Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Fulfilling data comprehensibility requirements on blockchain platforms using trusted execution environments

Scientific Students' Association Report

Author:
Csilling Bálint

Advisor:
Dr. Kocsis Imre
Dr. Attila Klenik

BUDAPEST, 2023

Table of contents

Kivonat

A blokklánc-alapú elosztott főkönyvi rendszerek által biztosított nagyfokú integritás mélyen gyökerezik abban a tényben, hogy a blokklánc csomópontjai általában mindannyian fenntartják a közös, főkönyvszerű adatbázis egy-egy példányát. Ez a szempont azonban közvetlenül ellentmond az alkalmazásszintű titkossági és adatvédelmi követelményeknek; ha a csomópontoknak képesnek kell lenniük a tranzakciók szemantikai validálására, akkor az olyan naiv kriptográfiai megközelítések, mint az egyszerű kliensoldali titkosítás, nagyrészt nem működőképesek. Míg erre a dilemmára kriptográfiai megoldások születnek, egy másik lehetőség a megbízható végrehajtási környezetek (Trusted Execution Environments, TEE), például az Intel SGX integrálása a blokkláncplatformokba. A vezető vállalati blokkláncplatformok, például a Hyperledger Fabric és a Hyperledger Sawtooth legalább kísérleti TEE-integrációs támogatást kaptak, azonban a decentralizált alkalmazások ilyen környezetbe történő tervezésének módszertana még kiforratlan.

Ebben a cikkben egy reprezentatív, szervezeteken átívelő felhasználási esettel foglalkozom: a főkönyv "egyetlen igazságforrásként" szolgál az objektumok attribútumadataihoz, a tranzakciók komplex validációkat valósítanak meg, és az elosztott főkönyv tartalmán attribútumalapú olvasási és írási hozzáférés-szabályozást kell érvényesíteni. További követelmény, hogy a főkönyv felhasználóinak és üzemeltetőinek attribútum-hozzáférési jogai dinamikusan (bár viszonylag ritkán) változhatnak.

Ehhez a felhasználási esethez szisztematikusan modellezem a követelményeket, ismertetem egy újszerű Intel SGX-alapú megközelítés Hyperledger Fabric feletti tervezését, és bemutatok egy prototípus megvalósítást. Értékelem továbbá a megoldásom várható érzékenységét a múltban talált SGX-kihasználásokkal szemben, és mérséklési javaslatokat teszek.

Abstract

The high integrity provided by blockchain-based distributed ledger systems is deeply rooted in the fact that the blockchain nodes, as a rule, all maintain a copy of the shared, ledger-like database. However, this aspect directly contradicts application-level confidentiality and privacy requirements; if nodes must be able to validate transactions semantically, then naïve cryptographic approaches such as simple client-side encryption are primarily unworkable. While cryptographic solutions to this dilemma are emerging, another possibility is to integrate Trusted Execution Environments (TEEs), such as Intel SGX, in blockchain platforms. Leading enterprise blockchain platforms, such as Hyperledger Fabric and Hyperledger Sawtooth, have received at least experimental TEE integration support; however, the methodology for designing decentralised applications for this setting is still immature.

In this paper, I address a representative cross-organizational use case: the ledger serves as a "single source of truth" for object attribute data, transactions are subject to complex validations and attribute-based read and write access control must be enforced on the distributed ledger contents. As an additional requirement, the attribute access rights of the users and operators of the ledger are subject to change dynamically (although relatively infrequently).

For this use case, I systematically model the requirements, describe the design of a novel Intel SGX-based approach over Hyperledger Fabric and present a prototype implementation. I also assess the expected susceptibility of my solution against the kinds of SGX exploits found in the past and propose mitigations.

1 Introduction

The need for cross-organisational data-sharing schemes is increasing in the real world, which is getting more connected from a computer science point of view. More and more businesses and other organisations require a way to share information, even for daily operations. A promising technology that could potentially fulfil this need is blockchain technology. Distributed ledger technologies, but more specifically in this context, permissioned private networks offer many features that enable a way for organisations to collaborate. These features include transparency, robustness, and privacy, among other things. The contradiction of confidentiality and blockchains, however, is still an unresolved challenge, which is problematic for most use cases of enterprise blockchain solutions. The problem is rooted in the fact that for a blockchain to reach a consensus, most nodes need to validate transactions that happen on the ledger. The validator node needs to have access to the data to validate a transaction, but this can violate confidentiality requirements. One emerging technology that could be useful to resolve this contradiction is the use of trusted execution environments [1] in nodes. HyperLedger Fabric [2], the de facto standard of enterprise blockchains, provides an experimental extension of the framework to utilise TEEs, but the methodology for designing decentralised applications for this setting is still immature.

This paper will be organised in the following way: First, I describe the main challenges and details of privacy and confidentiality in the context of blockchains, then give an outlook of how the problems are attempted to be solved by researchers, and I also provide my brief analysis on these solution approaches. Then, I will lay out the foundation for on-chain data comprehensibility and give a running example that will be used the aid the understanding of requirements. After which, I share my constructed requirement system for a potential system. Then, I explain important caveats of the technologies we can leverage. Then, share my proposed approach to designing such a system, explain the threat and system model of said approach, and give a brief introduction about the used cryptographical primitives. Lastly, I share my prototype implementation of the designed protocol and evaluate the results according to the requirements and the running example.

1.1 My contributions

In this paper,

1. I construct a requirement system for a specific type of cross-organizational blockchain cooperation model, which supports dynamic data comprehensibility rules, which is not a solved problem yet.
2. I also assess typical solution approaches in the literature about similar problems, and based on this review, I suggest the use of trusted execution environments to solve the challenges.
3. I also propose an approach based on the Hyperledger Fabric Private Chaincode framework [3] to fulfil the complex challenges that come with dynamic data comprehensibility without the use of application lever barriers. The approach also provides many unique features, such as auditable reads, complex data validation, and forced access.
4. I also implemented a prototype of the proposed solutions and evaluated the results with the requirement system, as well as with my running example.

2 Privacy and confidentiality in blockchains

Distributed Ledger Technology and specifically blockchains, target a very high level of data integrity so that the data maintained by a set of nodes, also called peers can be maintained without a third party as an authoritative source. Data is fully or partially replicated between the nodes (in most blockchain systems in practice this means full replication). Other extra functional aspects such as reliability, availability maintainability, confidentiality and privacy are usually secondary design targets, and their level is, in many cases, quite situational.

Confidentiality and privacy, as a rule are contradictory requirements with high integrity under the design philosophy of blockchain-based distributed ledgers. Simply put, if many nodes maintain a copy of the ledger, then that enables maintaining integrity even under Byzantine fault and attack models, but at the same time, the number of quote "eyes" seeing the contents of the Ledger is also highly increased. This core challenge is present across the board, from large, unpermissioned public access networks accounting for cryptocurrencies to bespoke, permissioned consensus and closed access networks created for cross-organizational cooperations.

There are many existing approaches for assuring either privacy or confidentiality in various blockchain and distributed Ledger settings, some of these are cryptographic, others are system architectural and yet another family of approaches utilize dedicated hardware support for confidential computing. In this work the target setting is permissioned blockchains with permission access for clients. There is a consortium of organizations that operates the blockchain network and the data stored by the network should be readable or at the very least comprehensible only by subsets of organizations and their systems and employees determined on a attribute per attribute basis. The additional challenge that we also tackle here is that the data comprehensibility rules at the level of organizations change on the time horizon off several months, that is not very frequently, but frequently enough for system level overhauls of a blockchain based solution not to be a practically viable option. This model intentionally reflects a quite usual setting in the public sector and specifically for various governmental bodies and agencies. There are various kinds of sensitive, private, personal and business data to which only subsets of the organizations can have access to, as they determined by pertinent regulations, and the set of organizations having the right to access and potentially handle a specific kind of personal or business data does change with changes to the legal environment, and the updates of the respective regulations. At the same time, there would be very high value in maintaining a single source of truth database of personal and business-related data; however, it has to be guaranteed that organisations of the public sector have access to the data stored only as long as laws and regulations allow them to do so. It is also a basic requirement that access to sensitive information should be

logged for audit purposes and preferably so that no single organisation is able to hide attempts.

Most applications of blockchain platforms require some level of privacy and confidentiality built into them, but this is often directly in opposition to what distributed ledgers are trying to achieve. Even a simple scenario can become problematic, where three organisations are trying to work together, wherein only two of the participants are allowed to read a certain document on the ledger state, and the third participant must agree to a world state, which it cannot read, thus cannot verify. Problems also arise from the fact that the ownership of a certain document should be able to dynamically change in most cases, which further complicates the cryptography of said documents and opens a door for side-channel attacks, where the participant just feeds outdated data to its local node to gain access to data it should no longer have access to.

2.1 Blockchains

Understanding blockchains, is key to fully understand this paper. For this reason, I provide a short explanation of the technology. Blockchain is a distributed ledger technology which is essentially a database that is managed by a network of computers rather than a single entity. The database is decentralised, meaning that it is not controlled by any central authority or server. Instead, each computer in the network, usually called a node, maintains a copy of the database/ledger, and all copies are kept in sync, usually with the help of a consensus mechanism.

The core concept behind blockchains is blocks that serve as containers for batches of valid transactions, which undergo the processes of hashing and encoding into a Merkle tree. Within each block, one can identify a set of essential components, including a reference to the preceding block via its hash, a timestamp, a Merkle Root summarising the transaction data, and an array of individual transactions. This iterative sequence of block creation serves the pivotal function of affirming the integrity of the preceding block and, ultimately, traces its lineage back to the initial genesis block. Following their incorporation into the blockchain, data housed within these blocks remain shielded from tampering and retain their auditability for the duration of the blockchain's existence. In the event of any modifications to a previously created block, the hash value will change, enabling a straightforward identification and rejection of altered blocks.

This concept was first detailed in the famous Bitcoin whitepaper [4] by an unknown person or group, Satoshi Nakamoto, and it is still the foundation for most blockchain platforms. The decentralisation of control among participants, the transparency of ledger data as well as the tamper-proof nature of blockchains enables them to be an essential building block for complex architectural systems from cryptocurrencies to large-scale enterprise solution, which I am targeting in this paper.

Although the categorization of blockchains take many forms, the most common visualization targets two specific aspect of platforms namely publicity and permissioning.

	Public	**Private**
Permissionless	Everybody can read and interact with the ledger*. Example: Bitcoin	Everyone can interact with the chain, but only predefined entities can read the ledger state. Example: Monet
Permissioned	Data is publicly readable, but only predefined set of entities can interact with it. Example: Ripple	Only predefined entities can read and interact with the ledger state. Example: Hyperledger Fabric

1. Table Blockchain quadrant

2.2 Typical solution approaches and related work

Building a scalable, performant, robust and complex business process with high levels of dynamic privacy and confidentiality consideration is a difficult balancing act that has not been solved yet in its entirety, but let's see how parts of the challenges are currently being solved.

2.2.1 Cryptographic solutions

The idea of having complex access control over shared data has been on the mind of many cryptographic researchers, and their breakthroughs enable us to build complex systems with purely cryptography. The most common approach is to use Attribute based encryption [5], [5]–[12] or some improvement thereof, to manage complex data accessibility requirements.

Typical challenges of cryptographic solutions are the handling of change in read access, for instance a revocation, and how that affects the system. In attempt to handle revocation, papers like [8] use forward secrecy, which means that the user will not be able to see subsequent states of the data, because it will be re-encrypted with differently to the previous state. This is problematic for two main reasons, one of them is having the need to re-encrypt all data upon any changes in access control, which even with an efficient algorithm, is an unnecessary price the maintainers and the environment has to pay [13]. Second problem is having access to all previous states of the data, this at first might seem like an unavoidable problem, but upon closer inspection, we can realise that having the possibility to access data and reading it is different. This means that in a system where users

are allowed to see encrypted data might privately keep a copy of this data to then later decrypt, even when it should no longer have access to it.

To combat the need for re-encryption and to achieve faster revocations, approaches [14], [15] used a proxy, that cannot decrypt the data itself, but it is used in the decryption process and enforces revocation constraints. Even though this addresses some of the problems, data is still stored in centralized way, and to access data, we must rely on a third party. Denial of service attacks, as well as having single points of failure is not ideal, and unfortunately most approaches use a cloud to store the data.

