Delft University of Technology, In Partial Fulfilment of the Requirements For the Bachelor of Computer Science and Engineering

Sign Hyperledger Fabric Smart Contracts using the TPM

Z. N. Starke*
Supervisor: Kaitai Liang†
EEMCS, Delft University of Technology, The Netherlands

January 24, 2022

Abstract

Block-chain technology is gaining momentum in both industry and academics. With this momentum there are a lot of potential gains, but also potential risk involved. This papers proposes a solution for security risks, like a man-in-the-middle-attack, of the permissioned block-chain distributed ledger software Hyperledger Fabric. A prototype is introduces that uses the Trusted Hardware Module for attestation. By combining the key hierarchy present in the TPM, to sign assets before storing them on the ledger.

1 Introduction

Block-chain technology is getting significant attention in both industry and academics. In the industry, we see various applications of block-chain technology like cryptocurrency and Non-fungible tokens (NFT's). Furthermore, in 2021 the market cap of the 11 biggest cryptocurrencies totaled one trillion dollars [16]. Apart from the popular applications, there are various practical applications for block-chain technology. With the advent of smart contracts, which are programs stored on a blockchain, the practical applications in supply chain management[13], healthcare [14] and the oil and gas industry[15] become more evident. According to a recent study [1] there are 159 papers and 12 surveys related to the construction and execution of smart contracts. These papers discussed topics like the impact, research challenges and opportunities of block-chain.

Although block-chain technology seems promising, based on the high market cap and research activity, there are a couple of issues that hinder widespread adoption. One of these issues is security. Security vulnerabilities can lead to huge economic losses in certain situations like the hacker attack on the crowdfunding project Decentralized Autonomous Organization in 2016 [6].

This project uses smart contracts on the Ethereum network where developers and investors are brought together to make the development of block-chain based applications possible. One vulnerability in the contract code caused an economic loss of 60 million dollars [6]. Therefore enhancing the security of smart contracts can bring a future where the world can make secure, fast, and decentralized transactions one step closer.

*`{z.n.starke}@student.tudelft.nl`
†`{Kaitai Liang}@tudelft.nl`

This research aims to answer the question: "How to use trusted hardware to make the execution of smart contracts more secure?". There are a variety of trusted hardware components with various algorithms to secure smart contracts. Take for example the Trusted Platform Module (TPM) [2], Intel SGX [3] and (PUF)-based security solutions for IoT [4]. These hardware components enable trusted computing which in turn creates a foundation of trust for software processes.

There is various literature on the security of smart contracts. As Kongmanee, J., Kijsanayothin, P., & Hewett, R. [18] state "correctness of execution alone is not sufficient to guarantee security of smart contracts". In their paper, they propose a formal verification technique to make the development of smart contracts more secure. With this technique, they can specify many possible executions that lead to security breaches. Cecchetti et al. [19] look at the security of smart contracts from the perspective of information flow. Another paper [20] discusses the issues regarding the proof-of-stake consensus algorithm. In this paper, they propose a hybrid consensus algorithm that can be used in systems for smart contract management. However, Practical research into the integration of trusted hardware and smart contracts is still lacking.

The goal of this research project is to develop a simple smart contract that uses the digital signature algorithm to secure the execution of the smart contract within a block-chain network. Consequently, the main research question can be broken up into the following sub-questions:

1. How does the digital signature algorithm work?
2. How does the execution of a smart contract within a block-chain network work?
3. How to integrate the digital signature algorithm into the execution of a simple asset transfer smart contract?
4. What effect does the digital signature algorithm have on the performance of the smart contract execution?

This paper follows the same structure as the sub-questions. First, there is a theoretical consideration chapter, where important technical concepts used in the paper are explained like sub-questions one and two. After the theoretical considerations comes the experimental setup. In the experimental setup chapter practical information about the development environment, block-chain network topology and smart contract implementation are given. Thereby answering the third sub-question. Thereafter, there will be a discussion about the performance implications that the digital signature algorithm has on the execution of the smart contract. This answers sub-question 4 and as usual this paper ends with some concluding sentences.

2 Theoretical consideration

2.1 Block-chain technology

The core ideas behind block-chain were first introduced in 1991 and 1998. In 1991 a paper [9] was issued which introduced the concept of a digital ledger and a technique to show that digitally signed documents were not tampered with. Moreover, in 1998 Leslie Lamport [7] introduced a consensus model for reaching an agreement on a result in an unreliable network of computers. Later the famous paper about an electronic cash system [8] combined the before mentioned articles to develop the first of many block-chain applications. In the remaining part of this subsection, there will be brief explanations of the different components that make up a block-chain network.

