Creating a TPM based smart contract for the Medical Supply Chain in Hyperledger Fabric

Author: Kevin Nanhekhan[1],
Responsible Professor & Supervisor: Kaitai Liang[2]
EEMCS, Delft University of Technology, The Netherlands
[1]k.r.nanhekhan@student.tudelft.nl, [2]kaitai.liang@tudelft.nl

January 23, 2022

Abstract

Smart contracts play an important role within the blockchain by ensuring that valid transactions are being recorded. However, there are critical concerns regarding the security and privacy of data within these blockchain applications. This research provides information on how the integration of the Trusted Platform Module can achieve more security in a blockchain application built on Hyperledger Fabric. Despite the implemented prototype and Trusted Platform Module functions working separately, combining the two into one working prototype was not successful. Tests analysis has been performed showing that the prototype performs worse due to higher latencies and has no security vulnerabilities. However, these are not conclusive statements, as not only the prototype did not work fully but the analysis tools were lacking. Recommendations have been made to use the described process of this research for future research and also the development of more analysis tools.

1 Introduction

Blockchain is a term nowadays often associated with cryptocurrencies like Bitcoin and Ethereum [23]. However, it also has an immense impact on various big sectors like Finance, Supply Chain and Healthcare [9]. It is defined as a decentralised, distributed database consisting of blocks linked together through cryptography that records all transactions where the integrity is continuously verified [23, 32]. In this way, there is no need for a central entity as the security is guaranteed through the strength and computing power of the entire network participating in the blockchain. **Smart contracts** play an important part in blockchain transactions. They are defined as event-driven programs executed and enforced by the peers (network participants) containing a public interface that handles the relevant events [18, 22]. The transactions with the proper payload invoke these interfaces where all valid transactions are recorded on the blockchain. In comparison to conventional contracts, smart contracts have the advantages of reducing risks, cutting down administration and service costs and improving the efficiency in business processes [32].

The **Medical Supply Chain** is an example that falls under the aforementioned big sectors with already working applications that benefit from the use of blockchain technology. A few examples for this are *HealthVerity* [8], *the Mediledger Network* [7] and *Medicalchain* [24]. However there are critical concerns regarding the practical deployment of blockchains which focus on security and privacy of data and computations within the applications [9, 22]. For example, malicious behaviour like frauds and attacks [32] can exploit the blockchain transactions and lead to leakage of millions of client records. To tackle these concerns, research is currently done on increasing the security of blockchains through integrating cryptographic functionality of **trusted hardware**. Trusted

hardware can be seen as hardware mechanisms that use cryptographic algorithms to securely attest aspects of the system configuration [21]. This is to detect and respond to tampering, provide protection of the system storage and also ensure the integrity of the software.

Related works. While not abundant, various research has already been done on this topic of integrating trusted hardware with smart contracts by taking a critical look at the challenges and issues [3, 4, 5, 17]. Many of these articles, journals and papers focus on the problems but these do not necessarily go into the fine details of how the integration of the trusted hardware's functionality with the smart contracts has been done. Additionally, as this is still a niche topic, the results and measurements on the performance and security are bound to be limited.

Contribution. To address this lack of knowledge and data in the literature, it is thus not only important for this research to look at only the theoretical side by doing a literature review but also the practical side by creating a simple prototype. For this prototype, a basic Medical Supply Chain will be implemented, because as mentioned before a lot of extra security can be added. To limit the scope of the research, in terms of the various trusted hardware that can be used, only one was chosen. The options of trusted hardware being: *Physical Unclonable Function* [16], *Intel Software Guard Extensions* [20], *Device Identifier Composition Engine* [28] and *Trusted Platform Module* [29]. Here **Trusted Platform Module** also known as **TPM** was chosen, with the reason being that nowadays most personal computers and notebook manufacturers use it in their products [10]. In addition, there is a lot of documentation regarding TPM readily available online. Thus with this all in mind, the main focus of this research is

"How to use the trusted hardware TPM to protect the execution of smart contracts?"

To make answering the main question easier, it will be broken up into the following sub-questions:

1. How does the execution of a smart contract within a ledger work?
2. What are the different functionalities of a TPM?
3. How can the functionalities of a TPM be integrated with a smart contract?
4. How can the performance of the secure smart contract be assessed?

Organisation. The structure of the paper is as follows. In section 2, the methodology focusing on how the research has been conducted will be explained. Section 3 provides a brief overview of the blockchain framework used and the TPM. How the prototype has been designed, will be illustrated in section 4 where the actual implementation will be described in section 5. The results based on certain performance metrics will be stated in section 6 where also discussion follows from. The responsible research in section 7 will reflect on the ethical aspects of the research and also the reproducibility. Finally, the paper ends in section 8 with the conclusions of this research and some future recommendations.

2 Methodology

Before focusing on the aims set in the previous section, this section provides an explanation of what has been done and also elaborates on the various choices made.

2.1 Literature review

Before starting the research, a Technology Roadmap has been created on the various stages of the research. This has been adjusted each week resulting in the final version which can be found

in Appendix A. As seen from this, the initial weeks of this research focused on doing literary research to gain a better understanding of the subject. To gather the various literature (journals/papers/articles), the following search terms were mainly used: *'Smart contracts', 'Trusted hardware', 'TPM'* and *'Trusted computing'*. Search engine Google Scholar, as well as journal databases like Springer, Science Direct, ACM and IEEE were used for this purpose. Only literature from within the period 2005-2021 were used as smart contracts and trusted hardware are more well-researched during this time period.

2.2 Coding Tools

Besides the theoretical aspect, there was also a focus on various options regarding the prototype. To start off, the prototype needed to be run on a blockchain framework so therefore **Hyperledger Fabric** version 2.4.1 [13] was used as a basis. Hyperledger Fabric utilises **Docker** [19] to run the different components in its application, so for this research version 20.10.7 is used. For this project, a laptop with Windows 10 operating system was used. As the Hyperledger fabric is Linux based, the virtual machine **Hyper-V** [26] was used as it is a native Windows hypervisor, offering a quick create option for Ubuntu 20.04 and also has an option for using a virtualised TPM. This is opposed to using Windows Subsystem for Linux which, despite also supporting virtualisation and Ubuntu 20.04, does not support interaction with the TPM. Hyperledger Fabric has support for multiple programming languages which are: Go, Javascript, Java and Typescript. The framework was initially built on Go and uses it as the default language where a lot of code examples are provided in Hyperledger Fabric's documentation. Hence the choice fell on programming the application and smart contract in **Go** (version 1.17.6). With this choice comes also the use of the open-source Go library **Go-TPM** (version 0.3.2) [15] for simple interaction with the TPM. Lastly, besides using the terminal to monitor the blockchain network, **Hyperledger Explorer** [12] was integrated for its simple Graphical User Interface.

