG0xMTcwMDVAaWNzZC5hZWdlYW4uZ3IB
-----END OPENSSH PRIVATE KEY-----

5. Client's signature

egFDCxet6OZ5ZObfH6YXCDSB1LkrUL2gAuVvVsJai8Ix7cTknMQX74QU17nhAsdzkPdSWr0ia5hsij62e3Pej+N2NRHYGeGq
VuWLW2y1zCfk/xeD3bzWrFjoA2iOD6MF25l2vjJTzZkl/
UTHUMF6Du0Bu6KsLoohUGK80iTYcM2JKRmQp+jQpvKbYtVH1igoY8drSs9KuvGK8KkJNdLgPiSmPUfBZWYhgtcyZfV0G
gE9dxLs3DdXUL7b69iT6WHZ3UuqmrItHSkORGgHdr4QtpclFWykS/
qbQzLzxKGuxCaDBejHRGS3UT303BJwi+rg1L3VhtG3wnaQjTHwgszk+xLDUD7/3soYaPfxxDmrE3+q12TX2NlV3Df2NiZn3Q
mhzJszx5CZ616UcgF5OUeM9SxNSRZcAIye4U3mzfOL4b2XQBRncL9ImyRl7orxAwS7nUgBmuPlcEgWScryH7rRxapusEFgJi
HNHAWavbaprTFYjLM2+ZeXeON+oXmvHr4uh1SMvnxa/Vf/
2gw3Sbr1PnYrQy1cKPB6RZtKMPnkBteOEQnzfGS3Ki6qRBoIakTNMBz5enqfvWtwcCu+WV55jQcXEBtfEo5zGPMsoF0K0Z
DDFvXxKWLfFUZ39L2qGK6xzDSl66W8EjHXhBQ3Sf+rBoZR80Kc+hKEU4cUx0/6d7M=