What has not been discussed yet, is modifying, or using the data without decrypting it first. In a distributed data storage where multiple parties keep a record of the shared state for the sake of robustness and to make tampering more difficult, it is a challenging task to compute with the data without first decrypting it. In a use case where a party wants to do a modification on a shared data, without giving the ability for parties to gain information about this sensitive data was a difficult challenge, but recent innovations in Fully Homomorphic Encryptions [16] enable us to maintain confidentiality and data integrity on date in use. According to a recent systematic review of FHE [16], the most common application for the concept is cloud computing, but they also noted that most papers that are being developed aim to "enhance efficiency and to reduce noise", that is because currently the technology is not highly efficient nor can it handle complex operations yet.

Zero-Knowledge Proofs [17] are also a promising new cryptographical solution that is being used to aid decentralized and transparent data systems with efficiency and privacy. To understand how ZKPs can help a system, consider a private data that a person owns, and dependent on this data, another party allows or denies an action. With the use of ZKPs, the person can prove to the other party that he is eligible for that action based on the state of its private data, without providing any information about the actual data. Recent approaches used client side ZKP generation and a blockchain to store and manage data. [18], [19] The problem with ZKPs is partially the computation inefficiency, as it is computationally expensive to calculate the proofs. [20] The other notable problem is the fact that if a computations depends on several sensitive data, that we do not have access to, ZKPs don't offer general way to handle this. (I imagined a scenario where a participant wants to calculate the average height of the people, and requests this secret from everyone, there is no way to share even a simple data like this with a party while maintaining confidentiality. This problem can be solved however with the combination of FHE, ZKP and a trusted third party, but we still cannot be sure that the submitted heights are not tampered with.)

In summary, cryptographic solutions provide useful approaches to data dynamic confidentiality, data integrity, verifiable computation, but often at a cost of intense computations, trusted proxies, single points of failure and they often lack capabilities for real-life use cases which need very complex data validation.

2.2.2 Architectural approaches

In recent years, the most common approach for solving privacy and confidentiality problems in a practical way usually consisted of using trusted third party [14], [15]. As mentioned earlier, pure cryptographic solutions usually lack practicality due to requiring large amounts of computation, as well as lots of communication between the parties.

Depending on the blockchain architecture used, there can be ways to segment the storage and access of data at the architectural level.

One important example is Hyperledger Fabric [2], which follows a much more sophisticated architectural model then the blockchain platforms primarily geared at the public space, notably Etherium. In a Hyperledger Fabric network, a set of organizations form a consortium by participating in securing the whole network. However, that network is actually comprised of multiple blockchain-based ledgers. These are called channels. While under the proper circumstances smart contracts-in Hyperledger Fabric can issue read requests from a channel to another, channels essentially partition the data stored and maintained by the network to various subsets of the organizations. This essentially architectural method facilitates the control of data visibility without resorting to cryptographic methods.

Additionally, Hyperledger Fabric also offers so-called *private data* facilities; a subset of the data stored on a (channel) ledger can be declared to be visible only by a subset of the parties maintaining that channel. Private data is persisted on ledger, only its hashes, and the channel peers who do have to see it exchange it through a dedicated peer-to-peer mechanism. In contrast to ledger-persisted data, private data can be also declared to have a time to live property; if that time elapses, the peers storing it delete the data.

The requirements tackled in this paper can be fulfilled in many practical scenarios using a combination of data storage partitioning with channels and private data objects. However, from an *operational management* point of view, reflecting changes in data access permissions at the organizational level through rearchitecting or reconfiguring a channel setup is problematic from the effort and service outage point of view. Additionally, the non-ledger stored nature of private data objects can be, and is many times a serious drawback. Lastly, the basic Fabric model is not amenable to creating solutions where the strong auditing of data access is to be enforced.

R3 Corda would be another important example. Corda is a DLT, but not a blockchain; there is a sophisticated approach for globally ordering transaction requests in a Corda network, but data is only stored by the network nodes that are actually pertinent to performing various transactions. For instance, when a network node creates an asset and hands it over to another one, the related data will be exchanged between these two nodes using the common network level

transaction ordering services. For the use cases targeted in this work this model is suboptimal, as it is ill-equipped to support distributed ledger usage as a common root of trust for a community of organizations. Additionally, Corda, at least for production deployments, is only partially open-source software (specifically, components enabling high performance are not free and open).

2.2.3 Trusted execution environments

An emerging way to ensure privacy and confidentiality on data in use is with trusted execution environment. This approach is fairly new, the technology first become truly recognized when Intel released its Intel SGX processors in 2015 [21]. TEEs promise to provide complete confidentiality, privacy, and integrity of data that is being used by the TEE. Having a completely trusted part of a computer can be utilised in diverse ways from cloud computing, to digital rights management (DRM).

TEEs use hardware-based isolation of components to differentiate the trusted and untrusted part of the system. This means that sensitive data can be stored and handled differently. The key components that enable rooting the trust in the hardware are called the Trusted Computing Base [22]. The TCB includes all of the hardware and software components that are required to establish a secure environment, for instance a secure processing unit, secure memory, the TEEs runtime or its cryptographic implementations. We can use these trusted components to create enclaves, which are generally a more specific uses of TCBs, to establish an environment to run software with high levels of confidentiality and privacy requirements.

The three main player in the TEE space is Intel with its Software Guard Extensions [21], ARM with the TrustZone [23] and finally AMD with its Secure Encrypted Virtualization [24]. The most widely used of the three is Intel SGX, which offers a way to fix vulnerabilities, after the processor has been bought with using remote attestation, among other things. (Discussion about Intel SGX is further extended later in the paper.)

Use of the technology is not wide-spread, but it is gaining traction [Appendix]. First use cases included digital content management, largest example being PowerDVD [25], which is a software that enables Intel SGX enabled computers to play Blu-ray content, using TEE to manage the disk decryption keys.

Although this is argumentative, it is often the case, that during the design process of an application, security is unfortunately only an afterthought, even in large scale enterprise solutions [26]. This is rarely the case at blockchain related solutions, as robustness, integrity, security and transparency is often the main reasons architectures consider using blockchains [27]. There is however and underlying issue between traditional blockchains and confidentiality. If we want to achieve transparency confidentiality and validity, we hit a hypothetical roadblock, which is that it is a challenging thing to verify and validate a data on

all of the nodes of a network (or at least enough nodes to reach consensus) and at the same time keep the confidentiality of said data. It is easy to see that TEE enclaves are more than useful to clear this roadblock, because they enable us to verify data on mostly untrusted nodes.

The most well-known technology is the Hyperledger Fabric Private Chaincode [28], which is an extension of the Hyperledger Fabric [3] blockchain framework. It uses Intel SGX as a trusted execution environment and enables smart contracts to be run in an enclave, which can access the ledger state opening the door for a large variety of use cases. (HF and FPC are both discussed in more detail later in the paper. [Section 3])

In the Rust and Substrate ecosystem, there has been a few attempts to create something similar to the FPC, but they did not gain significant attention [29]–[32] as of writing this paper. SubstraTEE [33] is a notable development that enables the use of Intel SGX in the Polkadot [34] ecosystem.

A very recent paper proposed a system where we can leverage TEE and blockchains for a privacy reserving divisible auction. [8] This paper also uses Hyperledger Fabric to experimentally test their approach, but they did not use Fabric Private Chaincode, nor evaluated and mitigate the effects of an untrusted ledger, an a compromised TEE.

Unfortunately, to this day, people have found vulnerabilities in every mainstream TEE implementation. A great review of Intel SGX and its vulnerabilities was released in 2022 with the title: "SoK: SGX. Fail: How stuff get eXposed" [35], which systematically reviews applications that used SGX and got their integrity compromised with vulnerabilities, as well list the most common vulnerabilities and what mitigation strategies we can do to regain confidentiality. One of the victims of a vulnerability was the Secret blockchain [36], which promises privacy-preserving smart contract execution, but through the use of a vulnerability called xAPIC and MIMO [37], [38].

To conclude this section, it is important to highlight that TEEs enable the possibility of many use cases, especially regarding confidentiality and computation, and blockchain system can benefit from these massively. Unfortunately, TEE technologies are not without problems, there are vulnerabilities found (and fixed) day to day, which can impact the usability of these systems, but this paper uses this technology as a black box to explore what is possible to create. A few vulnerabilities and mitigation strategies can be found in the Appendix.

3 On-chain data under dynamic comprehensibility control

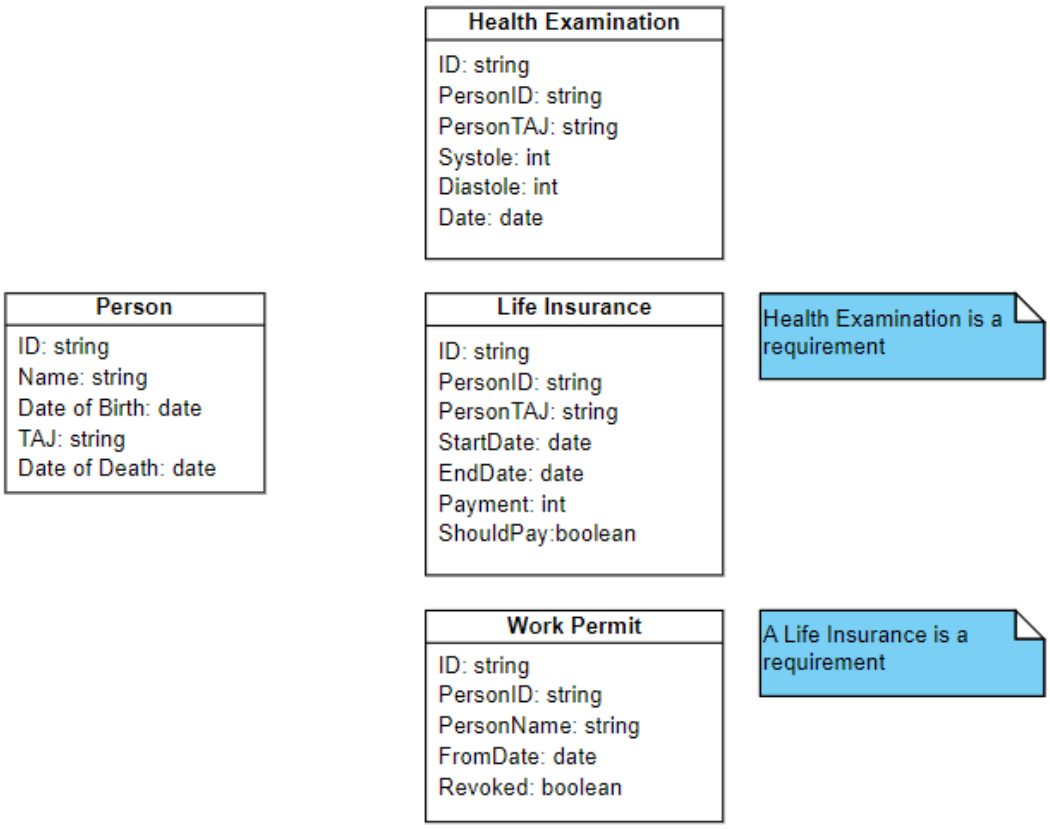
In this section, I will introduce a realistic scenario where organisations need to collaborate and manage data in an organised way. This example will be used to explain the requirements of a system I are trying to construct.

3.1 Running example

I introduce a simple, hypothetical running example which is nevertheless representative for my purposes. Let's imagine a scenario where at least four organizations must work together. Firstly, a state-governed entity responsible for administering born people, giving them an ID number, SNN number, etc. Secondly, a medical institution that is allowed to conduct medical examinations and register these results to the private, shared ledger. Thirdly, a life insurance company that can issue a life insurance for people. Lastly, one more state-governed entity, that can issue work permits to certain people.

In this hypothetical scenario, the medical institution can only conduct examinations, if it can verify the person's identity, requiring the input of the state governed entity, which should not share any information about the person with the medical institution, apart from the necessary information that the person is valid. Validity in this scenario means that the provided SSN and ID number are matching a real person in the shared private database. The life insurance entity requires applicants to have a recent medical examination, that checks whether the person has a healthy vascular system, where normal blood pressure numbers could be a great proxy for assessment, if the person is not *healthy*, the life insurance company should not be able to issue insurances. Similarly, the work permit related institution also requires people to have life insurance, to issue a work permit. Important aspect to highlight is the fact that the organizations are handling sensitive data, so upon a validation, no unnecessary data should be gained by the using party. (In the case of the work permit organization, the only information they should gain is the fact that the person has a life insurance.