2.3 Benchmark Tools

Creating a working prototype on its own is not sufficient as research results, hence empirical data needed to be gathered. For measuring the performance, **Hyperledger Caliper** (version 2.2) [11] was used as it allows for the use of custom benchmark tests. Measuring the security of the prototype proved to be a lot more difficult because of the limited options. In spite of the fact various analysis tools exist for measuring blockchains, these are used for different chaincode programming languages (like EVM bytecode or Solidity) [25]. Thus the options were restricted to the static analysis tools *Revive-cc* [27], *Chaincode Analyzer* [14] and *Chaincode Scanner* [6]. Nevertheless, Chaincode Scanner at the time of this research was no longer available and the Chaincode Analyzer did not manage to run due to outdated dependencies which left **Revive-cc** as the sole option.

3 Background

In this section, a brief overview of Hyperledger Fabric will be given as well as the different TPM functionalities explored based on the gathered literature. Additionally, this section also serves to answer the subquestions in their respective subsections.

3.1 Hyperledger Fabric

Components. Hyperledger Fabric is known as a permissioned blockchain where users need prior approval and be authenticated before using the network, opposed to permissionless blockchains [13]. Also, the platform is an open-source modular blockchain framework that allows for pluggable modular components ranging from consensus algorithms to cryptographic libraries. The program

Docker is used to run each component in its own container environment. The most important components forming the foundation are:

- **Channel**: Partition of the Hyperledger Fabric network where only joined Peers can interact. Data shared through the channel is ensured to be isolated and confidential.
- **Ledger**: Series of blocks on which transaction details are recorded. Each channel has one ledger where all joined Peers keep a copy of the ledger.
- **Client**: Application using the Hyperledger Fabric SDK that acts on behalf of the user by invoking a transaction to the Hyperledger Fabric network.
- **Committing Peer**: Node updates the local ledger based on the validity of blocks sent by the Orderer Peer.
- **Endorser Peer**: Node that receives transaction proposals from the Client, maintains a ledger and runs the chaincode containers to do read/write operations to the ledger.
- **Anchor Peer**: Node which other peers from different organisations can discover and communicate with.
- **Orderer Peer**: Node that endorses transactions, orders them and distributes them to all committed peers on a channel.
- **Chaincode**: Technical container of smart contracts installed on each peer and invoked by an Orderer on specific channels. Chaincode is most often used as a synonym for smart contracts.
- **Certificate Authority**: A trusted entity that is responsible for issuing and revoking Public Key Infrastructure-based certificates. These certificates are used to represent the different entities within the Hyperledger Fabric network.

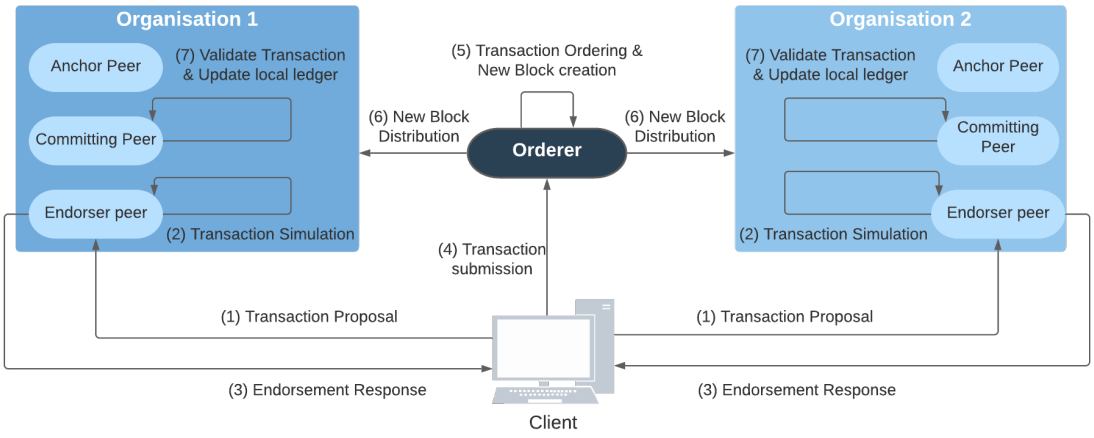

Figure 1: Sequence Diagram of the Transaction flow within a Hyperledger Fabric channel

Transaction Flow. Hyperledger Fabric uses endorsement policies with a majority vote to ensure which chaincode can be executed and endorsed within the channel of a network [13, 31]. After that, the transaction flow can start (see figure 1) where a user tries to invoke a transaction through the Client. A transaction proposal is sent to the different Endorser peers who validate the transaction by executing the installed Chaincode which generates a response that is endorsed with the peer's certificate and sent back. Having received an approved transaction, the Client sends this to the Orderer peer. The Orderer properly orders all transactions and sends a new block to all the organisations in the channel. Each transaction is then verified whether these are signed by the appropriate Endorser peers. If valid, then the committing peers update their local copy of the ledger.

3.2 Functionalities of the TPM

The TPM has various functionalities from which a distinction can be made. These are **Secure storage**, **Platform measurement and reporting** and **Platform authentication** [30].

Secure storage. One can store data securely by using keys generated by the TPM [10]. The TPM consists of a *Hashing engine, RSA key generation, RSA signing and encryption* and also a *Random number generator* which can be used in the generation of keys [30, 2].

Platform measurement and reporting. Platform Configuration Registers (PCR) play a vital role. These are registers in the TPM that can compute hashes or also known as measurements of a component and store it [2]. These measurements can be extended which means that the current PCR value is the accumulation of all the measurements it has stored. The PCR value can be used as an authentication signal by for example assuring the state of a platform to a third party or ensuring data is only accessible to authorised software [30].