The first component is the distributed ledger. A ledger is an append-only list that records the whole transactional history of the block-chain. The distributed part comes from the fact that the ledger is recorded on every peer in the network.

The second component is the collection of nodes. Nodes are individual systems within a block-chain that work together to handle and record transactions. Every block-chain context has different roles, but one of the most fundamental roles inside the network is that of the peer. A peer is responsible for hosting the ledger and validating transactions [10].

A block-chain is a distributed network of computers, therefore there must be a way for the network to come to a joint agreement on results. An example of this challenge is deciding the order of transactions, in other words deciding which user publishes the next block. This is handled by the consensus mechanism. The consensus mechanism is the reason there is no need for a trusted third party because each node in the network handles the validation of transactions and comes to a consensus on the order of these transactions.

Smart contracts that are used in block-chains like Ethereum are quite different from the ones used in the Bitcoin block-chain. The big difference is that the former is Turing Complete, which means that smart contracts can implement complex business logic in languages like Golang, Java, etc [11].

2.2 Hyperledger fabric

According to Shrivas, M. K., & Yeboah, D. T. [17] there are 2 types of block-chain categorized by the level of authorization needed within the network. On the one hand, there are permissionless block-chains where everyone can join using their own resources. On the other hand, there are permissioned block-chains where prior to joining the network there should be a degree of authorization. Hyperledger Fabric is an example of a permissioned block-chain. Each block-chain has its own terminology for describing the functionalities and components of its technology. In the next subparagraph, there will be a brief introduction of the terminology and structure used by Hyperledger Fabric.

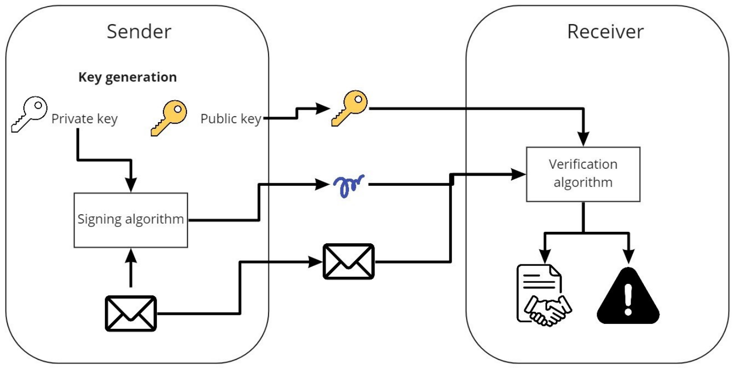
Assets are at the heart of Hyperledger Fabric. They define anything that can have monetary value which can be exchanged over the network. The mechanism that defines the business logic that governs the assets is called chaincode, this is a smart contract in block-chain terms. The identities that create, update and transfer assets using the smart contracts are called organizations. Organizations are members of the network that are allowed to interact with the network. Each organization comes with its own set of peers, orderers, admins and clients. These identities are issued by certificate authorities who also belong to an organization. First, the participant must be registered and upon enrolling receive their own MSP folder. An MSP is also known as a membership service provider is a set of certificates that reside on the network. The MSP is used to identify if the component is authorized to do what it tries to do.

2.3 Trusted Platform Module

The Trusted Platform Module, TPM, is a security module that is a building block for trusted computing. The Trusted Computing Group has developed standards to describe the meaning of trusted computing. The Trusted Computing Group (TCG) defines the trusted platform module as follows: "TPM (Trusted Platform Module) is a computer chip (microcontroller) that can securely store artifacts used to authenticate the platform (your PC or laptop)." [5]. The basic security features that the TPM provides are protected capabilities, integrity measurements, storage and reporting [2]. The basic idea behind the TPM is to move security from software to hardware. This idea is based on the fact that if the infrastructure where the software runs is compromised the security of the software can be bypassed by a malicious actor. This is one of the biggest flaws in modern security systems.

The main functionality of the TPM is to provide tools for secure authentication of the system itself. One of those tools is key encryption. Normally keys would be stored on the system in the form of a file, however, on systems with a TPM these keys are stored inside the trusted hardware [2]. This measure ensures that even if the system is compromised the keys cannot leave the TPM, so the keys cannot be stolen. Other cryptographic tools that the TPM provides are secure storage, random number generation, various key engines and registers for processing from within the TPM [2].