These organizations all create document like data regularly, and these data are usually highly sensitive. Yet, these entities must share some of these documents, or parts of the documents with each other to function properly. This also means that they must have complete trust in the other party, that it will not corrupt or modify the data because of financial or other incentives. We also have to assume, that upon a change in the accessibility of a data, or just a hierarchical change in an organization, pervious accessors and handlers of the data, that could access information suddenly "forget" and not retain access or copies of previous information.



1. Figure Simple data model of the use case

According to dynamically changing permission scheme, organizations should have the ability to read out a certain information from the ledger in its entirety. This should be only allowed, if the preconfigured number of participants agree on the legitimacy of the read request.

To aid legal and compliance matters, it is also required by the organizations, that any access to sensitive information must be auditable, and trackable, without the possibility of forgery, or unlawful access.

3.2 Requirements

My system design targets the following functional, integrity, confidentiality maintainability and audibility requirements:

F1. The system should follow the distributed ledger pattern in the context of a *consortium* of organizations; that is, incoming transactions should be validated, ordered and executed through an appropriate consensus mechanism, ran by members of the consortium.

F2. The system should handle personal data, where persons have a set of attributes, documents of which they are the unambiguous subject of, and there can be relationships between the documents pertaining to a person.

F3. The system should support domain-specific validation logic for person, attribute and document creations, updates and deletes.

F4. The system should also host a dynamically updateable authoritative data access and handling control list, which defines that which organization, as a member of the consortium, is allowed to create, read update or delete which attribute and document(s) of any given person.

I1 – DLT integrity: The tolerance of the system against organizations with Byzantine fault/attack behavior should be maximized in the context of maintaining distributed ledger integrity.

C1 – read protection: Without the collusion of multiple organizations, it should not be possible for an actor at an organization to query data on-ledger to which it currently has no access right.

C2 – operation attempt based information leakage: a malicious actor at an organization should not be able to infer any information about the data it does not have currently access right to through attempting (ultimately unsuccessful) operations on the data.

C3 – observation-based information leakage: a malicious actor at a participating organization should not be able to infer any information about the data it does not have currently access right to through observing the changes performed on such parts of the ledger which are replicated at it.

M1 – minimally disruptive changes to control list: changing the data access permissions should lead to minimal disruptions in operations – ideally, it should not require any maintenance downtime.

AU1 – In the absence of collusion between multiple organizations, every successful read access by any member of a participating organizations should be logged irrepudiably.

3.3 Hyperledger Fabric

To understand my approach, there are certain technologies and concepts that need to be understood. I analysed and assessed the usability and security aspects of technologies, to understand problems are already solved, and how we can utilize and combine concepts to fulfil the complex set of requirements.

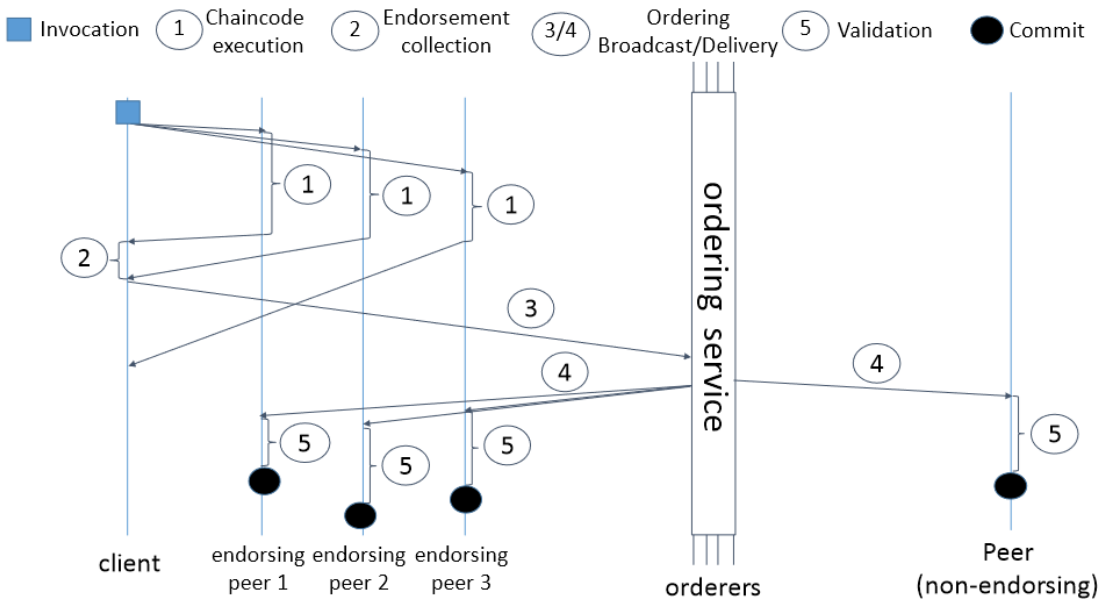
Hyperledger Fabric [2] is an enterprise blockchain platform framework from the Linux Foundation. It's modular architecture and open-source code base enables developers to quickly build secure, performant, and customized blockchains for a wide range of use cases.

Fabric is mainly used to build permissioned networks, meaning that the participants of the system are known, and trusted, anonymous participants are not allowed to participate or interact with the blockchain in a meaningful way. Hyperledger Fabric is also built on distributed ledger technology, which means that participants of the network keep a "copy" of the state of the ledger, and any modifications or manipulation of data must reach a consensus among the members to be considered permanent. This makes the forgery and the unwanted manipulation of data extremely difficult. Participants of the network can create complex privacy controls, so confidential data can only be seen or manipulated by agreed-upon parties.

Complex business processes can be implemented using smart contracts, which can automatically execute program codes and create transactions on the distributed ledger state. The transactions are trackable and nearly impossible to retrospectively manipulate the history of thereof.

I now give an outline to how a Hyperledger Fabric blockchain typically operates. (Fabric is highly customisable, this process could be different) The following section will be based on [2], [39] pg.291.

There are organizations that established a Hyperledger Fabric network, by settling on configuration parameters, as well as physically creating the network. A Fabric client, which is not a computing resource of the network creates a transaction proposal to reflect a change in the world or achieve some goal. After creating the proposal, it has to send it to peers. Peers are nodes that execute, validate and commit transactions. After a peer received the proposal, it can execute the transaction on condition that it has the required Chaincode locally on the peer. The execution happens based on the current state of the local blockchain, and it generates a read-write set as an output. Any changes that might have happened as a result of the transactions are not yet persisted on the local blockchain, the results are now only signed and sent back to the initial client. The client receives this data, called an endorsement, and depending on the endorsement policy, it has to receive this from multiple peers, multiple organisations. Once it has enough endorsement that fulfilles the policy, it creates an endorsed transaction and sends it to the ordering service. The ordering service is also consisting of computing nodes (or just a single one), that essentially determine the order of the blocks by collecting transactions into a single block, than broadcasting this block to other nodes. After receiving a block, peers validate the transaction by conducting policy, syntax and read-write set validation. If it has passed all types of validation, its result can be committed to the world state, and the initial client can be notified of the change in the world state.

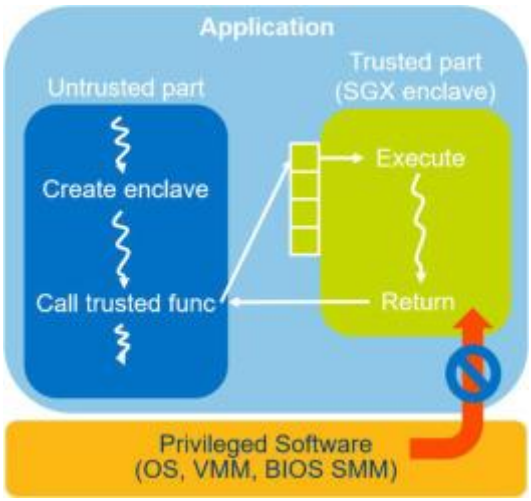


2. Figure High level transaction flow Source: [2] Fg.4

3.4 Intel Software Guard Extensions

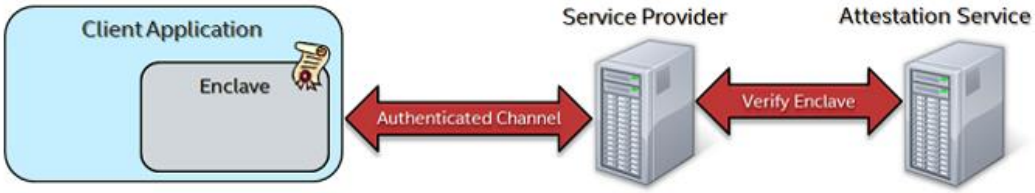
Intel's Software Guard Extensions [21] is a trusted execution environment technology first introduced in 2015. It adds a hardware-enforced layer of security to the regular computing platform by extending Intel's CPU architecture. It aims to provide integrity and confidentiality in computation, even when the rest of the architecture is compromised.

Based on a hardware-rooted trust, the architecture ensures that only trusted code can run in the enclave, and only trusted code can access certain parts of the memory. Communication with this isolated trusted component can be conducted thorough and interface, but the enclave has no direct access to I/O. Although the memory of the physical enclave is limited, it can use a process called sealing to store confidential data outside the enclave by encrypting the data with its sealing key.[40]



3. Figure Code execution with SGX Source: [1]

Another important feature of SGX is its ability to provide remote attestation, which allows a third party to verify the integrity of an SGX enclave without having access to its contents. With its remote attestation protocol [41] an enclave can prove its identity, that it is running a genuine platform with Intel SGX enabled, that is has not been tampered with, that it is running at the latest security level. This feature is particularly useful in scenarios where multiple parties need to verify the integrity of an SGX enclave, such as in a distributed system where each node needs to verify the identity of its peers.



4. Figure Remote Attestation at a glance Source:[41]

At the beginning, Intel provided SGX capabilities to its consumer CPUs for a few generations, but in 2021, it phased out the inclusion of enclaves in these consumer products such as the 12th Generation Intel Core processors [42]. Luckily as of 2023, Intel SGX is still being provided, and developed in enterprise hardware such as the Intel Xeon series. This is crucial for the usability of the technology, since vulnerabilities are often found by researchers and attackers, and fixing these issues

usually requires either providing guidelines for users, software or even hardware fixes.

In recent times, Cloud providers such as Microsoft Azure or Alibaba Cloud have started to provide Intel SGX enabled hardware to be used by costumers, further improving the adaptability and usability of the technology.

3.5 Fabric Private Chaincode

3.5.1 Background and motivation

Even in private blockchains, the state of the ledger is by nature public to its predefined user entities. This is essential for the blockchain to work because nodes that validate transaction data from other nodes must have access to the date they are validating. This is not the say that all blockchains are uncapable of storing application layer encrypted data, but when multiple participants of a blockchain must work together based on a predefined set of agreed upon rules, data generally cannot be kept private. To illustrate this differently, let's imagine a simple auction, where participants make bids for a certain item, but we want to keep the identity of the bidder, as well as the actual bid private. Normally, it is not possible for all participants of the blockchain to come to the same conclusion of who the winner is, since they have no way of assessing other participants' bids. This is the reason why blockchain and this type of private data is a difficult challenge. Lot of non-decentralized solutions would call for an external trusted party, which would then collect the bids then announce the winner, but this introduces a lot of problems. Apart from being a single point of failure, participants of the network would need to not only trust this party with their private data, but also assume that no other party will collide with the mediator. Another aspect worth mentioning is that this approach would also require participants to share the evaluation algorithm with this third party, which could also be problematic.