Platform authentication. At the time of manufacturing, a special unique public/private key pair called the Endorsement Key (EK) has been set. Additionally, when taking ownership of a TPM, another special key is generated called the Storage Root Key (SRT). The EK and SRT are long-term keys and since these are identifiable are not meant for the signing but only for encrypting [30]. In order to do signing of application-specific data with the TPM, Application Identity Keys (AIK) can be generated. AIK 's opposed to SRT and EK can be proven to be generated by a TPM but not which TPM to ensure user privacy [2]. This is done through the use of privacy certification authority (privacy CA) where EK can be used to prove an AIK is from a TPM without revealing which TPM. In addition, another way is by using Direct Anonymous Attestation (DAA) which is based on group signatures. DAA has the advantage over privacy CA by allowing the key creator to choose whether EK and AIK have any association or their relation is completely anonymous.

TPM versions. At the current moment a modern version called TPM 2.0 is already in use. Tpm 2.0 shares the same functionalities as its predecessor TPM 1.2 except with some additional functionality and improvements [2] as listed below in table 1:

Functionality	TPM 1.2	TPM 2.0
Algorithm agility	Sha-1 hash algorithm	Virtually any hashing algorithm
Authorisation	Needs owner authorisation	Uses clever policy decision to decide authorisation
Key Loading	time-consuming private-key decryption	quick symmetric-key operation
Fragility PCRs	Brittle PCR because sealing to PCR value approved by particular PCR value	Non-Brittle PCR because of sealing to PCR value approved by a particular signer
Management	Difficult to give authorisation for different roles	Built-in different authorisations, policies and hierarchies
Identifying Resources	By handle where the user could be tricked into authorising a different action	By name of which the resource is cryptographically bound to

Table 1: Differences functionality between TPM 1.2 and TPM 2.0.

4 Experimental Design

The design of the prototype and how the TPM is to be integrated will be illustrated in this section. The third subquestion will be partially answered here in a theoretical manner.

4.1 Designing the Medical Supply Chain

Concept For the prototype, a simplified Medical Supply Chain was created. This design is based on the *Commercial Paper* example from the *fabric-samples* repository (provided by Hyperledger Fabric), as these contain many similarities between how the application and chaincode work. The idea for the Medical Supply Chain was to have a pharmacy network called *MedStore*, which provides authorised participants among others with the important options to issue, request and approve medical supplies. MedStore network can for example be used throughout the country where every channel created could represent a city where multiple participants reside. There are two distinct types of participants, namely, *Customers* and *Regulators*. Customers can perform various actions such as checking all available medical supply, request and cancel a request for a medical supply and also checking their own history. Regulators on the other hand can issue a new medical supply, approve or reject a customer's request and also check the transaction history in various manners. The duty of the regulators is thus to not only (re)supply the stock of medical supplies but also to act as an administrative figure within the network by monitoring the transactions and taking the necessary actions. The number of peers participating in the network has been kept to two. The prototype could be extended to include more peers such as private organisations that handle the same as Customers. However, the general structure has been kept simple as the prototype just serves as a basis for integrating the TPM.

Folder structure. As seen in figure 2, the prototype is structured as follows. On the top layer, *customers* and *regulators* have each their own respective and similarly-formatted folders containing the application file, the smart contract (chaincode) and lastly a configuration folder with a setup file (shell script) for connecting to the network. Both chaincode folders contain the exact same files to comply with the endorsement policies Hyperledger Fabric uses. The setup files use the **peer lifecycle chaincode** [13] to automatically package, install, approve and allow invocation of the chaincode on a channel. When running the application, the peers will be enrolled where a wallet and keystore folder will be generated in the organisation folder containing the peers' identity used for invoking the smart contract. Besides the organisation folders, there are the *networkDeploy* script for starting up the test network (creating the peers, orderer node and necessary certificates) and *networkClean* script for stopping the network. Lastly, there are the folders *explorer* for starting Hyperledger Explorer to monitor the network and *caliper* for running Hyperledger Caliper's benchmark tests.

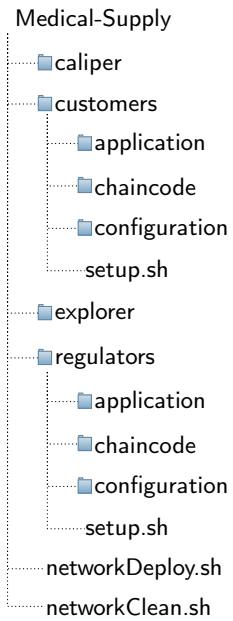

Figure 2: Folder structure

4.2 Designing the TPM's integration

As mentioned in section 3, Hyperledger Fabric is known as a prominent enterprise blockchain platform where it is been proven to be very secure against attacks from outside the network [5, 1]. Even so, that does not mean it does not have its share in security limitations, for example against peers that have already joined a channel in the network. When such a peer acts malicious and

works together with adversaries outside the network, confidential data on the ledger is in danger. This is known as a Wormhole Attack.

Checksum. To mitigate a small part of this problem, a checksum will be added as a field to the medical supply object. This checksum is a hash of the object's information and will only be initiated at the time a medical supply has been issued by a Regulator. Whenever a transaction occurs that affects such a medical supply, before being submitted the checksum will be used to verify the contents have not been tampered with by comparing it to a recalculated checksum.

Anonymity Furthermore, the identities of the users can be known, therefore it is best for them to be anonymous. Hyperledger Fabric already provides a built-in function for the smart contract to get a hashed identity of the smart contract invoker (peer). Nonetheless, the function is not very useful for the prototype as certain functions allow for its users to provide a name themselves (for example for using a pseudo name). Here it is thus not viable to expect the customers and regulators to fill in a hashed version of their enrolled identity. So instead of purely relying on the Hyperledger Fabric's hashed identity, the TPM is going to be used to hash the provided names within the smart contract. As other blockchain applications might not have access to the in-built function like Hyperledger Fabric does or want to authenticate their users in a different way, this can serve as an example to show how TPM can be used as an alternative option.

Authentication. The use of *Random number generators* in smart contracts have been proven to be problematic in certain applications when run by different peers [31]. However, an advantage can be taken from that non-deterministic nature for the purpose of generating unique keys. As users provide their names to invoke certain transactions, it is best to add an additional authentication method to avoid the risk of impersonation. Here the idea is for the user to first invoke the smart contract with the name they want to use to get a unique key. The key and name are stored on the ledger, where every endorsed transaction requires this combination before being submitted. This way a malicious peer can not impersonate a different user as it does not have a unique key.