Figure 1: High-level overview of digital signature algorithm



2.4 Digital signature

Figure 1 shows a high-level overview of the digital signature process. The flow of the algorithm is as follows: First, the sender generates the public-private key pair. Then the sender signs the message using his private key to generate a signature. The public key, signature and message are then sent to the receiver. The receiver in turn uses the sender's public key together with the signature to derive the message of the sender. This process ensures that the message was sent by the sender and was not tampered with during transmission. Otherwise the signature and public key would not produce the original message.

3 Experimental Setup

In the following sections, there will be a more detailed discussion of the different tools that were used in the development of the signed smart contract. After the discussion of the tools, there will be a detailed explanation about the experimental procedure that produced the result which can be found in chapter 4.

3.1 Tools

To be able to answer the main research question different tools were explored to benefit the design and implementation of a prototype. This prototype uses the cryptographic functionalities of the TPM to digitally sign assets to secure the execution of a smart contract. In the remainder of this section, this prototype will be defined as a signed smart contract.

The first two tools were used to set up a development environment. Linux Ubuntu 20.04 was chosen as the development environment because of its support and strong community that proposes and shares solutions. To run Linux on Windows, Hyper-V was used to run a virtual machine that emulates Linux Ubuntu 20.04.

The third and fourth tools used were Hyperledger Fabric and Hyperledger Caliper developed by the Hyperledger Foundation. Hyperledger Fabric is an open-source project to enable the development of modular software architecture. It is a distributed software package that has specifically been designed for use in the development of enterprise-level block-chain applications. Hyperledger Caliper is a benchmark tool to measure the performance of a block-chain implementation.

The fifth tool used was the programming language Golang for the development of smart contracts. This language was best suited, because of the ease of learning and it is one of the few languages supported by the Hyperledger Fabric software. This naturally led to the final tool used which is the Go-TPM library to communicate directly with the TPM installed on a system.

3.2 Experimental protocol

3.2.1 The technical roadmap

Figure 8 in the appendix shows the technical road map of this project. First, the virtual machine and Linux were configured and set up. Then a test network using the predefined network Hyperledger Fabric provides was set up. During the setup of the test network, the framework and development language were studied. In the next two weeks, the TPM was configured and a hello world example was built using the Go-TPM library. In the fifth and sixth week, the business logic that governs the block-chain implementation was developed. After the implementation of the asset transfer smart contract, the cryptographical functionalities of the TPM were leveraged to implement the digital signature algorithm into the asset transfer smart contract. During the merging of the smart contract with the TPM functionalities, a technical hurdle was encountered. Hyperledger Fabric uses a containerized approach, which means that every component of the network was launched as a docker container. However, the TPM device file is not accessible to the docker container where the chaincode runs. Therefore, a new test network without the use of docker was built to resolve this issue. After that, a chaincode as a service approach was taken to enable the functionalities of the TPM in the chaincode. For this, a chaincode server was run locally and only a package containing the address of the chaincode server was installed on the peer. The last week was used for the writing of the report and finishing up the smart contract.

3.2.2 The network topology

A visual representation of the network topology can be found in figure 3 in the appendix. The network consists of two organizations that both run one peer and one orderer organization. All three organization have their own CA running within their network. In figure 3 it is also shown where the different MSP folders to identify nodes in the network reside.

The colored arrows in figure 3 represent the different communication channels between the components of the network. The black arrows indicate communication between the components belonging to the same organization. The CLI tools, also known as a client, are used to propose transactions to the peers. The peers simulate and validate the proposals. After the simulation, an endorsement response is sent back to the client. If the endorsement response is successful, the client will sent the endorsed transaction proposal to the orderer service (the orange arrows), which will order the proposals into blocks. The size of the block is decided upon by the organizations participating in the network. The blue arrows indicate the communication between the peers and the chaincode server. Instead of the chaincode being installed on every peer, the peers communicate with the chaincode server which is responsible for the smart contract execution. After the execution of the smart contract, the chaincode server sends back a response and payload back to the peer.