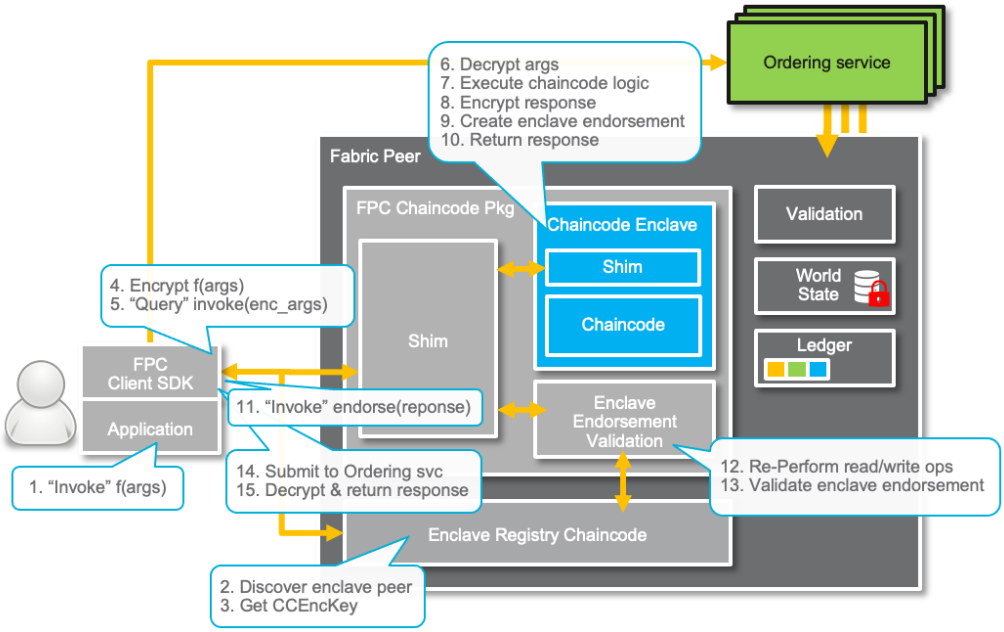
To solve this complex problem, trusted computing was introduced into the Hyperledger Fabric blockchain platform [3], [43]. The motivation behind this development and research is three fold, according to the Fabric RFC [44]. I only focus on the main reason for now, which was to enable new use cases for Hyperledger Fabric with strong privacy requirements. They described the problem perfectly in the RFC [44].

"FPC is primarily motivated by the many use cases in which it is desirable to embody an application in a Blockchain architecture, but where in addition to the existing integrity assurances, the application also requires privacy. This may include privacy-preserving analytics on sensitive data such as regulated medical or genomic data, supply chain operations requiring contract secrecy, private voting or sealed bid auctions. With Fabric's current privacy mechanisms, these

use cases are not possible as they still require the endorsement nodes to be fully trusted. For example, the concept of channels and Private Data allows to restrict chaincode data sharing only within a group of authorized participants, still when the chaincode processes the data it is exposed to the endorsing peer in clear. In the example of a voting system, where a government may run an endorsing peer it is clear that this is not ideal."

3.5.2 Modified transaction flow

In FPC if a client wants to create a transaction proposal, it can encrypt the arguments of the invocation with the public encryption key of the enclave, more specifically the Chaincode's enclave, and send this to a peer. To access this public key, the client must request this from the peer, and with the result, query the attestation result of corresponding enclave. This is critical to ensure that the enclave is running a legitimate Intel SGX hardware, as well as to make sure that it is running the intended Chaincode, that it wanted to invoke. The peer will transfer these arguments to the enclave, process it and return the result to the client. Even if the peer is malicious, it cannot gain access to the contents of the arguments or the execution of the Chaincode.



5. Figure FPC Transaction Flow Source:[44]

3.5.3 Storage in FPC

Without diving deep into the storage model of Hyperledger Fabric, I will describe how an application would use it. Fabric's world state is essentially stored in a key-value storage on peers. This key value storage is generally public to the participants of the network, not just the keys, but also the values can be seen by any participant. Off course it is common to use application layer encryption to

make date private, but it is not mandatory. While reading of data can be done without necessary invoking transactions, any modification to data has to go through the transaction flow detailed earlier. This architecture alone does not support advanced privacy and confidentiality use cases. The FPC was built on this as an extension. Compared to a regular Chaincode, where we can manipulate and read the world state with getState(key) and putState(key,value) as well as with getStateByPartialCompositeKey(partialKey) developers can use the same commands, but they work differently.

In FPC chaincodes, we can also access and modify the public key value storage, but by default, these functions will use the state encryption of the enclave, meaning all the stored data will be only comprehensible for the enclaves. Chaincode developers must make sure that they do not write faulty chaincodes, that leak confidential data in the results, because that is not encrypted for the client. Problems with the FPC storage will be detailed in [Section 4.4.5].

3.5.4 "Barriers"

From the previous sections, one might jump to the conclusion, that essentially every building block is given to create a complex enterprise solution, that handles private and confidential data with dynamic comprehensibility and complex validation logic. This is unfortunately not the case. The original FPC [3] paper titled "Blockchain and Trusted Computing: Problems, Pitfalls, and a Solution for Hyperledger Fabric" shows us the problems with blockchain applications and confidential computing.

As mentioned earlier, Hyperledger Fabric uses a execute-order-validate paradigm, which means that an attacker might gain sensitive information about the world state by executing a lot of transactions. The paper uses the example of a private auction. In this instance a malicious participant could potentially close the auction locally to reveal the highest current bid, then to submit a slightly higher bid. This would technically possible since the transaction could be built on top of valid blocks. To combat this, they proposed the idea to adapt applications to include *barriers*. Barrier in the auction example would be to require open and close transaction during the auction, and logically bids could only be submitted during the open phase, and the result cannot be seen until the auction is closed. This application-level mitigation combined with the FPC-s solution eliminates very critical attack vectors. The solution is to only allow transaction to happen upon committed block chain state. This restricts the attacker from placing the *barrier* itself, but this method still does not address many issues with data confidentiality and privacy.

3.5.5 Critical problems

In the paper [3] they formalize the information leakage with a simple model [pg.6].

„...we model a blockchain as stateful functionality $F : S \times T \rightarrow S$. At any time the state of the chaincode is an element of S. The clients invoke transactions in T, which may contain operations with arguments according to F , but these are subsumed into the different $t \in T$. Given $s \in S$, applying a transaction $t \in T$ of F means to compute $s' \leftarrow F(s, t)$, resulting in a subsequent state $s' \in S$...."

The goal was to achieve "security against up to resets" which means that a malicious peer might only obtain $s_{k+1}^* = F(s_k, t^*)$ for any $k \in \{0,1...,m\}$. Informally, this means that with the use of roll-back attacks, they might get access to every state that could happen in one transaction. They acknowledge, that this is a problem, because what this essentially means, is that we can not build applications with it that release sensitive data based on the ledger state. In my example use case, this could mean that an organization might be able to read sensitive information that is should not have access to, or it just might get information it should no longer have access to. Important to mention here that my protocol also requires auditability for every data access, which would also be violated.

There has been plans (GitHub Issue: [45]) to extend FPC with the Roll-Back protection and thus Trusted Ledger, but this is not yet implemented.

Another problematic aspect of the current FPC implementation, is that it does not support multiple private keys to be used by a chaincode. This combined with not having the capability to use composite keys, the possible data structures become impractical and even inefficient. To illustrate this, consider looking at the use case data structure and then the prototype implementation [Figure 1, 17], where instead of using a constant time look up for unique keys, we must create a large structure to store every private data used by the smart contract, then parse this on every read. Which could have been a very efficient key-value lookup turned into parsing every data, a key-value lookup, modification, marshalling the entire data and storing it. (This also comes with a few beneficial side-effects, explained in [Appendix]) According to the developers, this only a limitation of the so called FPC Lite, so we will hopefully see an extension of this in the future.

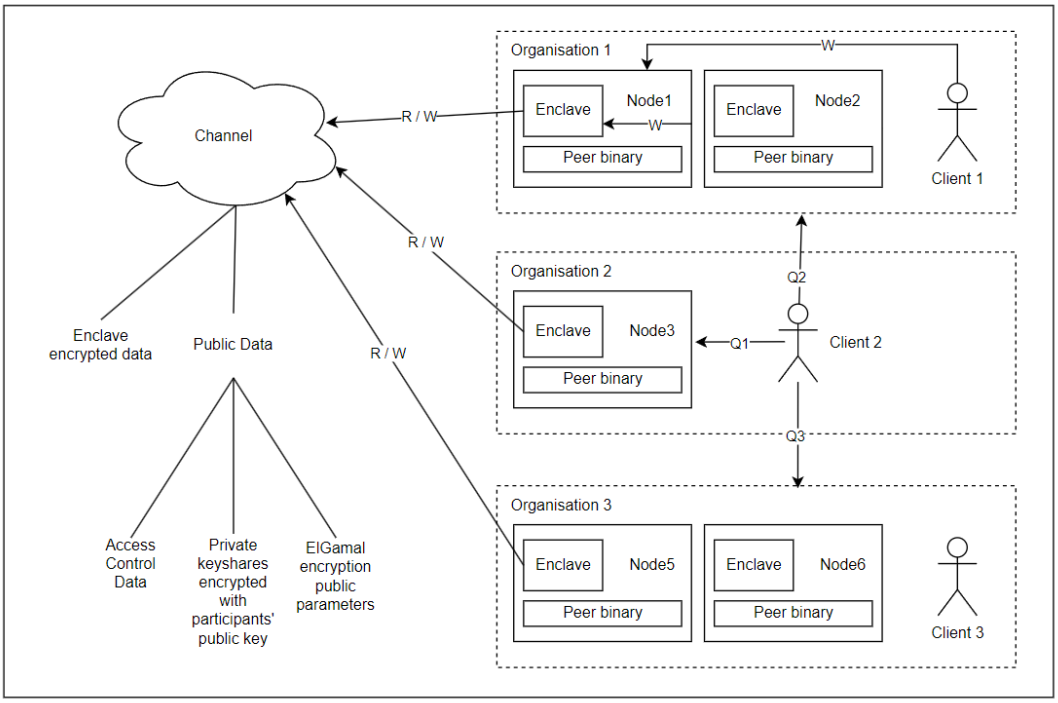
Continuing with other FPC Lite limitations, it is not possible to invoke another chaincode from an enclave. This is not necessarily a use-case breaking problem, but we should consider that implementing complex applications into one chaincode can become unmaintainable. The full list of FPC Lite limitations can be read studied in the shim interface code [46].

Lastly, I wanted to note that the only TEE platform that the Hyperledger Fabric Private Chaincode supports at the moment is the Intel SGX platform.

4 A TEE-based blockchain comprehensibility management approach

Now that I assessed the state of the art technologies, methodologies and its potential problems in the context of our goals, I propose an approach to fulfil the above-mentioned complex requirements.

4.1 Proposed approach



6. Figure High level view of the approach

We can utilise confidential computing with the Hyperledger Fabric Private Chaincode implementation to have a permissioned blockchain architecture with trusted execution environments. The actual protocol will manifest in the implementation of a chaincode (smart contract) running in the enclaves (Trusted execution environments of the FPC), as well as an external program to run cryptographical algorithms outside the blockchain ecosystem. The protocol relies on threshold cryptography to enable high levels of security and mitigate the risk of a single point of failure. Assuming that the used TEE is safe to use, and that the used cryptographic algorithms are *hard* [47] to solve the approach can fulfil every stated requirement.

Organisations of the network establish a set of rules about interacting with the system. These rules only represent the permissions of an organisations, and how they can interact with the public chaincodes they constructed for cooperating with each other. After establishing this public machine-readable document, one of the participants uploads this document through a transaction, which if parties

agreed upon this document, everyone should accept. Every following transaction will first be validated by this permission scheme. The permission scheme validator must be a function A: $S \times T \rightarrow R \in \{0,1\}$, which for every state s $\in S$ and transaction t $\in$ T it results in either true or false, representing whether a ledger state, which has the current permission scheme allows the transaction to happen (transaction includes the identity of the caller)

Next to access control, parties also establish a validator function, that represent whether a transaction invocation is valid in terms of the need of the business process/application. This validator function formally V: $S \times T \rightarrow R \in \{0,1\}$, checks for meaningful arguments, and notably the configured validation. To use my example, use case, the previous function would check whether a medical institution has the right to issue a work permit(no), and this V function would validate whether the transaction was called with valid arguments, and that a work permit is being issued to person with valid life insurance (which is a requirement in this scenario).

Now that we have a chaincode and blockchain that conforms to the above description, the participants can start to invoke transactions to store confidential data on the ledger. Invocation happens trough a safe channel (FPC client) with the use of asymmetric cryptography, so the contents of the transaction remain unknown to other participants, only the enclaves are capable of decrypting and validating the transaction with the chaincode functions. Important to note that fact of something changing in the key value storage does not remain unknown to other participants. For instance, a node receives a transaction proposal it needs to execute, it uses its enclave to execute the transaction, sends back the proposal, and if the transaction is committed to the ledger, it will be publicly available information, that the key-value storage of the blockchain has changes. This is not a concern for my use case, but in the Appendix, I propose a simple solution to handle this.