5 Experimental Work

This section will expand upon how the prototype was implemented and also show the function implementation of the TPM. This section will finish answering the third subquestion from a practical aspect. Not every aspect of the implementation will be discussed, so all code has been made available at the following Github repository: `https://github.com/Kevin-RN/medical-supply`

5.1 Implementing the Medical Supply Chain

Algorithm 1 Pseudocode of the Medical Supply class

```
1: class MEDICAL SUPPLY
2:     medName : STRING                    ▷ medicine name
3:     medNumber : STRING                  ▷ medicine number
4:     disease: STRING                     ▷ disease to cure
5:     expiration: STRING                  ▷ expiration date
6:     price: STRING                       ▷ medicine price
7:     holder: STRING                      ▷ current holder
8:     state: STRING                       ▷ medicine status
9:     class: STRING                       ▷ unique namespace
10:    key: STRING                         ▷ unique ledger key
11: end class
```

Object class The medical supply class, as seen in Algorithm 1 is a core part of the prototype's transactions. *State* field is used as enumeration of the current status of the medical supply (being either Available, Requested or Send). *Class* field is used for a unique namespace (e.g. org.medstore.medicalsupply) within the chaincode in case multiple other classes are implemented. *Key* field is the most crucial field since that is used to store and query the medical supply from the ledger. A unique key is generated by combining the string "MedStore" with the medicine name and number (e.g. MedStore:Aspirin:00001). Normally using the in-built query function with empty parameters would allow for querying multiple entries on the ledger. However, that does not work at the current version of Hyperledger fabric as null characters (`\u0000`) are added in between composite keys resulting in no entries retrieved. Hence the string "MedStore" has been added so all medical supplies share part of the key which makes querying multiple entries possible.

Algorithm 2 Pseudocode of the Issue function before TPM integration

```
1: function Issue(ctx, medname, mednumber, disease, expiration, price)
2:     error = Check whether invoker is from the Regulators organisation
3:     if error ≠ NULL then                                  ▷ Check if no errors occurred
4:         return error that invoker is not authorised
5:     end if
6:     medicine = Create new medicine from provided arguments
7:     Set medicine to Available
8:     Add medicine to the ledger through ctx                ▷ context transaction handler
9:     if error ≠ NULL then
10:        return error that adding medicine did not work
11:    end if
12:    return medicine
13: end function
```

Chaincode Functionality. Various functions have been implemented for the smart contract, these have been listed in Appendix B. A small subset of these functions is only meant for Regulators. So whenever these functions are invoked, the organisation of the transaction invoker (peers), retrieved through an in-built function of Hyperledger Fabric, will be verified. An example can be seen in the Issue function of figure 1. Another thing that can be seen from this figure is the use of the Context Transaction Handler. This is used in Hyperledger Fabric to submit transactions to and from the ledger. Querying entries from the ledger could only be done using keys and not on fields of stored objects. Hence functions implemented for retrieving multiple entries had to first get all entries from the ledger and then perform a filter based on criteria (e.g. status is available) through the use of for loops.

5.2 Integration of the TPM into the Medical Supply Chain

A few remarks are in order before going into the specifics of the TPM implementation. Trying to access TPM can only be done after taking ownership of one of the two folders it utilises. This is done with the `sudo chown <user> <folder>`. The folder *'/dev/tpm0'* can only be accessed by one at a time whereas *'/dev/tpmrm0'* by multiple. For the prototype it is expected multiple peers will call transactions concurrently hence accessing the TPM is done through the *'/dev/tpmrm0'*.

Hashing. Through the Go-TPM library, various TPM functions are available. One of them is a hash function, which as seen in figure 3 hashes the input using the SHA-256 algorithm. To use the checksum, two functions were added inside the Medical Supply class. These are used for initialising the checksum and verifying the checksum. Initialising the checksum is only used for functions such as *InitLedger* and *Issue* (see Appendix B), where a new Medical Supply object has been created. The verifying checksum function is used whenever a medical supply is retrieved to check if it has not been altered. In the case medical supply are not valid, they will not be used (not returned or change state) during the transaction but still remain on the ledger. Since the TPM

Algorithm 3 Pseudocode of the tpmHash function

```
function tpmHash(input)
    update input to be all lowercase
    rwc, err = open tpm2 on the /dev/tpmrm0 path
    if error ≠ NULL then
        return error that tpm couldn't be opened
    end if
    dataToHash = convert input string to byte array
    hashdigest, hashError = use the tpm2 hash function using SHA-256 algorithm on the dataToHash
    if hashError ≠ NULL then
        return error occurred while hashing
    end if
    return hashDigest
end function
```

hash function is already in place it can be reused for hashing the user name to provide anonymity. Thus besides where a checksum is needed, the hash function will also be called when a customer invokes a transaction where a user name is required, like requesting a medical supply.

Algorithm 4 Pseudocode of the tpmKey function

```
function tpmKey(input)
    rwc, err = open tpm2 on the /dev/tpmrm0 path
    if error ≠ NULL then
        return error that tpm couldn't be opened
    end if
    randByte, error = generate random bytes using tpm2
    if error ≠ NULL then
        return error that generating random bytes failed
    end if
    parentHandle, err = generate primary key using random bytes
    if error ≠ NULL then
        return error that creating primary key failed
    end if
    flush parentHandle from tpm2 to free memory
    privateKey, publicKey, error = Create key using parentHandle and randByte
    if error ≠ NULL then
        return error that creating key failed
    end if
    createdKey = load privatekey, publickey and randByte as object in tpm2
    dataBlob = byte array
    initVector = byte array
    tpmKey, error = use Encrypt Symmetric on createdKey with dataBlob, initVector and randByte
    if error ≠ NULL then
        return error that creating symmetric encryption key failed
    end if
    return string of tpmkey
end function
```

TPMAuth. A new class called TPMAuth was created which works similar to the medical supply class. The key used for the ledger is a combination of the "TPMAUTH" string and the user name. With this new class comes an extra field tpm key for each smart contract function. The user now not only provides their username but also a tpm key to authenticate themselves first before being able to invoke these functions. For this purpose the function *TPMKeyGen* (see Appendix B) was created which when invoked will create a TPMAuth object based on a tpm key (generated by function seen in figure 4) and a hash of the user-provided string. The function returns the tpm key to the user which needs to be stored to use later on. For convenience, the implemented application automatically stores and reads this tpm key from a temporary file. Every stored hash user name is unique so trying to create a new TPMAuth based on a previously provided user name will result in an error. Furthermore retrieving from the ledger will only be done to verify whether the provided tpm key and username are valid before invoking the smart contract function. This means a user can not request to see his stored tpm key on the ledger as that would be exploitable.