Algorithm 1 SignAsset

```
$pubKey, privKey \leftarrow$ TPMGenKeyPair()
$hash \leftarrow$ sha256(asset)
$sig \leftarrow$ TPMSign(hash, privKey)
storeOnChain(sig, pubKey, asset)
```

Algorithm 2 VerifyAsset

```
$sig, pubKey, asset \leftarrow$ queryLedger(assetID)
$hash \leftarrow$ sha256(asset)
if TPMVerify(hash, pubKey, sig) then
    return true
else
    return false
end if
```

3.2.3 The smart contract

There were 3 distinct smart contracts developed. The sequence diagrams of these smart contracts can be viewed in the appendix. In figures 4, 5 and 6 the flow of the messages between the different actors of the network for the create, update and transfer smart contracts are described. The first layer of communication between the client and peer is represented in figure 3 with the black arrows. These messages consist of transaction proposals and endorsement responses. Except for figure 6 there the first layer is the price agreement phase of the smart contract this happens out-of-band between the organizations.

The layer of communication between the peers and the chaincode server is indicated by the blue arrows in figure 3. The peer sends an API call to the chaincode server with as the first parameter the function name of the chaincode it wants to invoke. The rest of the parameters are the input arguments for the function. In both figures 4 and 6 the chaincode connects to the TPM to sign an asset, this process is visually shown in figure 7.

First, the server opens a channel to the TPM, through which a key hierarchy can be instantiated. The TPM contains 3 types of keys, namely endorsement, storage and platform keys. The endorsement keys are used to identify the TPM, storage keys that can be used by local applications and platform keys are used by the TPM itself. The key hierarchy that is used for the prototype, first creates a storage root key which will act as a parent for the application key. The storage root key will enable the application key to attest that it originated from the TPM. The application key has the ability to sign assets and the signatures provide the authentication, non-repudiation of the signer and integrity of the asset. In other words, it enforces the facts that the sender is real, the sender cannot deny having signed the asset and that the asset has not been altered. The SignAsset and VerifyAsset algorithms describe the process of signing and verifying the asset.

The SignAsset algorithm first generates a public and private key pair. Before the asset can be signed the asset is hashed using the sha256 algorithm. The signing algorithm is generally slower than the hashing algorithm, so this is used to save time and reduce the length of the produced signature. After the asset has been hashed, this hash is signed using the application key of the TPM. At the end the signature, public key and asset are stored on the ledger.

The verifyAsset algorithm reads an asset from the ledger and computes its signature. Then the TPM verifies that hash, public key and signature if the asset has not been tampered with since the time of signing this function returns true, otherwise it returns false.

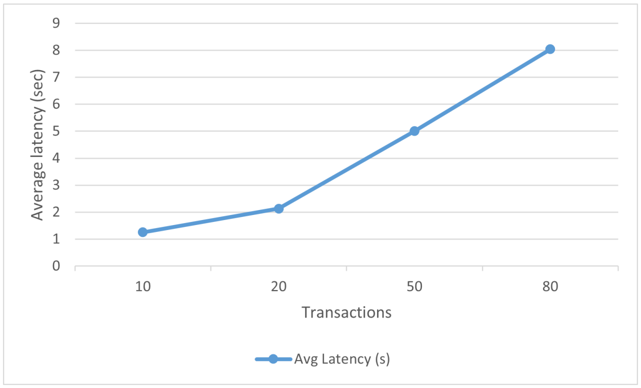


Figure 2: Average latency of the create asset smart contract

4 Result and discussion

Using Hyperledger Caliper the performance of the created asset smart contract was measured, the result is shown in figure 2. As shown in the figure the latency increases as the number of transactions increases. This is as expected because the load on the network increases. In other words, the higher the workload on the network the longer it takes for an asset to get created. This indicates that there could be a potential bottleneck when the smart contract gets used in a work-intensive network. However, the network used to test this smart contract was static, so this can be handled by dynamically adjusting the number of peers in the network, based on the load on the network.

Efforts have been made to benchmark the other smart contracts, but due to the characteristics of the block-chain network, this effort was not successful. For the update and transfer function, there already should have been assets on the block-chain to reference, but between populating the network with test data and querying that data there is not enough time. Some of the early created assets are already committed on the chain, so when reading the asset to update or transfer it. There is a multi-version concurrency control read conflict error, which indicates that there has been a dirty read. In other words, the smart contracts are not developed to test their performance using the techniques of Hyperledger Caliper. Sadly, there was not enough time to come up with a workaround.

The development of this prototype introduces a lot of complexity to the block-chain implementation. To be able to run an chaincode as a service advanced knowledge of Hyperledger is needed. This could could hinder the adoption of this method of securing smart contract execution. However, the security guarantees achieved by this prototype is worth the effort. Also, as soon as the chaincode server is built there is no need to update chaincodes on every peer where the chaincode would be installed which is also introduces complexity. As long as the server address doesn't change the peers can invoke methods and the chaincode can be invoked.