To expose confidential data to a participant that has the right to view data is complex task, if we want to achieve auditable reads and mitigate risk of TEE and blockchain related attack vectors. Before users can read data from a system that conforms to this protocol requires an initialization process. This initialization process uses a trusted third party for a one-time key generation, which results in a public encryption key stored on public ledger, and private decryption key shares that are shared among the network participants. This is done by using the participants' public keys to encrypt their share of the secret. This cryptosystem (ElGamal Threshold Encryption [48], [49]) enables the enclaves to use the public key to encrypt the confidential data within the enclave with the configured threshold parameters, and then return the encrypted response. This is essentially the only data that comes out of the enclave apart from status messages. Now the participant can request partial decryptions of the data from other participants using secure channels, and participants after validation the request, can send the

partial decryptions to the participant, who can combine the shares to get access to the data.

Depending on the configuration of threshold parameters, different number of peers would be needed to decrypt the data. If in a (t,n) threshold cryptosystem where n is the number of participants and t is the threshold number of decryption shares needed for decryptions, denial of services attacks are possible, if the number of malicious peers m $>$ n-t.

Although the approach uses a trusted third party to generate encryption parameters, it is important to highlight that for the reading of data to being, the blockchain must reach a consensus on the state of the public encryption parameters. If a participant challenges the "validity" of the third party, it can refuse to endorse the initializing transaction, or refuse to provide partial decryption.

To achieve audibility on the data reads, we can notice that no reading of sensitive data can go unnoticed, since at least the threshold number of participants need to validate and participate in the process. To physically persist an account of data access to the blockchain, we could forbid the use of blockchain queries to read this encrypted data, which means that one participant's endorsed transaction is enough to persist this information. We can assume that it is in the interest of at least one participant of the network to maintain audibility.

Lastly, I wanted to highlight the validation functions A and V do not provide any information about why a certain invocation failed, this is to make it hard, if not impossible to obtain sensitive data from attempting to invoke a lot of speculative transactions. Since the validation logic, as well as permission scheme is public, if a participant has no access to a resource, it cannot gain sensitive information about it. (If the developers of the prototype implementation are careless, it is technically possible to have a data model and validation logic that could leak some information about data)

4.2 System and threat model

I identified two main potential adversaries to the system.

Firstly, a participant of the network, meaning a party that actively participates in the consensus mechanism, validating changes to the shared data, even reading some data through the reading protocol. This type of attacker could have different goals, but the three main attacking goals could be to

1. Disrupt the service, hindering the participants of the network to reach a consensus.
2. Attempt to gain access to confidential data
3. Attempt to forge and share faulty data

Secondly, an external party, that is not part of the network, therefore not having access to data shared in the system. This party could benefit from stealing and potentially selling the confidential data.

4.2.1 System model

Here is a very high-level view of the system components that interact with each other during the operation of the system.

Event Submitter: A user creating an event invocation to achieve some goal with the system (submit data, request data ect...), and using it's credentials to interact with a client. The interaction is private by the nature of the interaction.

Submitting Client: A client of the blockchain system using the credentials by the submitter and its identity to forward to invocation requests of a submitter to an endorsing peer. The data of the request is encrypted with the public key of the processing enclave, so the Endorsing Peer cannot read the arguments of the request.

Endorsing Peer: An endorsing peer is a node of the network responsible for execution, validation of transaction requests. It receives a transaction request encrypted, which it cannot decrypt, and it forwards this request to the enclave that resides in the Peer itself. After the enclave return a response, it safely returns the response. A Peer contains a database, which the state of the ledger.

Enclave: The enclave is the trusted execution environment of the system, it receives encrypted transaction request, which it decrypts then processes. After processing it encrypts the response with the public key of the client, then returns a response.

Ordering Service: A node of the network that receives endorsed transactions, verified by signatures of the validators, then its goal is to batch these transactions into blocks that will be sent to other participants. The order of the transactions is defined by this component, and participants of the network learn the newly agreed upon ledger state from this component.

Observing Participant: This is an entity that receives the new proposed world state and after validating it, it can observe what it is. If another participant request data from it, it might refuse to answer.

Key Generator Agent: A trusted third party responsible for privately generating and uploading the public key for the encryption, as well as encrypting the key-shares with the corresponding person's public key.

4.2.2 Assumptions

To understand the context in which the threat model is defined, it is crucial to state assumptions I have about the system's environment and the behaviour of its users. It is reasonable to assume that the participants of the system do not want their confidential data to be exposed. I also assume that the computer nodes

of the network have some way of communicating with each other in a way that is sufficient for a HyperLedger Fabric network to work. I assume that participants of the network can create, store, and use their cryptographic private keys in a secure way, and most notably, that the used cryptographic primitives guarantee confidentiality by being *hard* to break computationally.

In this system we also assume that the used trusted execution environments work as intended and their secret keys are never exposed. I am aware that they are side-channel attacks and other vulnerabilities regarding TEEs and Intel SGX, but the proposed protocol does not rely on any specific implementation, but rather the concept of confidential computing.

4.2.3 Threat model

Speculative transactions: A malicious Event Submitter can use a Client to invoke speculative transactions on a malicious Peer. By doing so, it can observe what the Peer responds, and potentially extract information about system. Since validations don't output any information about the problem, it is difficult to extract information this way, but not impossible, based on the implementation.

Timing attacks: Analogous to the previous threat, Event Submitters might measure the response times of enclaves to assess which validation could have failed. This attach is very unlikely to work, since large enterprise software like HyperLedger Fabric networks are known to be hard to benchmark, and extractable information is very limited. Solution to the problem could be randomize the execution order of the validating function within the enclave.

Client leaking credentials: A potential threat to the integrity of the ledger state is a Client application that was modified to be malicious, recording and leaking the credentials of the owner Participant, or acting on its behalf.

Eavesdropping: This attack is only possible, if the chaincodes have application-level security issues, but as this is not unlikely, it is worth noting. To illustrate this, I construct a scenario where among other organizations an organization only has permission to do one thing, let's say to flip a binary bit stored on the ledger. Since the genesis state of the ledger is known, parties can essentially know the value of this variable just by observing whether an action by a participant resulted in a ledger state change. This can be mitigated by nonces in these extreme scenarios, but this is an application layer issue.

Denial of service: On of the most potent mode of attack for any component of the system is to use denial of service. Peers can refuse to endorse other transactions, the ordering service can become unreactive, halting the creation of blocks, Participants can refuse to send partial decryptions to other Participants, this way not helping anyone get access to sensitive data. This issue is critical, but

it can be mitigated using a correct endorsement policy, choosing good[1] threshold encryption parameters, and using multiple ordering service nodes.

Corrupt third party: If the trusted third party that generates the keys is corrupt, the threshold encryption scheme is broken. This is only a problem however, if the corrupt party is also colliding with one of the network participants, since in the context of a permissioned network, there access for the corrupt agent to read the encrypted data.

Wrong public key third party: When generating the the keys for the trehsold encryption scheme, the trusted third party uses the public key of each user to encrypt their share of the secret key. If the public key, that the third party uses for encryption is not correct, this would mean that the user won't have access to its private key share.

4.3 Cryptographic primitives

I now discuss the essential cryptographic primitives used in the proposed protocol. Important to note that a lot of trust is being rooted in the security of these cryptographic algorithms and cryptosystems. On a high level, the protocol can still be secure and function if the underlying cryptosystem is changed to a different one, but I made efforts to find the most optimal, state of the art specific cryptosystem to aid to protocol.

In the context of blockchains, the use of cryptography is essential from the tamperproof linearity of blocks to simply using the internet to privately communicate between the nodes. The cryptography of blockchains and its environment such as how parties can securely communicate with each other will not be discussed in detail as it is not in scope of this paper.

4.3.1 ElGamal encryption

In the core of confidentiality and privacy, there needs to be a way to encrypt and decrypt data, before storing it on the ledger or making it available to be read again. A simple way of doing this is using ElGamal Encryption [48]. This is a public-key asymmetric cryptosystem that is based on the Diffie-Hellman problem, and its difficulty to break them is as hard as to find discrete logarithms over finite fields. To describe the cryptographic bases of the algorithm, I quote Christian Cachin [50] pg:1

"The discrete logarithm problem (DLP) means, for a random $y \in G$, to compute $x \in \mathbb{Z}_q$ such that $y = g^x$. The Diffie-Hellman problem (DHP) is to compute g^{x1x2} from two random values $y_1 = g^{x1}$ and $y_2 = g^{x2}$. It is conjectured that there exist groups in which solving the DLP and DHP is hard, for example, the multiplicative

[1] Good: Meaning that a minority of malicous peers can not halt the system.

subgroup $G \subset \mathbb{Z}_P *$ of order q, for some prime $p = mq + 1$ (recall that q is prime). For example, $|p| = 2048$ and $|q| = 256$ for 2048-bit discrete-logarithm-based cryptosystems, which is considered secure today. Using the language of complexity theory, to say that a problem is hard means that any efficient algorithm solves it only with negligible probability."

Although ElGamal encryption is not explicitly used in the protocol, improvements of the algorithm are, and highlighting the basis of the cryptosystem is useful to have a deeper understanding of the used algorithms.

Here are the outlines of ElGamal based encryption algorithms.

Algorithm outline 1 ElGamal key generation

1: **procedure** GENERATEKEYS
2: Choose a large prime p such that $p = 2q+1$ where $q \in \mathbb{P}$
3: Find a generator function g of order q
4: Choose a random $x \in \mathbb{Z}_q$
5: Compute $ß = g^x$
6: **return** public key $< p,g\ ß >$, secret key $<x>$
4: **end procedure**

Algorithm outline 2 ElGamal Encryption

1: **procedure** ENCRYPTION(m, p, g $ß$)
2: Choose a random $k \in \mathbb{Z}_q$
3: Compute $c_1 = g^k \bmod p$
4: Compute $c_2 = mß^k \bmod p$
5: **return** ciphertext c $=< c_1, c_2 >$
4: **end procedure**

Algorithm outline 3 ElGamal Decryption

1: **procedure** DECRYPTION(c, p, x)
2: Compute $m = c_2\ c_1^{-x} \bmod p$
5: **return** m
4: **end procedure**

4.3.2 Threshold ElGamal Encryption

To make the decryption of an encrypted data extremely difficult for malicious party, we can leverage threshold cryptography. In essence, a ElGamal threshold

encryption (t, n) is a public key encryption schemed that enables the parties to have an encrypted data that only the cooperation of at least t parties can decrypt. This is a great way to ensure, that even if most of the system is compromised and somebody gets ungranted access to a file, it still needs at least t-1 other corrupt participants to get sensitive data.

Threshold ElGamal encryption is a modification on ElGamal encryption, where we use a secret sharing scheme to distribute the secret x among n participants. A common approach of doing so is using Shamir's secret sharing algorithm [51].

In my approach, a trusted third party generates the private keys and the key shares. It is possible to use distributed key generation [50], [52], but to implement the algorithm in a blockchain environment, where there is no trusted ledger is challenging. Computational and communicational overhead of such solutions is also something to consider.