5.3 Limitations and problems of the Prototype

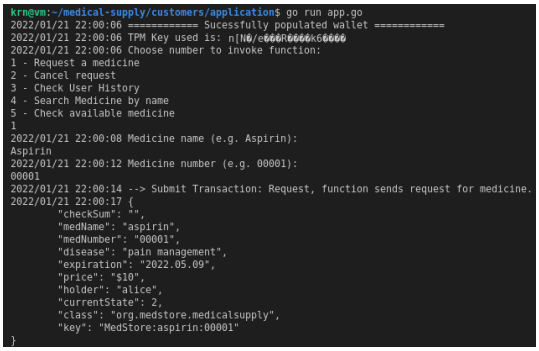

Figure 3: Working example of only the prototype

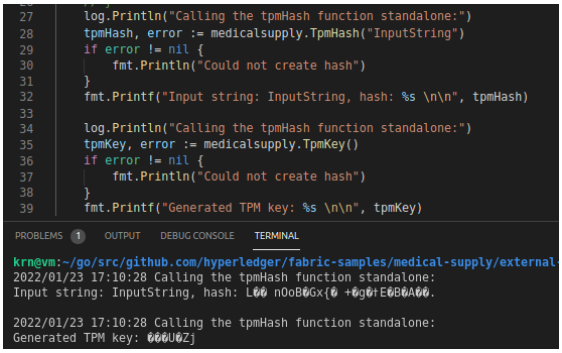

Figure 4: Standalone example to show TPM functions

Working prototype. As seen from figures 3 and 4, the prototype and the TPM functions work separately from one another. In spite of that, attempting to combine the TPM functions into the prototype proved to be quite problematic. The prototype was not able to fully work with TPM integrated due to Docker, which Hyperledger Fabric uses to run its components, not being able to access the TPM folder location. Docker cannot access the folder location as it runs in its own environment separate from the local system. A solution was attempted in two ways, by mounting the folder location to the docker containers and by using chaincode as an external service instead of installed on each peer. However, due to lack of time and knowledge on these, this was not accomplished within the time frame of this research.

Chaincode limitations. Besides not having a fully working prototype, as mentioned in previous sections, the prototype is not extensive enough to be fully useful in a practical setting as the smart contract logic is flawed. For example, the transactions return entire objects with their unique ledger key to the peers. A different example would be that transaction input was not checked to be the correct format. Random text on the expiration field could be filled in when issuing a new medical supply. For these reasons, this prototype should not be considered for practical usage. Still, for the context of this research it is adequate enough, as the purpose of this research is to show how a blockchain application can become more secure through having TPM integrated.

Dependency limitations. The application is not very optimal in not only its implementation but also the dependencies. The current version of Hyperledger Fabric is still undergoing changes for improvements. This also goes for the Go-TPM and TPM2-Tools library for interacting with the TPM as the entire spec of TPM 1.2 and TPM 2.0 spec are not yet completely implemented. Lastly, the helper libraries which have been utilised can perform worse than other libraries or re-implementing the functions themselves.

6 Experimental outcome

This section presents the results obtained from performance and security analysis. Each subsection of these analyses will be accompanied by a discussion. This section will answer the fourth and final subquestion.

6.1 Performance analysis

Performance test settings. Before utilising Hyperledger Caliper [11] for conducting a performance analysis on the prototype, custom benchmark tests had to be created. These tests are set up to initialise the ledger (adding the medical supply), perform the transaction (e.g. requests) and afterwards clear the ledger (deleting the medical supply). All transactions are set to run for a total length of a minute each and have five workers run concurrently invoking the transactions. By setting these with a time limit, information can be gathered on the latency and the number of transactions invoked. Hyperledger Caliper has been run a total of five times where the averages of the measurements were taken and put in table 2. Average results were taken for accuracy reasons, where the total amount of testing runs to be used was done up to five, as otherwise it would become too computationally intensive for the machine it was run on. It should be noted that during testing, due to high load the transactions querying multiple entries from the ledger could not manage to finish in time resulting in time-out errors. Re-running these individually, instead of all at once, was done for it to resolve this issue.

Function Name	Amount invoked	Send Rate	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Through put
Issue	1,120	18.8	1.14	0.10	0.35	18.8
Request	21,181	358.0	0.08	0.01	0.02	357.0
CancelRequest	22,095	374.4	0.08	0.01	0.02	374.4
ApproveRequest	19,664	333.1	0.11	0.01	0.02	333.1
RejectRequest	22,357	378.8	0.09	0.01	0.02	378.7
CheckHistory	3,549	60.1	0.23	0.02	0.10	60.0
CheckUserHistory	2,945	49.8	0.40	0.03	0.12	49.7
CheckAvailableMedicine	3,270	55.2	0.36	0.03	0.11	55.2
CheckRequestedMedicine	3,181	53.7	0.29	0.03	0.13	53.5
SearchMedicineByName	5,260	53.6	0.22	0.03	0.07	53.3
ChangeStatus	21,242	359.9	0.08	0.01	0.02	359.9
ChangeHolder	21,102	357.6	0.07	0.01	0.02	357.5

Table 2: Average performance results over five runs before the TPM integration

Prototype before TPM changes. From table 2 it can be seen that each transaction is relatively fast with the highest latency being around an average of 6.39 seconds. A distinction can be made between transactions that have a high amount of being invoked with lower latency and transactions that are the opposite with a low amount of being invoked and higher latency. This is to be expected as the latter is the type of transaction that queries all entries from the ledger and filters on certain criteria (e.g. on only available medical supply). The Issue transaction is the only exception as it has the worst performance compared to other transactions that handle a single medical supply object or even multiple. This can be reasoned to be the fact that other transactions only query an existing entry from the ledger and update it, whereas the Issue transaction creates one from scratch and adds it to the ledger. Here Issue transaction would take more time to compute which causes a delay in its latency, which on its turn causes less to be invoked within the test time limit.