5 Conclusions and Future Work

This research aimed to see how trusted hardware could be used to secure the execution of smart contracts. Through this paper, the different research questions stated in the introduction were answered. By first investigating the knowledge needed to understand the question and come up with a design for a prototype. Then give a detailed explanation about how to implement such a design. In the end, it is shown how to use the TPM to implement a digital signature algorithm to secure the execution of an asset transfer smart contract. During the development of the secured smart contract, the potential of the TPM was not fully used, for example, encryption. Encryption could be used to make the execution of smart contracts even more secure by encrypting the asset before storing it on the chain. In this way, there could be added privacy and security to the block-chain implementation. To conclude, it is possible to use hardware-based attestation to enhance the security of smart contracts.

6 Responsible Research

All the code is available online at: gitlab.com/HeyZedd/tpm-smart-contract, so the whole project can be rebuilt based on the gitlab repository. There is also an read me which explains how to setup the network and interact with the network. For the curious readers who want to conduct similar research, all the literature used is available through academic databases. So as long as the reader has access to those, it is possible to reproduce the results found in this paper.

Although, the digital signature algorithm used claims to give certain security guarantees, this research did not do any form of formal security validation or testing of the proposed security guarantees. This could be a topic for future research, to test and validate the security guarantees this prototype claims to give.

References

[1] Hu, B., Zhang, Z., Liu, J., Liu, Y., Yin, J., Lu, R., & Lin, X. (2021). A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems. Patterns, 2(2), 100179.

[2] Ezirim, K., Khoo, W., Koumantaris, G., Law, R., & Perera, I. M. (2012). Trusted Platform Module-A Survey. The Graduate Center of The City University of New York, 11.

[3] Zheng, W., Wu, Y., Wu, X., Feng, C., Sui, Y., Luo, X., & Zhou, Y. (2021). A survey of Intel SGX and its applications. Frontiers of Computer Science, 15(3), 1-15.

[4] Shamsoshoara, A., Korenda, A., Afghah, F., & Zeadally, S. (2020). A survey on physical unclonable function (PUF)-based security solutions for Internet of Things. Computer Networks, 183, 107593.

[5] Trusted Computing Group. "Trusted platform module (TPM) summary," March 2018. [Online]. Available: https://trustedcomputinggroup.org/resource/trusted-platform-module-tpm-summary/.

[6] Atzei, N., Bartoletti, M., & Cimoli, T. (2017, April). A survey of attacks on ethereum smart contracts (sok). In International conference on principles of security and trust (pp. 164-186). Springer, Berlin, Heidelberg.

[7] Lamport, L. (1998, May) The Part-Time Parliament. in ACM Trans. Comput. Syst. (pp. 1331-69). ACM, New York.

[8] Nakamoto, S., ”Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008. https://bitcoin.org/bitcoin.pdf

[9] Narayanan, A., Bonneau, J., Felten, E., Miller, A., and Goldfede, S., Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction, Princeton University Press, 2016.

[10] Yaga, D., Mell, P., Roby, N., & Scarfone, K. (2019). Blockchain technology overview. arXiv preprint arXiv:1906.11078.

[11] Zhang, C. (2019). Truxen: A Trusted Computing Enhanced Blockchain. arXiv preprint arXiv:1904.08335.

[12] Fang, W., Chen, W., Zhang, W., Pei, J., Gao, W., & Wang, G. (2020). Digital signature scheme for information non-repudiation in blockchain: a state of the art review. EURASIP Journal on Wireless Communications and Networking, 2020(1), 1-15.

[13] Song, J. M., Sung, J., & Park, T. (2019). Applications of blockchain to improve supply chain traceability. Procedia Computer Science, 162, 119-122.

[14] McGhin, T., Choo, K. K. R., Liu, C. Z., & He, D. (2019). Blockchain in healthcare applications: Research challenges and opportunities. Journal of Network and Computer Applications, 135, 62-75.

[15] Lu, H., Huang, K., Azimi, M., & Guo, L. (2019). Blockchain technology in the oil and gas industry: A review of applications, opportunities, challenges, and risks. Ieee Access, 7, 41426-41444.

[16] Christopher, E., & Jumma, M. A. A. M. S. Role of Blockchain and Cryptocurrency to Redefine the Future Economy.