4.3.3 Secured Threshold cryptosystem

The before mentioned algorithm is based on the Diffie-Hellman problem, but it is only secure against passive adversaries [50]. The way in which an active adversary can exploit this is quite simple. With the use of chosen ciphertext attack [53], if the attacker can have access to the decrypting oracle, and get the partial decryption of chosen cyphertext, it can gain information about the encryption. In the context of blockchains, every node of the network is technically a decrypting oracle, so even if I introduce a software solution by only allowing transactions to take place (to slow down attempt and introduce another barrier to brake), it will still be a potential threat to our system. To make the algorithm secure against an active adversary, we can use the work of Victor Shoup and Rosario Gennaro [49] which provides an improvement to algorithm, making it proven to be secure against chosen cyphertext attacks. They propose two cryptographic schemes, Threshold Diffie-Hellman 1 and 2. The algorithms are similar, but to quote [49] "*The first scheme, which we call TDH1 (for Threshold Diffie-Hellman), is secure assuming the hardness of the computational Diffie-Hellman problem [DH76]. The second scheme, TDH2, is secure under the stronger assumption of the hardness of the decisional DiffieHellman problem, but is more efficient than TDH1.*"

(In the prototype implementation, I use TDH2)

Algorithm outline 5 TDH2 Cryptosystem Encryption [49] Simplified

1: **procedure** ENCRYPT(m, h, g, L)
- ⇨ m: message
- ⇨ h: public key
- ⇨ g: generator
- ⇨ L: label

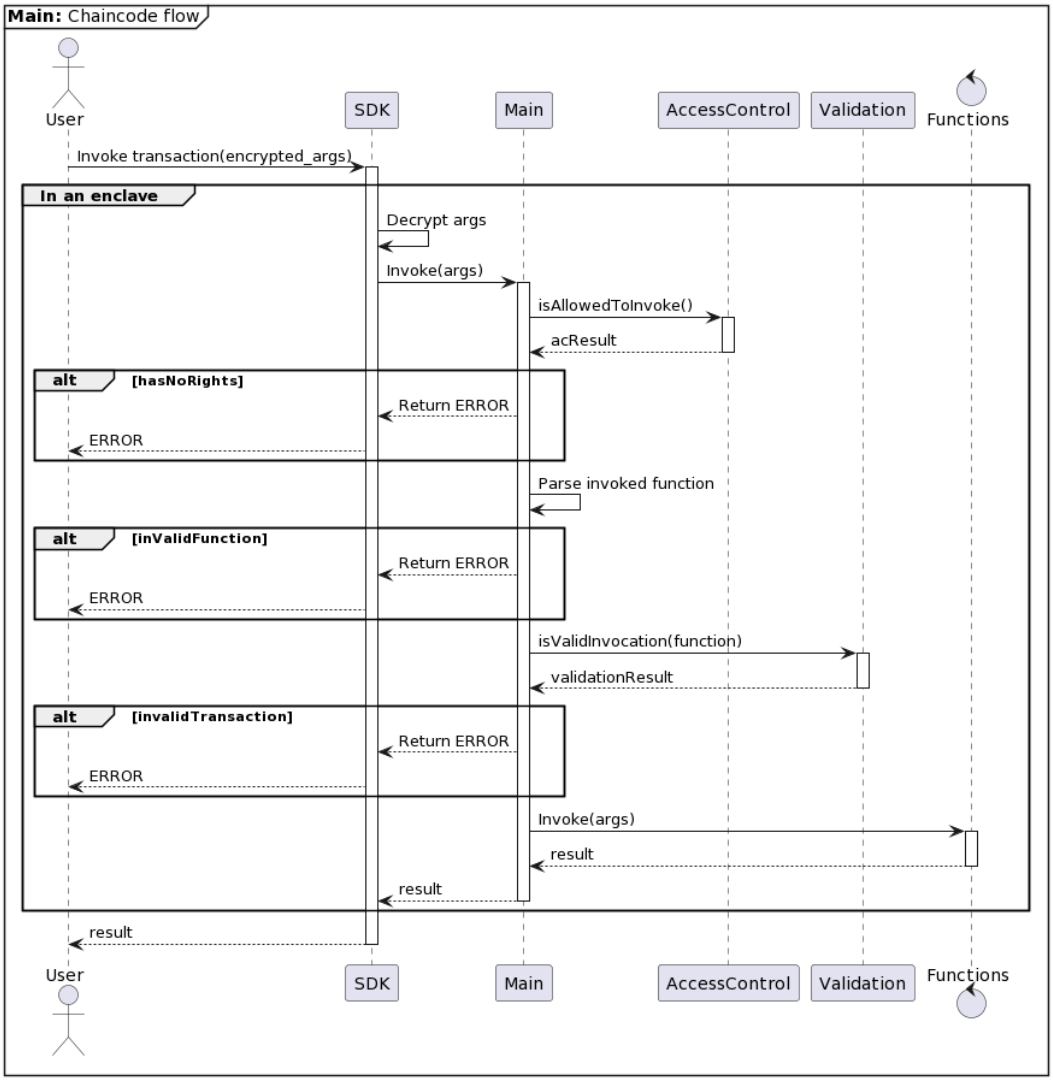
2: Generate random $r \in \mathbb{Z}_q$

3: Generate random $s \in \mathbb{Z}_q$
4: Compute c = $H_1(h^r) \oplus \quad m$
5: Compute $u = g^r$
6: Compute $w = g^r$
7: Compute $\bar{u} = g^{-r}$
8: Compute $\bar{w} = g^{-s}$
9: Compute $H_2(c, L, u, w, \bar{u}, \bar{w})$
10: Compute $f = s + re$
11: **return** $<c, L, u, \bar{u}, e, f>$
12: **end procedure**

4.4 Protocol outline

I know discuss the protocol in more detail, by examining the processes and sequences that happen during the operation of a network. Some of the sequence diagrams are not conventional in the sense that the lifetime of entities might seem strange at first. What these sequence diagrams are attempting to signal, is that in a blockchain environment, every valid transaction is eventually validated and executed on the nodes, and these components are also constantly "active".

Flow of the chaincode invocation



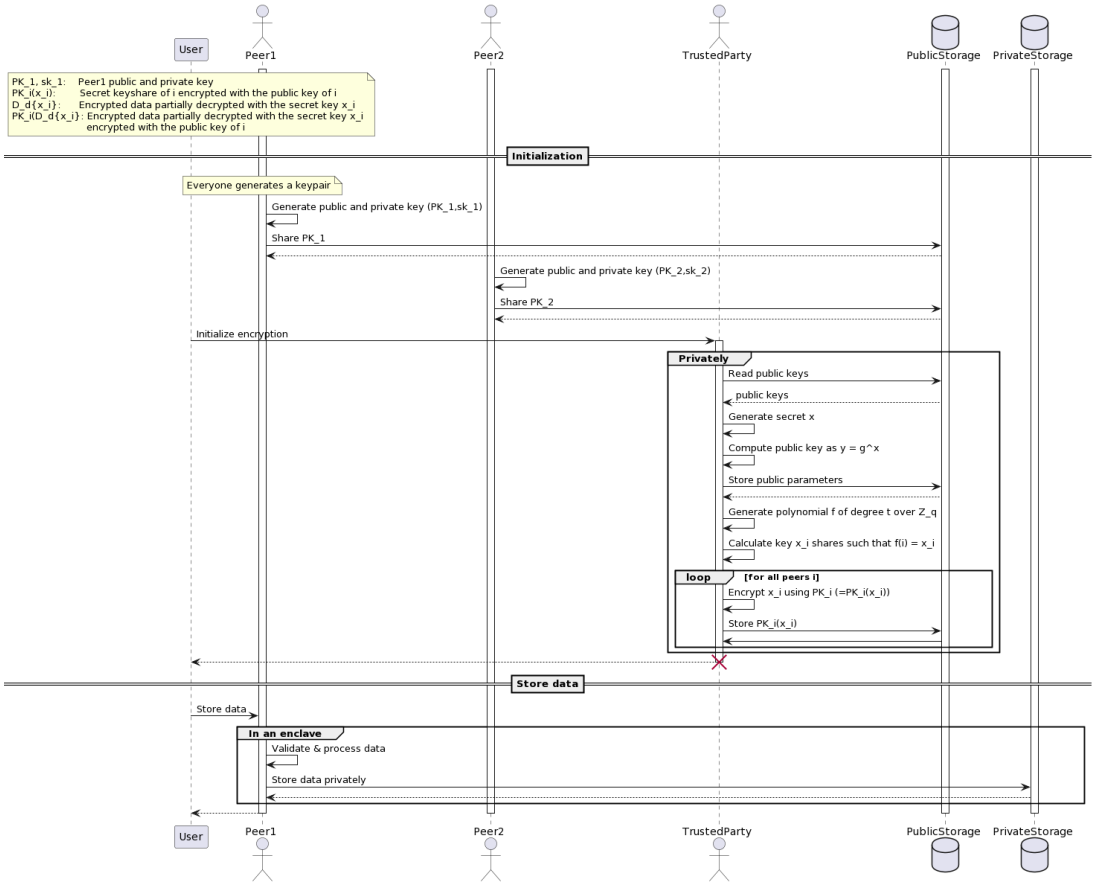
7. Figure Flow of the chaincode invocation

On this figure, we can observe how a chaincode invocation happen, and how the components of the chaincode interact with each other. This sequence starts with the action of a user, that wants to invoke a transaction on the blockchain. When the encrypted request reaches the enclave, it can use its enclave decryption key to decrypt the arguments of the request. After which the SDK will forward the arguments to our chaincode, which is done by invoking the invoke function of the chaincode with the arguments. My protocol describes that there should be an access control scheme implemented in the chaincode that strictly only returns the information, whether the invocation is allowed to happen based on the current access control scheme and the invoking user. If this validation fails, the error is returned to the user. (Note that the error message does not travel within the enclave to the user, but first to the SDK, then the message is forwarded from the SDK to the user, but this does not happen in an enclave) If the validation

succeeds, the chaincode parses the invocation arguments and the actual function that is being called, and if the called function exists, it calls a validator function, that returns whether the requested transaction can happen or not, based on the predefined domain specific validation. If the invoked function does not exsist, or the domain specific validation fails, it returns the message "error" to the user. If the validation succeeds, the chaincode executes what the original invocation requested, then returns the result to the SDK, then the client. Important note that the Users don't directly interact with the SDK, but rather though the use of a Client and a Peer node, but this is purposefully omitted from the diagram for the sake of illustrating the transaction flow.

Initializing encryption

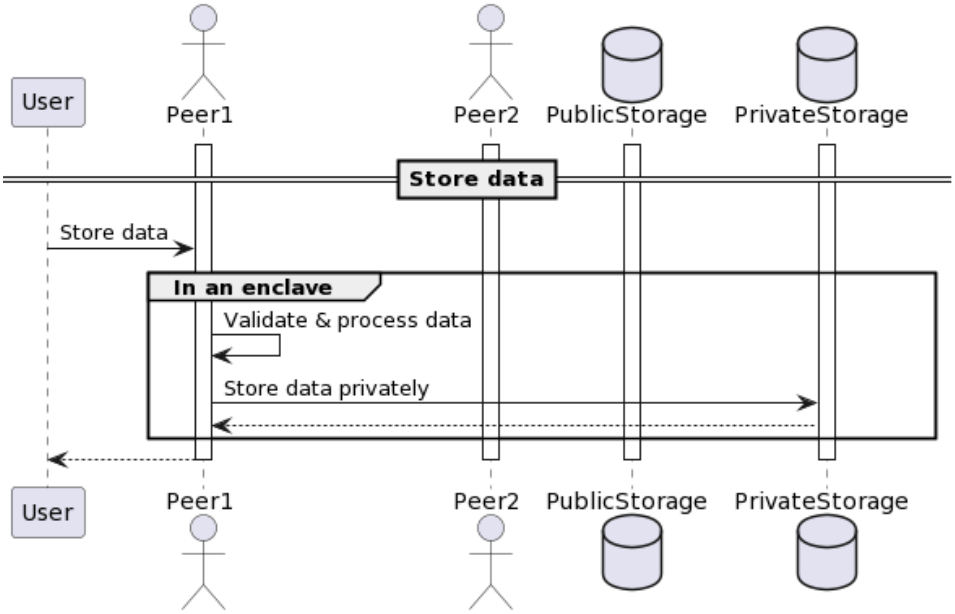
For the users of the system to gain the ability to read out sensitive data from the blockchain, it needs to go trough an initialization process. The process starts with the users privately generating a public and a secret key, that they store on the public ledger (only the public key). After every participant shared their public key, they can agree on using a trusted third party, that will use a centralized key generation scheme to create a threshold cryptographic system with the preconfigured threshold parameters, then split the keys using secret sharing algorithm [4.3.2], then use the participants public key to encrypt their share of the secret key, then finally share these encrypted key shares on the public ledger and also share the public encryption parameters publicly. The initialization process is done here, users can either start using the network, or refuse to participate, if they assume that the third party was corrupt.



8. Figure Initialize encryption

Storing confidential data

Submitting a valid transaction that extends the ledger state with new data is simple. Note that the initialization of the encryption is not a requirement for data to be stored and handled, the encryption initialization is only requirement for reading out sensitive data from the ledger. This sequence diagram abstracts away the low-level details of the chaincode invocation, and it aims to show how storing of data works on a high level. After a User submits a request to store a data on the network, it uses a Peer that it has access to invoke a transaction. The Peer will use its enclave to validate the request in ways that was detailed previously, and then store the data privately. The only information the User receives from the enclave whether the transaction was successful, or not.