Function Name	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
TPM Hash	5.98	4.63	4.72	6.16	5.97	8.69	4.86	4.78	4.20	5.19
TPM Key Generation	153.88	165.92	150.55	144.34	145.69	148.48	144.69	157.66	145.57	149.49

Table 3: Performance TPM functions over ten runs measured in miliseconds

Function Name	Max Latency (s)	Min Latency (s)	Avg Latency (s)
Issue	1.34	0.25	0.54
Request	0.37	0.16	0.17
CancelRequest	0.29	0.16	0.17
ApproveRequest	0.29	0.16	0.17
RejectRequest	0.17	0.16	0.17
CheckHistory	0.45	0.17	0.25
CheckUserHistory	0.61	0.19	0.29
CheckAvailableMedicine	0.45	0.16	0.27
CheckRequestedMedicine	0.51	0.17	0.27
SearchMedicineByName	0.61	0.23	0.31
ChangeStatus	0.25	0.16	0.02
ChangeHolder	0.26	0.16	0.02

Table 4: Rough estimation of the average performance results over five runs with TPM integration

Rough estimation with TPM. As mentioned in section 5, the prototype could not run with TPM integrated where performance testing would be impossible. Nonetheless, the performance of the prototype without the actual call to the TPM but with TPMAuth class and checksum in place can be measured. A hard-coded result will be returned instead of utilising TPM functions for creating the hash and key. As seen from table 3, the run-time over ten runs can be measured from the TPM hashing and key generating functions on their own. Taking the total of these results and adding them to the various latencies provides a rough estimation of how well the prototype with TPM would perform. This has been presented in table 4. The rough estimation does not cover the number of transactions invoked, the send rate and the throughput as these are calculated differently. Assumed is however that these values have a negative correlation with the various latencies based on table 2. With this assumption in mind, it can be seen that compared to table 2 the prototype with TPM would perform slightly worse. It is worth bearing in mind that the transactions which query multiple entries will have a bigger latency. A checksum needs to be computed for each entry which the rough estimation does not take into account since the average run-time for the TPM hash was only added once.

6.2 Security analysis

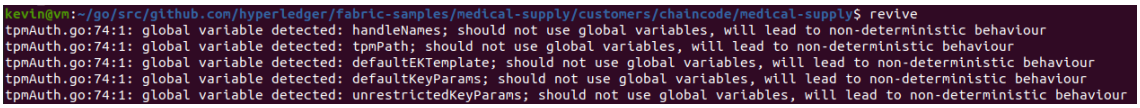

Figure 5: Revive-cc vulnerability results on the terminal

Revive-cc is a statistical analysis tool, so running the chaincode was not necessary to measure security. Before the integration of TPM, the Revive-cc [27] showed no vulnerabilities in the chaincode. After the integration of TPM on the other hand, running Revive-cc showed several vulnerabilities on the chaincode (see figure 5). These vulnerabilities concern the use of global variables and were

quickly resolved by changing these to local variables. Rerunning Revive-cc resulted in no vulnerabilities shown anymore. Although the prototype shows no difference before and after the TPM integration, this does not mean it can be used as conclusive evidence regarding the use of the TPM for security. Revive-cc can only detect up to six kinds of vulnerabilities: *Blacklisted chaincode imports, Global state variables, Goroutines, Phantom read of the ledger, Range over map, Read after write*. Hence quite limited in its capability to determine security. Moreover, it is a static analysis tool meant just for scanning files. Thus it cannot perform security analysis at run-time. Lastly, while it can be said that the prototype is somewhat secure against the few vulnerabilities it has been tested for, depending on a Revive-cc is not enough.

7 Responsible Research

As is part of most research, this section will cover the ethical aspects of the research and also look at the reproducibility of what has been done.

7.1 Ethical aspects

Ethical literature. All the studies used for this research come from scientific databases that provide peer-reviewed work. This selection of studies has been chosen based on keywords mentioned in section 2. The sources that are not from studies are either open-source libraries, frameworks documentation or official sites of the applications that have been used or mentioned. Every scientific insight has been referenced where their respective author and/or source is listed in the reference list.

Ethical code. The prototype implemented was done to show how trusted hardware like TPM can make a smart contract, used for example in a Medical Supply Chain, more secure. This is to inform blockchain developers of the security possibilities of trusted hardware. The prototype does not rely on real-time data, nor does it rely on private information as it was run on a local system. Even though pseudocode has been described, the prototype is just a simplified version that can not be used in a practical setting. Furthermore, this research poses minimal risk as the TPM integration follows a general format to mitigate security limitations that have already been described in other literature. Even more, these solutions are at the current moment still theoretical and not applied in practice. Therefore, this research will not affect current programs in place, by for example exploits of malicious users as these systems are more likely to be sophisticated.

7.2 Reproducibility

Fully reproducing the research from scratch could prove to be difficult, as a lot of errors were encountered in utilising Hyperledger Fabric and TPM. Not a lot of online documentation and forums covered these errors. Hence an attempt was made in this paper to describe every step of the process in a detailed yet concise manner. A brief overview of the Hyperledger Fabric as well as TPM functions has been provided to inform the reader on the core topics of this research. Furthermore, the research has been described in such a way that the reader can follow along in the steps taken. A Technology Roadmap is included in Appendix A to show what task has been done on a per week basis. As seen in this report, an explanation of the code has been used to show a general way of how TPM functions can be utilised. This comes accompanied by code fragments written in pseudocode to avoid being language-specific. So readers with no experience with the programming language Go can follow along. Not all code has been described but it is fully available at the Github repository: `https://github.com/Kevin-RN/medical-supply`.

8 Conclusions and Future Work

Due to the presence of critical concerns regarding security and privacy, this paper presented the need for blockchain applications to be more secure and how trusted hardware can fill in that gap by being integrated with smart contracts. Thus, the main focus was to research how trusted hardware like Trusted Platform Module (TPM) can protect the execution of smart contracts. Preparation has been done for this research by doing a literature review and exploring various coding and benchmark tools for a prototype. For the prototype, a simple Medical Supply Chain prototype was created in the blockchain framework Hyperledger Fabric. The cryptographic functionality of the TPM was used to make the prototype more secure. For the design, details have been given on how TPM is to be used to mitigate the problem of a Wormhole Attack. While the prototype and TPM functions managed to work separately, the goal of combining the two to create a working prototype with TPM integrated did not succeed. This is due to Docker not being able to access the TPM as it is separate from the local machine. Several attempts have been tried but were not accomplished within the time frame of this research.