[17] Shrivas, M. K., & Yeboah, D. T. (2018, December). The disruptive blockchain: Types, platforms and applications. In Fifth Texila World Conference for Scholars (TWCS) on Transformation: The Creative Potential of Interdisciplinary.

[18] Kongmanee, J., Kijsanayothin, P., & Hewett, R. (2019, November). Securing smart contracts in blockchain. In 2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW) (pp. 69-76). IEEE.

[19] Cecchetti, E., Yao, S., Ni, H., & Myers, A. C. (2020, May). Securing smart contracts with information flow. In International Symposium on Foundations and Applications of Blockchain.

[20] H. Watanabe, S. Fujimura, A. Nakadaira, Y. Miyazaki, A. Akutsu and J. Kishigami, "Blockchain contract: Securing a blockchain applied to smart contracts," 2016 IEEE International Conference on Consumer Electronics (ICCE), 2016, pp. 467-468, doi: 10.1109/ICCE.2016.7430693.

7 Appendix A

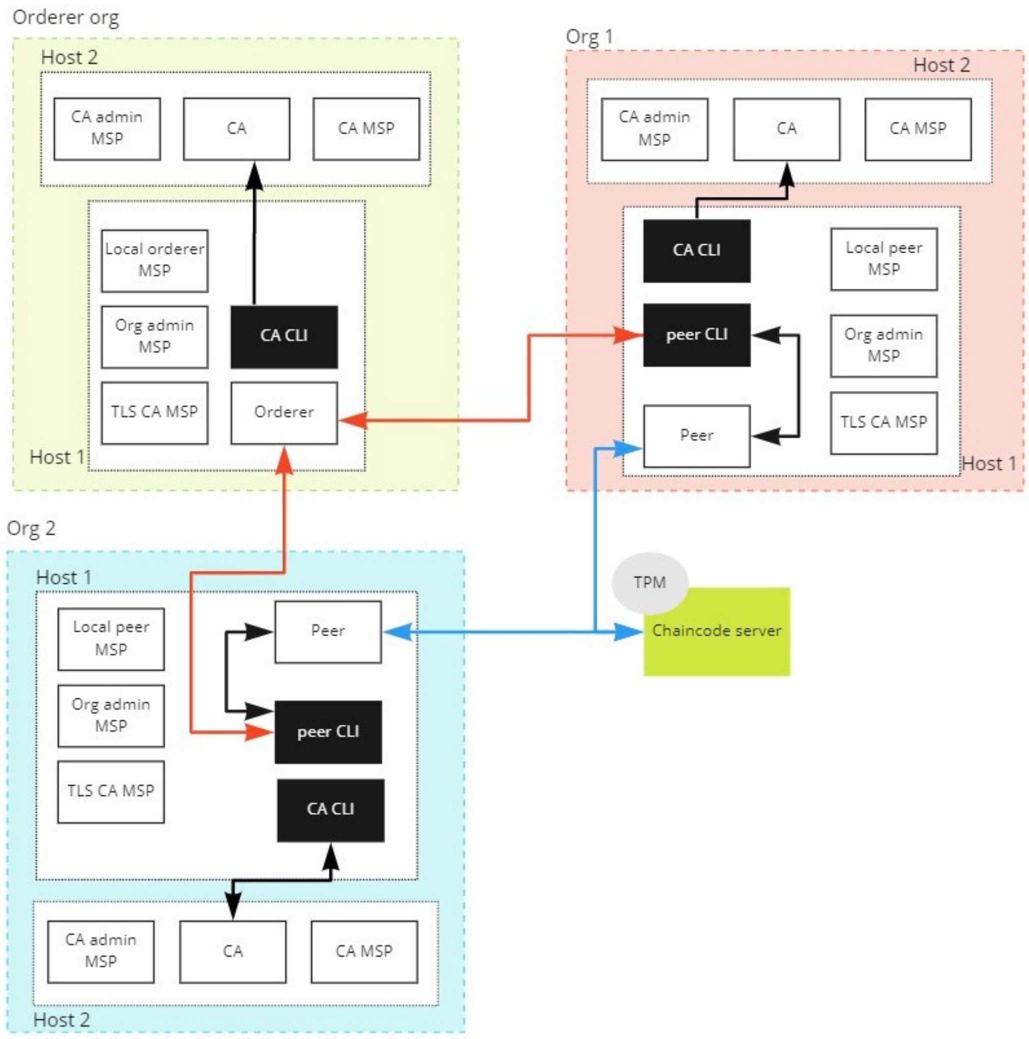


Figure 3: Network topology

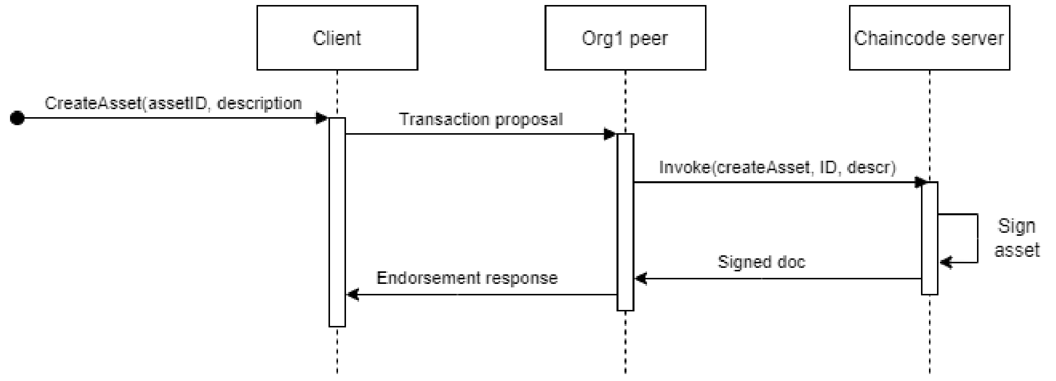


Figure 4: Sequence diagram describing create asset smart contract

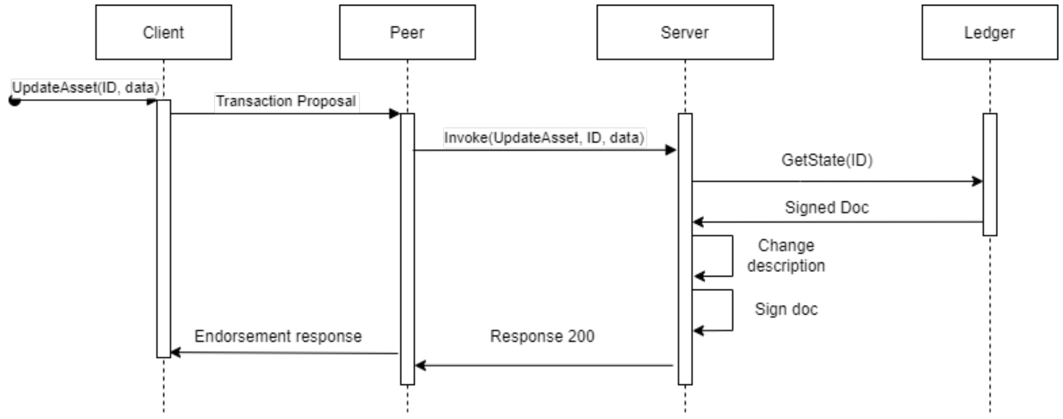


Figure 5: Sequence diagram describing update asset smart contract

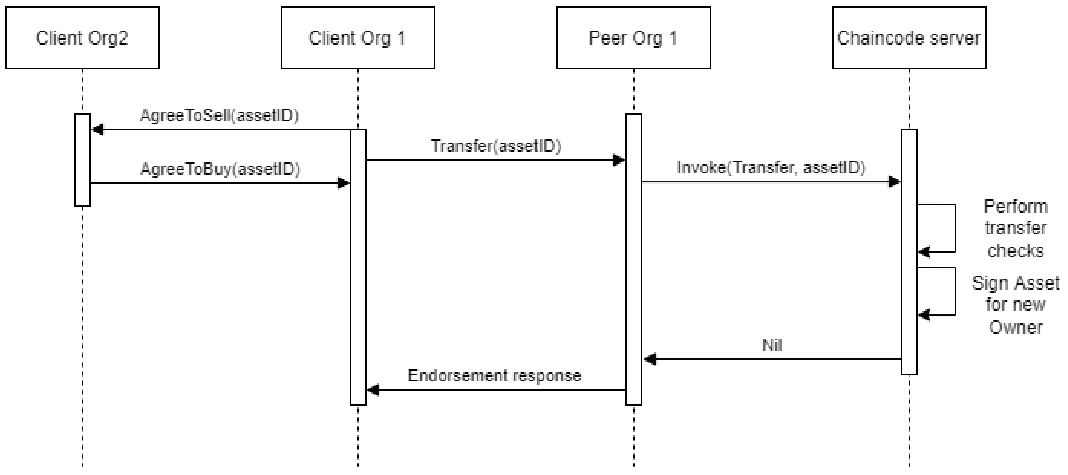


Figure 6: Sequence diagram describing the transfer smart contract

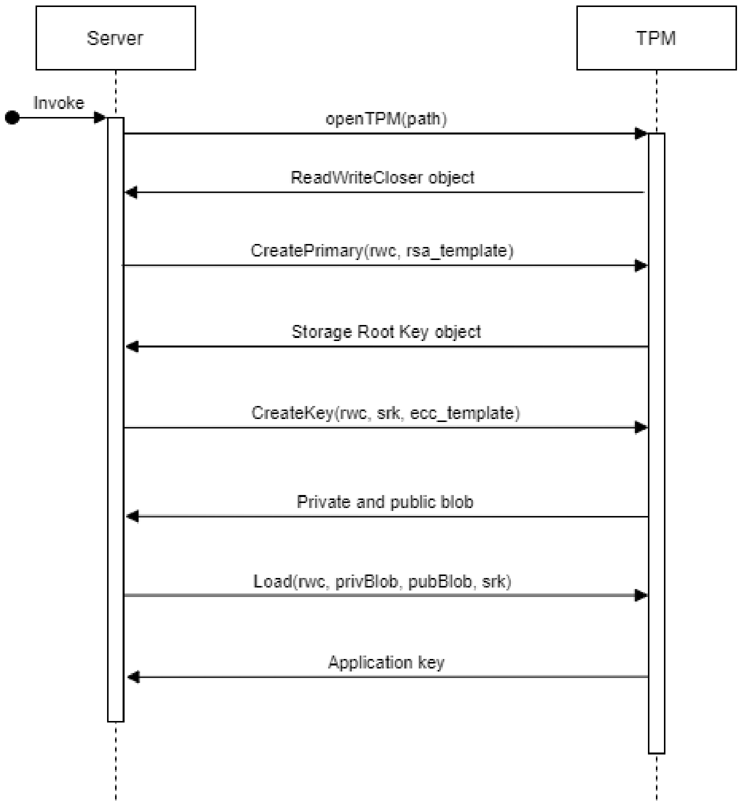
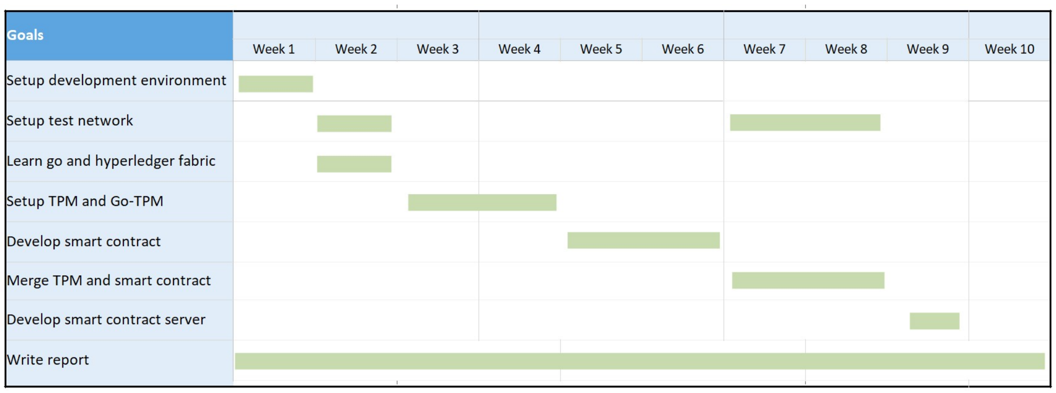


Figure 7: Sequence diagram describes the creation of the signing key



Goals	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10
Setup development environment	🟩									
Setup test network		🟩					🟩	🟩		
Learn go and hyperledger fabric		🟩								
Setup TPM and Go-TPM			🟩	🟩						
Develop smart contract					🟩	🟩				
Merge TPM and smart contract							🟩	🟩		
Develop smart contract server									🟩	
Write report	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

Figure 8: A technical roadmap

Name	Send Rate (TPs)	Max Latency (s)	Min latency (s)	Avg Latency (s)	Throughput (TPS)
10 transactions	128.2	1.29	1.21	1.25	7.8
20 transactions	281.7	2.19	2.05	2.13	9.1
50 transactions	320.5	5.18	4.84	5.00	9.6
80 transactions	318.7	8.30	7.71	8.04	9.5

Table 1: Create asset smart contract data