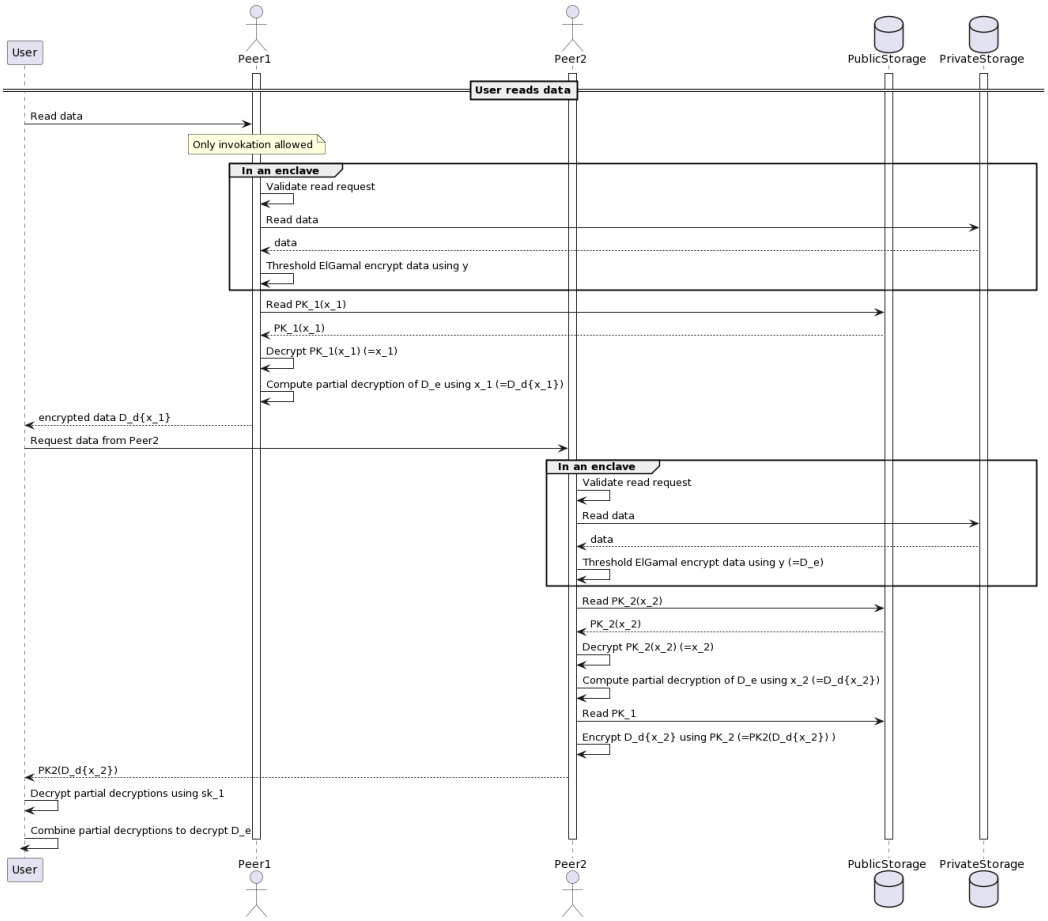


9. Figure Store data

Reading confidential data

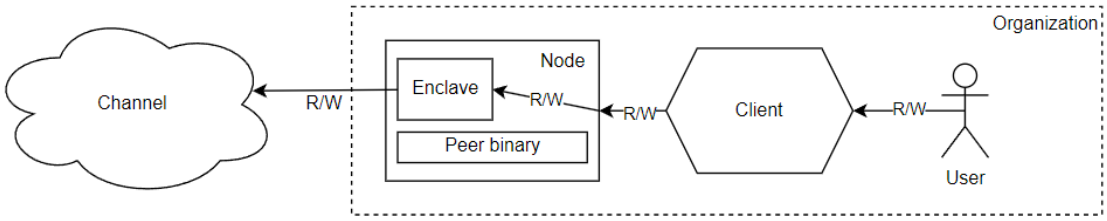
To read out sensitive from the ledger, a user needs to collaborate with other participants of the network. On a high level, threshold decryption has to be used by the user to gain access to the data. The number of collaborating participants needed for the decrypting the data entirely depends on the configuration of the threshold cryptography. This sequence diagram displays the reading process with a threshold parameters (t, n) = (2,2). The user begins the process by request data from its local Peer1. It uses its enclave to validate the read request, and if the validation succeeds, it uses the public encryption key of the threshold cryptosystem to encrypt the data. The encrypted data is then returned to the Peer who reads out the publicly available encrypted key share of Peer1, and after decrypting the secret share, it can use this secret to partially decrypt the data. After this is successfully done, (or in parallel, the order of data requests are not relevant) the User can repeat the read request with Peer2, that is from a different organization (not displayed on the diagram). Peer2 will use Peer1's public key to safely transfer the partial decryption by itself to Peer1. After our user gathered

at least the threshold number of partial decryptions, it can combine these to get gain access to the encrypted sensitive data.

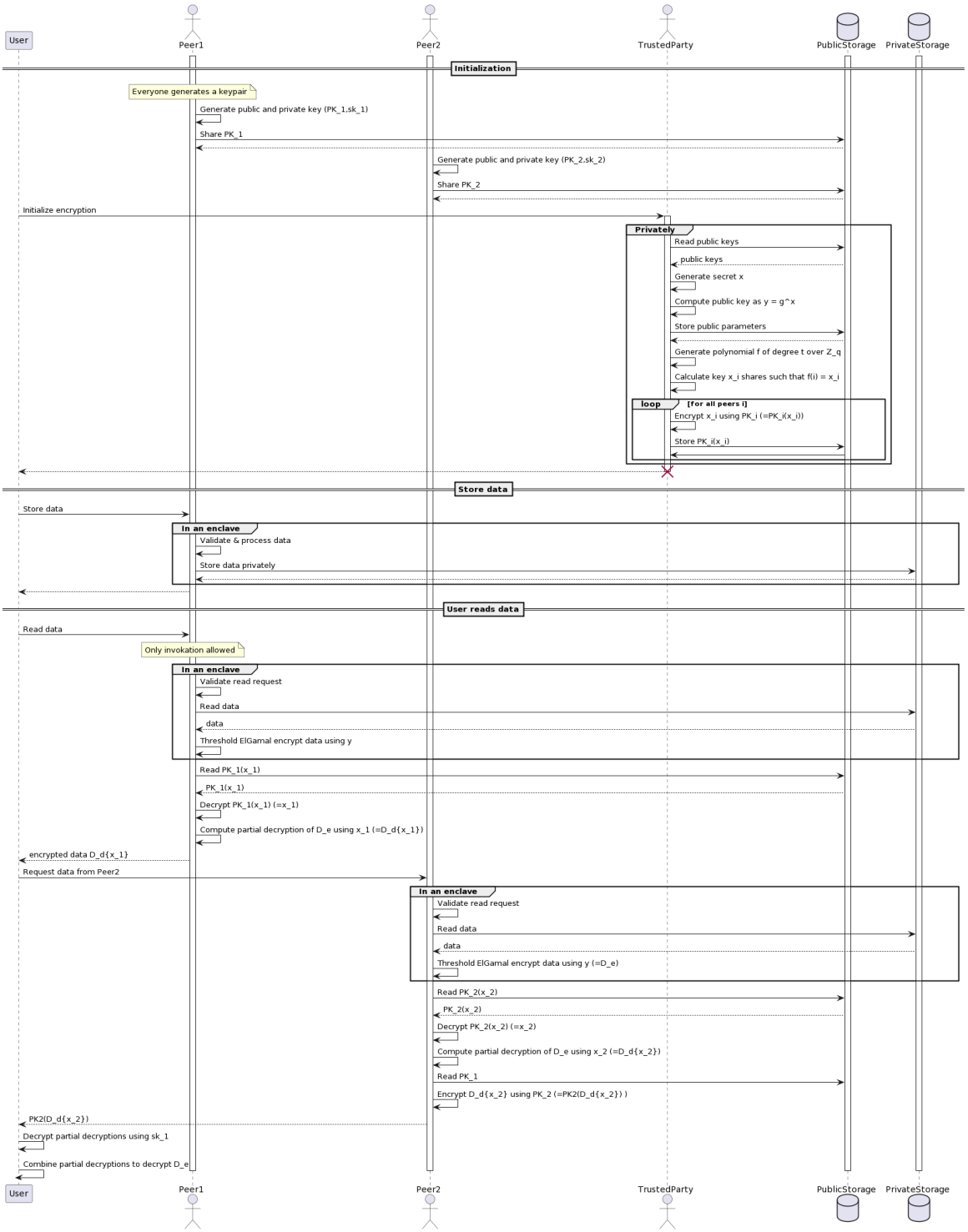


10. Figure Read data

High level view of interaction with the blockchain:



11. Figure A user interacts with the ledger



12. Figure Full example using 2 parties.

5 Implementation and evaluations

5.1 Implementation

5.1.1 Overview

The prototype implementation targeted the proposed protocol, as well as the example use case detailed in the running example section. The prototype consists of a

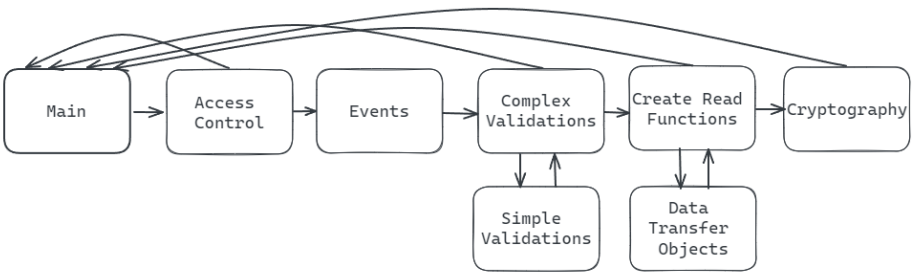
1. Hyperledger Fabric Private Chaincode, which implements the protocol as well as the example use case, written in C++ relying on the Intel SGX C++ interface and the FPC shim.
2. CLI client written in Go to interact with the network.
3. Integration testing scripts to automate testing of functionality, also used for regression testing. (Using the Hyperledger Fabric integration testing library)

The majority of the implementation is the chaincode that implements the protocol. The FPC shim is the interface that we can use to interact with the blockchain ledger as well as the enclave. The starting point of every FPC chaincode from the developer's point of view is the invoke function. Which has the following signature:

```
int invoke(uint8_t* response, uint32_t max_response_len, uint32_t* actual_response_len, shim_ctx_ptr_t ctx);
```

The argument names are verbose, what needs to be noted is that the transaction data can be extracted from the ctx struct, which is also our way of interacting with the blockchain.

The written prototype is a functional style C++ implementation, and essentially stateless. The structure of the code can be well illustrated with a diagram.



13. Figure Component and program flow

To understand the program flow from a different view, take a look at the sequence diagram [Figure 7].

5.1.2 Implementation details

Now I discuss the implementation more in detail, with examples from the code base. The repository with all the chaincodes, clients and tests with bootstrapping instructions can be found at the GitHub repository created for this project.