Despite this major problem, an attempt was made in gaining some results. The performance result is that the prototype with TPM integrated will perform worse where transactions that query multiple entries will have the highest latencies. These results should be taken with a grain of salt as a rough estimation was used and not an exact measurement. Furthermore, it should be noted that these performance tests were run on a local Hyperledger Fabric network. A delay is therefore to be expected for similar implementations on a public network. As for the security analysis, no insightful results were to be gained due to the tool's limited capability. As no other tools are available for security analysis, no conclusive statements can be made on the security effectiveness of TPM for this research.

Future Work. While a working prototype and test results to back that up did not manage to succeed, this research still managed to partially achieve the main goal by describing the process involved. In future work, this process can be helpful for exploring the possible options of solving the main problem by adjusting either Docker or using chaincode as a external service. Another recommendation that can be made for future work involves the creation of more analysis tools. This is due to the lack of tools available and compatible with Hyperledger Fabric. With this comes the benefit of more research being able to evaluate the benefits of having TPM but also other types of trusted hardware integrated. Lastly comes the recommendation that different trusted hardware usages should be explored for improving security. In conclusion, if the aforementioned recommendations are to be followed, that would not only benefit similar research but also practical deployment of blockchain applications.

References

[1] Nitish Andola, Raghav, Manas Gogoi, S. Venkatesan, and Shekhar Verma. Vulnerabilities on hyperledger fabric. *Pervasive and Mobile Computing*, 59:101050, 2019.

[2] Will Arthur and David Challener. *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security*. Apress, USA, 1st edition, 2015.

[3] Zijian Bao, Qinghao Wang, Wenbo Shi, Lei Wang, Hong Lei, and Bangdao Chen. When blockchain meets sgx: An overview, challenges, and open issues. *IEEE Access*, 8:170404–170420, 2020.

[4] Marcus Brandenburger and Christian Cachin. Challenges for combining smart contracts with trusted computing, 2018.

[5] Marcus Brandenburger, Christian Cachin, Rudiger Kapitza, and Alessandro Sorniotti. Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric, 2018.

[6] ChainSecurity. Chaincode scanner, 2020. Available: https://hyperchecker.chainsecurity.com/.

[7] Chronicled. Mediledger, 2019. Available: https://www.mediledger.com/.

[8] Curisium. Healthverity, 2022. Available: https://healthverity.com/.

[9] Batsayan Das, Srujana Kanchanapalli, and Vigneswaran R. Enhancing security and privacy of permissioned blockchain using intel sgx. 2020.

[10] Kenneth Ezirim, Wai Khoo, George Koumantaris, Raymond Law, and Irippuge Perera. Trusted platform module – a survey. 2012.

[11] The Linux Foundation. Hyperledger caliper, 2022. Available: https://hyperledger.github.io/caliper/.

[12] The Linux Foundation. Hyperledger explorer, 2022. Available: https://www.hyperledger.org/use/explorer.

[13] The Linux Foundation. Hyperledger fabric main documentation., 2022. Available: https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html.

[14] FujitsuLaboratories. Chaincode analyzer, 2020. Available: https://github.com/FujitsuLaboratories/ChaincodeAnalyzer.

[15] Google. Go-tpm module, 2021. Available: https://pkg.go.dev/github.com/google/go-tpm.

[16] Jorge Guajardo. *Physical Unclonable Functions (PUFs)*, pages 929–934. Springer US, Boston, MA, 2011.

[17] Thomas Hardjono and Ned M. Smith. An attestation architecture for blockchain networks. *CoRR*, abs/2005.04293, 2020.

[18] Bin Hu, Zongyang Zhang, Jianwei Liu, Yizhong Liu, Jiayuan Yin, Rongxing Lu, and Xiaodong Lin. A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems. *Patterns*, 2(2):100179, Feb 2021.

[19] Docker In. Docker documentation, 2022. Available: https://docs.docker.com/.

[20] Intel. Intel software guard extensions., 2021. Available: https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html.

[21] Cynthia E. Irvine and Karl Levitt. Trusted hardware: Can it be trustworthy? In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, page 1–4, New York, NY, USA, 2007. Association for Computing Machinery.

[22] Merit Kõlvart, Margus Poola, and Addi Rull. *Smart Contracts*, pages 133–147. Springer International Publishing, Cham, 2016.

[23] Shu Yun Lim, Pascal Tankam Fotsing, Abdullah Almasri, Omar Musa, Miss Laiha Mat Kiah, Tan Fong Ang, and Reza Ismail. Blockchain technology the identity management and authentication service disruptor: A survey. *International Journal on Advanced Science, Engineering and Information Technology*, 8(4-2):1735, 2018.

[24] Medicalchain SA. Medicalchain, 2020. Available: https://medicalchain.com/en/.

[25] Sarwar Sayeed, Hector Marco-Gisbert, and Tom Caira. Smart contract: Attacks and protections. *IEEE Access*, 8:24416–24427, 2020.

[26] Scooley. Introduction to hyper-v on windows 10. Available: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/.

[27] sivachokkapu. Revive cc, 2020. Available: https://github.com/sivachokkapu/revive-cc.

[28] Trusted Computing Group (TCG). Device identifier composition engine, 2021. Available: https://trustedcomputinggroup.org/work-groups/dice-architectures/.

[29] Trusted Computing Group (TCG). Trusted platform module (tpm), 2021. Available: https://trustedcomputinggroup.org/work-groups/trusted-platform-module/.

[30] Allan Tomlinson, Keith Mayes, and Konstantinos Markantonakis. *Introduction to the TPM*, pages 155–172. 12 2008.

[31] Kazuhiro Yamashita, Yoshihide Nomura, Ence Zhou, Bingfeng Pi, and Sun Jun. Potential risks of hyperledger fabric smart contracts. In *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 1–10, 2019.

[32] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491, 2020.