6. Client's Go code so as to create the digital signature:

```
package main

import (
	"fmt"
	"crypto"
	"crypto/rand"
	"crypto/rsa"
	"crypto/sha256"
	"crypto/x509"
	"encoding/base64"
	"encoding/pem"
)

func main() {
	// Create signature
	hashed := sha256.Sum256([]byte("36kf94md93jnmgf84jd96j03u48t948yu4k5804yujb"))
	block, _ := pem.Decode([]byte(`-----BEGIN RSA PRIVATE KEY-----
MIIJKAIBAAKCAgEAsOwZWhPHn8AzYRK7nvoQ5dZQ5XtRDEW0KjbJbEYnXwKjmyD1
aMid1ud+PmsRbSYfGh+DC5zDWnwL/MBaF7427VASCNmZZ1CDyJTB32dGf1Hghn2b
nwQUjOJeWIttFfCz7K3G1DwRVAwwqw3yR1oqUYjIhVJkRk2/1Tp7DvTJKCMGvo+C
B/EbVBNC00FSSNiLxPp+nCPHQdLGHut9Q7z5D9IIRBlanMNswohHiH79HPIyHxS9
K+dJ/jEQyIWFJGLNLCo58GXxSK+Sr5uG5gOkJ+qNI9UjFRdzE1oBSbQ3K+gz/wGl
9TxXVqBhkhcTXpOUCixtcakdSTE0ctIVEnOudIyBpUnMCkhHwzDBP7vHh8qyFq5Q
PDpZpfEI8VaVfzl20Vl7YyuhisKHzDeWrVGXNMRU60Qk01vC4DrvITIantgrJGlt
divclEO1H5MceTi7oov60nH26Dx8HlAxOGZbtW8/9RklZYTM87OULypUHLadSfWG
dl8ZywWruUg8U8zGxV5DKQfELmUHeKh9ZMKKux6HsOYUNx67MOUEBNiBvOYKCXLq
D0eJib5aQwVTzGfWZ1OkW3iGEQFa3cPTJjSkSj3MUf7JPNAiyRzr3Wuo+NAmbF1o
4sVx0HaDXcDR1EOX4Tc4Wd/JHSLEJQdjoOtTWEPgL3wfIlndPaVXlehHfaUCAwEA
AQKCAgAgU9PCENEuEImS2EBuKRVDWejINcM2cdocyIv/e8Yf5zSL2PbeoaGtrfj0
YM37WrbeKBni2k8bzoTGN0N/CSOyMypIcbJFHYIm+X9/WbiY9RYInRT9dlpm78n7
deaF1siZm5s0FpG4AM43w0Gc5g3LfindqpNmATjnNltb/UBwVA4cbc59swGiWC6i
uVHWu7K2WGsgDq2PCntnFPJ7mSENUfxurQ9Qi6jo1svzVfjb/ClKYpi6V/W+O6rs
aPFOF7uTsVeK5KW0+VfdPJbmmo25OXN2s7BIX1OImq6XkBVwsWom019GszLmQkHx
1sOUSXUIctvAo/0U7ae+v0JYb8jVdwgpXdSYZ7XgWGseFqZy7+CkArFM4B+o7Rcf
4dU/TYGpf2Tk0mjPiLlEbA46lizaAySg2gnytYgrE3CZ344ynUC+HMdJDqcpGNbc
4fl4hxfBRiR/ep0wKBAmPuyrtXtjsvl0ZAhkCUvEFL6HajhCsmNBf214Xu4diFOB
/82Nnv7w47QTWg6zbR2GwPBEtXlsLUH3F+EDRfSpDFAcSflO4DDUhOvzEnpZQxwT
gYe0Fu9TFWtnngqwtw0zGRwAPbMtid2D6u/4C6tXfsAu9J07X/EwVVWY7p6Bj/0A
hlHqikPWYE5j+pOqQ2y+3Wv+9PMGASBuUUrRB75FCXHMsloJ9QKCAQEA5N4WFEvO
QE9k/0NmYVvluV/U5gBhEpsg5pB/WP9MACbynjCcfvuqciWUrktEG6GqJbAgZ9Jz
V7R/RTE7AtyoERlH84fJ5JpKTkxjxvZyQVgqiNKyF+nUazFhVeA8K+ehmaJMiW3L
Q1d6Fiu22cyHYAD0lrq+OYq7XCTx7OeyoO9m/56foVMf+Aa0tBilDKH/peok/ya7
f1oIlb2Q6jM8UF1tygaFn2R2RUl+Yw7iqqweIyInJQFqxDrf4jd69it2k8JrR5ro
pdFSKSc5fAKBwKk0XkP49LZJvDTNKAnE0UPpJJBUNxtDCb+A7WXf/92xVKJNQ3YK
1B8WMNQ0GYa8NwKCAQEAxeWF426dNVlxk6YdNibu9PY/vpQPixjci73LsWojLKK7
B4Rn/D7g/JYu9PYDl9bLzQqYDXPQwKILSSpHPJoI6vShO5HgDo9ne0G/DOUBoZfW
NJZWXwoL8fNnKlqtYivhauplrlXBK40BCGQrbHo5VeKE/E8BUIoR+6NVJkfAQ6pk
XPRhw0NiBGbu3xnIwki59c0a0QNZZbQVBwO9tEDCcHC2pNagKJhvxdOB4RKpa69t
R+K0DKpSh07ZTBFYu1pAWS+5K8c+uyUfmarSeubVho1Eg029R9c07OrpX0l2rSxX
lgy6JyiADV3zmi0IiWvMQdCtD3ve7Fmj2+vlPEx/AwKCAQEAz6uBZ1s9xHcGUuWK
oa8sbR+65BTWAHcIzI2zMEM/aCfvIIFFj6e2iqr1FY4wN85iwJ3nWa3JgXzDuzon
rLoiOpmxAxZkqO0jnZaNNv1qwUZKGirs9Ov1FmsRQkYc803UAb8WKeGZffqDSljS
KyD+eauERL0gXMA4sCzJ5Mh8+rEgybPabx0pNfqHv59ZLHRWr+sdNPSlT7KXdaA0
PX7OLLIKLDXsy6vx8X71ktbv8CQLmJ+l21tE0NgHTJJBHdxkVUEl7SxwC/46yiLQ
c2km6XXfkeWlog9CKufSfJEYqTYw+D+KWxJ+qvq89yzoq5gzD3PSHfOfccjfbuEu
gTcT0wKCAQB8tcvOK2L+v8MHXOODwL1NufEp1HyHF7/EgHLg9xX7hwF+Fz7Ag4+d
Hbexv2c8RKxiJo7zy0x+WJ/sF6yU0C1Wg5snwyIDivOXhTM0mQySHPFsamE7u0UT
0GPvGP2ypByGF/jECF6XZ2CDPTzwO4XuTbWomtElRB1ZCvEaEpVjDvn4ajlBqDEY
vRVDmygc51/pOrvvg8DOitHFfeIKudTOrvkqn+aGLqohbMyoXe5OYlrntkHoyV0A
z/u4KCZWyuVKTvzOTd2GajqjyDakJNvi0xud81uY7lH4C4HN/qm/L/ZyJMsa6jbo
NZDSVBbRIxGxWCP3YgR1xXexm9L2UO6pAoIBACmyQg/SyBCo/Lj9Q/yA+LZH1LZp
gGQq4HPQbhMJctIPAJelydRBST7fwprL2IARTO3j9LDdReQ9h0x7LCe1aFMccn8F
DzSlUbwKI4C22gSQ57LqyJaCDzMZwDl8es0J1bsbg607cDQcWkAvo6xOBW04JQ1P
N6ViXBpsBvPbOnrVjkEPa3r4TlKPgRsDvGFdM9hTzph1LWsd7k50XsygaHflY3er
```

```
zcAYlCaQFECYmFawK2JA08hwAPcBywG9M3eO55nbLjriQMqeGJ+2T4CldS+C39Yh
rkLGLVEesRXqwz2nDga34qYfPJ5eb2CskkIxnF071eddclx1ddtwXu/m4Fw=
-----END RSA PRIVATE KEY-----
`))
fmt.Println("block.Bytes")
fmt.Println(block.Bytes)
		if block == nil {
			fmt.Println("Invalid PEM format given.")
		}
		privateKey, err := x509.ParsePKCS1PrivateKey(block.Bytes)
	if err != nil {
	  fmt.Println(err)
	}
fmt.Println("privateKey")
fmt.Println(privateKey)
		signature, _ := rsa.SignPKCS1v15(rand.Reader, privateKey, crypto.SHA256, hashed[:])

		// Encode signature
		signatureBase64 := base64.StdEncoding.EncodeToString(signature)

		fmt.Println(signatureBase64)
}
```