A Appendix: Technology Roadmap

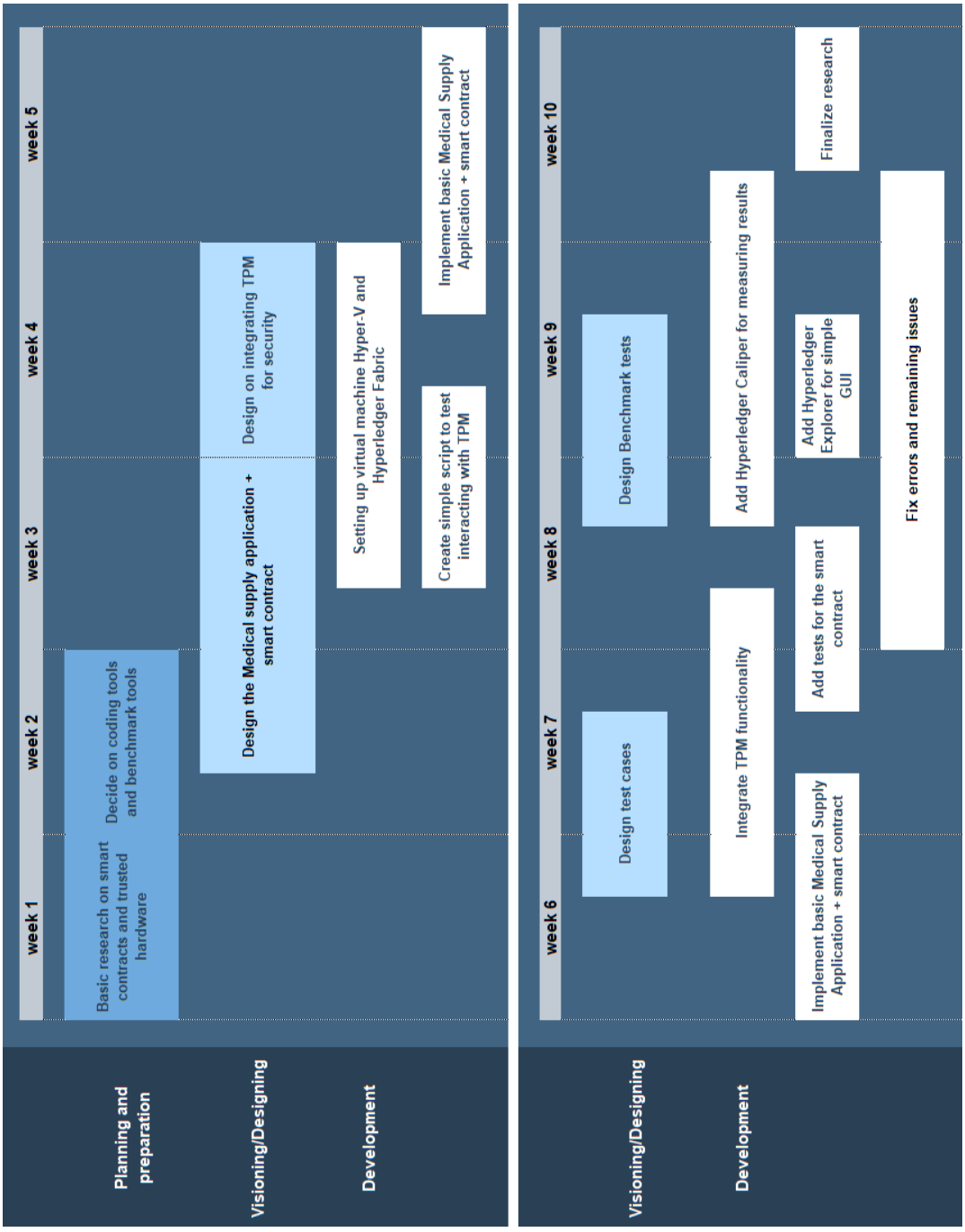

Figure 6: Technology roadmap of the research

B Appendix: List of implemented chaincode functionality

List of chaincode functionality implemented in the prototype which the peers within the MedStore network can invoke. These functions contain the changes made by integrating TPM such as added fields for the user name and tpm key.

- **TPMKeyGen(***username***)** - Generates tpm key, creates new TPMAuth object with the key and hashed user and stores it on the ledger. Returns error when TPMAuth already exists on the ledger.
- **InitLedger(***username, tpm key***)** - Adds a base set of available medical supply to the ledger.
- **Issue(***medicine name, medicine number, disease, expiration, price, username, tpm key***)** - Issues a new available medical supply to be added to the ledger. Returns error when invoker is not a Regulator or if adding medical supply to ledger failed.
- **Delete(***medicine name, medicine number, username, tpm key***)** - Deletes a medical supply from the ledger. Returns error when invoker is not a Regulator or if deleting medical supply from the ledger failed.
- **Request(***medicine name, medicine number, username, tpm key***)** - Sends a request for a medical supply on the ledger. Returns error when medical supply is not available or if requesting it failed.
- **CancelRequest(***medicine name, medicine number, username, tpm key***)** - Cancels a request for a medical supply on the ledger. Returns error if retrieving or cancelling failed.
- **ApproveRequest(***medicine name, medicine number, username, tpm key***)** - Approves a requested medical supply. Returns an error if invoker is not a Regulator, medical supply could not be found or if accepting request failed.
- **RejectRequest(***vmedicine name, medicine number, username, tpm key***)** - Rejects a requested medical supply. Returns error if invoker is not a Regulator, medical supply could not be found or if rejecting request failed.
- **CheckHistory(***username, tpm key***)** - Gets an overview of all medical supplies on the ledger. Returns error if there is no medical supply found on the ledger or invoker is not a Regulator.
- **CheckUserHistory(***username, tpm key***)** - Gets an overview of all requested and sent medical supply of an user. Returns an error if no medical supply is found on the ledger belonging to the user.
- **CheckAvailableMedicine()** - Gets an overview of all available medical supplies. Returns error if there is no available medical supply found on the ledger.
- **CheckRequestedMedicine(***username, tpm key***)** - Gets an overview of all requested medical supplies. Returns an error if there is no requested medical supply found on the ledger or invoker is not a Regulator.
- **SearchMedicineByName(***medicine name,***)** - Searches all available medical supplies given a medicine name. Returns error if no medical supply by provided name is found on the ledger.
- **ChangeStatus(***medicine name, medicine number, status, username, tpm key***)** - Manually change the state of a medical supply. Returns error if invoker is not a Regulator or changing status failed.
- **ChangeHolder(***medicine name, medicine number, holder name, username, tpm key***)** - Manually change the current holder of a medical supply. Returns error if invoker is not a Regulator or changing holder failed.