Chaincode

The chaincode is invoked by a participant of the network, and the invoke C++ function will be called. In this function, my implementation parses the transaction arguments as well as the function, that it tries to invoke. After this, it checks whether the invoking entity has the permissons to use that function. In this implementation and use case, the functions that the entites try to invoke are called events, which essentially are the *endpoints* of the application. The access control checking component uses the public ledger state to get the current permisson scheme, parses this information, then return a boolean representing whether the caller has the permissions to invoke that event. If this function returns false, the enclave returns to the caller with no information, apart from the text: ERROR. This is not due to a lack of effort in returning verbose error messages, but to make speculative execution attack ineffective.

```
if(!isAllowedToInvoke(function_name,tx_creator_name_msp_id, tx_creator_name_dn, signature, ctx)){
    LOG_DEBUG("DCBTEECC: Not allowed to invoke");
    result = "ERROR";
    prepareResult(result, response, max_response_len, actual_response_len);
    return 0;
}
```

14. Figure Access control code fragment

If the access control component validates that the user has the right to invoke an event, the program continues by parsing the function arguments, and running complex validation to check whether this operation is allowed or not. This validation first checks whether the invocation arguments are valid, for instance whether a personBorn event was called with a valid SNN number, a valid date, and then it uses the chain state to validate more complex business logics. For instance, for the life insurance entity has the requirement for issuing a life insurance, that the person in question has a health examination result, that states that the person is healthy. (Notice here that the life insurance entity does not gain any more information about the person's confidential record than what is necessary, and already agreed upon, by creating the chaincode)

```
bool validIssueLifeInsurance(std::string id, std::string taj, std::string from, std::string to, int cost, int payment, shim_ctx_ptr_t ctx) {

  if (!isValidTaj(taj) ||
    !isValidId(id) ||
    !isValidDate(from) ||
    !isValidDate(to) ||
    !isValidCost(cost) ||
    !isValidPayment(payment) ||
    !idExists(id, ctx) ||
    !isAlive(id, ctx) ||
    !isHealthy(id, ctx)
  ) {
    return false;
  }

  return true;
}
```

15. Figure Validation function for an event

```
bool isHealthy(const std::string& id, shim_ctx_ptr_t ctx){

    health_examination_t health_examination = getHealthExamination(id, ctx);
    if(health_examination.id == id
        && health_examination.diastole > 50
        && health_examination.diastole < 100
        && health_examination.systole > 90
        && health_examination.systole < 150){
        return true;
    }

    return false;
}
```

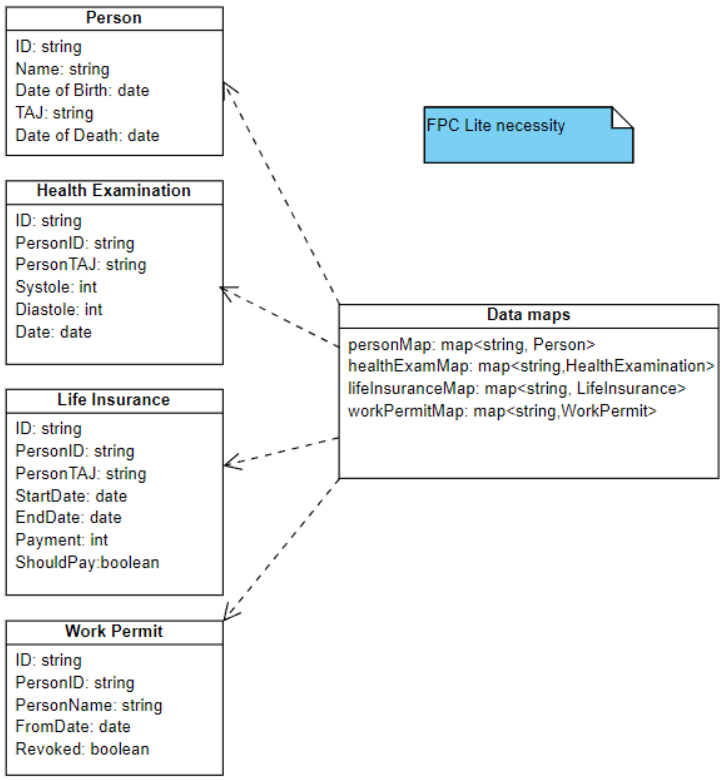
16. Figure Stateful validation

In this example, the blood pressure measurement of a person is used as a proxy for being „healthy” [2]. If these validation functions fail, only the text: ERROR is return the invoking client.

If the validation was successful, the program continues by actually doing what the user wanted to achieve, like issuing a life insurance to the ledger privately.

For storing data the implementation uses a data model represented by diagram below. This is done by creating a struct for the data, initializing its fields, than getting the current state of the ledger, inserting the newly created data to the ledger state, then using marshalling to turn the ledger state into a text format, then privately storing it on the ledger.

[2] *This is not medical advice.*



17. Figure Prototype's data model

The create and read functions follow a similar path to how one might save a new work permit.

```
std::string putWorkPermit(work_permit_t work_permit, shim_ctx_ptr_t ctx){
    // check if datamap exists
    uint32_t datamap_bytes_len = 0;
    uint8_t datamap_bytes[MAX_VALUE_SIZE];

    get_state(DATAMAP_KEY, datamap_bytes, sizeof(datamap_bytes), &datamap_bytes_len, ctx);
    if (datamap_bytes_len == 0)
    {
        LOG_DEBUG("datamapCC: datamap does not exist");
        putDataMap(ctx);
    }

    // get datamap datamap from json
    data_map_t the_datamap;
    unmarshal_data_map(&the_datamap, (const char*)datamap_bytes, datamap_bytes_len);

    // insert or update a work_permit
    the_datamap.work_permits.insert(std::pair<std::string, work_permit_t>(work_permit.id, work_permit));

    //marshal datamap to json string and save new state
    std::string datamap_json = marshal_data_map(&the_datamap);
    put_state(DATAMAP_KEY, (uint8_t*)datamap_json.c_str(), datamap_json.length(), ctx);
    return OK;
}
```

18. Figure Save data

First I obtain the data map holding the ledger state, convert it from a json to a cpp structure, inserting the new object, then privately save the modified json string to the ledger.

One of the functionalities of my system is the ability to securely extract sensitive data from the blockchain, if the entity has the right to do so, and at least the threshold number of participants allow the read. To read a sensitive data, the requester invokes the chaincode as a regular transaction, and if the usual validations pass, the enclave will use the stored and initialized public threshold encryption key to encrypt the data and return it to the user.

One of the most challenging parts of the implementation was to implement the TDH2 encryption algorithm in a C++ chaincode. Normally, a developer that is not educated on the highest level should not attempt to implement cryptographic protocols, since this is poses high security risk. Even if somebody had to implement cryptographic algorithms, the person should rely on using audited and tested cryptographic libraries for the cryptographic primitive implementation. This unfortunately was not entirely possible in my case, since the enclave code compilation is not a standard C++ program compilation, we can not have access to default libraries, some of the standard libraries ect..., which means that most cryptographic libraries are not integratable with the chaincode, especially not C++ implementations of the TDH2 [50] encryption scheme. In the end, I implemented the TDH2 cryptosystems encryption part on the ledger while using the OpenSSL C++ library, and porting a Python implementation of the algorithm [54] to C++. The resulting encryption algorithm is functional, but it has not been externally audited for security, and it serves as demonstration tool for the prototype implementation, as the algorithm can be easily followed and understood while reading the code.

For illustration purposes, here is a fragment of the encryption algorithm.

```
size_t max_len = std::max(BN_num_bytes(m_bn), BN_num_bytes(h_bn));
unsigned char * m_bin = static_cast < unsigned char * > (OPENSSL_malloc(max_len));
unsigned char * h_bin = static_cast < unsigned char * > (OPENSSL_malloc(max_len));
BN_bn2binpad(m_bn, m_bin, max_len);
BN_bn2binpad(h_bn, h_bin, max_len);

// Perform the XOR operation byte-wise
unsigned char * result_bin = static_cast < unsigned char * > (OPENSSL_malloc(max_len));
for (size_t i = 0; i < max_len; i++) {
    result_bin[i] = m_bin[i] ^ h_bin[i];
}
// Convert the result back to a BIGNUM
BIGNUM * result_bn = BN_bin2bn(result_bin, max_len, nullptr);
```

19. Figure Fragment of TDH2 implementation

The implementation of the algorithm, as well as the entire project can be found on GitHub [55].

Other parts of the chaincode implementation included creating JSON marshal and unmarshal functions, that can be used to transform C++ structures into

plaintext JSON representation, and vice versa. Writing these codes could have been automatized.

```
std::string marshal_person(person_t * person) {
    JSON_Value * root_value = json_value_init_object();
    JSON_Object * root_object = json_value_get_object(root_value);
    json_object_set_string(root_object, "id", person -> id.c_str());
    json_object_set_string(root_object, "taj", person -> taj.c_str());
    json_object_set_string(root_object, "name", person -> name.c_str());
    json_object_set_string(root_object, "birth_date", person -> birth_date.c_str());
    json_object_set_string(root_object, "death_date", person -> death_date.c_str());
    char * serialized_string = json_serialize_to_string(root_value);
    std::string out(serialized_string);
    json_free_serialized_string(serialized_string);
    json_value_free(root_value);
    return out;
}
```

20. Figure Marshalling code of a person

Lastly in the chaincode implementation, there are the functions that are responsible for parsing the access control scheme that a user provides to initialize said scheme, and some utility functions to make to code more readable like the function that prepares the enclave to return a result.

Go Client

To interact with the blockchain, we can use the interface of the Smart Client made by the Hyperledger Foundation. This makes the encryption of the invocation arguments and the decryption of the results hidden from the user and the developer. The use of the Client was to aid the process of automated testing, where the Go client makes the connection with the channel, then invokes the transactions.

```
func main() {
    ccID := os.Getenv("CC_ID")
    logger.Infof("Use Chaincode ID: %v", ccID)

    channelID := os.Getenv("CHAN_ID")
    logger.Infof("Use channel: %v", channelID)

    // get network
    network, _ := utils.SetupNetwork(channelID)

    // Get FPC Contract
    contract := fpc.GetContract(network, ccID)
```

21. Figure Initialize connection with the chaincode

```
// Iterate through the transactions and invoke them
for _, tx := range transactions {
    result, err := contract.SubmitTransaction(tx.Name, tx.Args...)
    logger.Infof("Invoking %s %s ", tx.Name, tx.Desc)
    if err != nil {
       logger.Fatalf("Something went wrong: " + err.Error())
    }

    if (strings.Contains(string(tx.Desc), "succeed")) {
        logger.Infof("Result: %s", string(result))
    }else{
        logger.Infof("Transaction failed: %s", string(result))
    }

}
```

22. Figure Code fragment to automate transaction invocation

Bootstrapping scripts

I used the integration testing libraries of FPC to bootstrap a Hyperledger Fabric network with private chaincodes enabled, then invoke the Go clients to submit transaction, to finally tear down the network. The testing bash script can also be invoked with arguments to selectively run the testing suites.

5.2 Testing

To accelerate development as well as to protect functionality I implemented a 4 different automated testing scripts. These are essentially bash scripts that build on top of the integration testing framework of the Hyperledger Fabric integration testing framework. Bootstrapping a Hyperledger Fabric network, especially with Private Chaincode enabled, installing the Chaincode, then manually invoking the now fully working blockchain with my test cases could take upwards of 30 minutes, and I also had to verify the results. Apart from being impractical, it is also a common way people introduce bugs into existing code base, by omitting regression testing, because it simply takes too much time. For this reason, I decided early in the development, that I will utilize a test-driven development inspired approach of writing integration tests first, then the Chaincode.

Luckily the integration testing framework is also well equipped to handle Fabric Private Chaincode enabled blockchains, and apart from helpful bash scripts, we can also use the Fabric Smart Client [56] written in Go. With these tools, I created the following testing suites.

Functionality testing suite: In this testing suite, the so called happy path of the system is being tested. In this example use case of the protocol implementation, we have to entities interacting with the blockchain by essentially submitting events to the blockchain. For instance the state governed entity makes a Chaincode invocation to register that a new person was just born. Even for this simple event to be successfully accepted to the ledger state, a significant amount of validation have to be made. In the person born example, we have to validate that the invoking entity has the rights to invoke this Chaincode function, that the arguments are valid, like age of birth is a valid date, ID number is not already

existent, the name is valid ect. A person being born is one of the simplest invocation validation wise, when a work permit is being issued, we also have to check whether the person has a life insurance, to name a complication.

The aim of the functionality testing suite is to provide valid argument data, that will not fail any of the validations, therefore testing the end functionalities of the Chaincode. By end functionalities, I refer to data being stored on the ledger, in the context of the person being born event.

```
INFO [Functions] main -> Use Chaincode ID: DCBTEE_test
INFO [Functions] main -> Use channel: mychannel
0 UTC - cryptosuite.GetDefault -> INFO No default cryptosuite found, using default SW implementation
INFO [Functions] main -> Public encryption parameters: Threshold(k/n): 3/5 Group generators(G/G_hat): 696045/1445688, Group order: 2
INFO [Functions] main -> Invoking initAccessControl transaction to initialize access control: should succeed
INFO [Functions] main -> Result: OK
INFO [Functions] main -> Invoking initEncryption transaction to initialize encryption parameters: should succeed
INFO [Functions] main -> Result: OK
INFO [Functions] main -> Invoking PersonBorn transaction to create person (Alice): should succeed
INFO [Functions] main -> Result: OK
INFO [Functions] main -> Invoking IssueHealthExamination transaction to create health record for (Alice): should succeed
INFO [Functions] main -> Result: OK
INFO [Functions] main -> Invoking IssueLifeInsurance transaction to create life insurance for (Alice): should succeed
INFO [Functions] main -> Result: OK
INFO [Functions] main -> Invoking IssueWorkPermit transaction to create work permit for (Alice): should succeed
INFO [Functions] main -> Result: OK
INFO [Functions] main -> Invoking hasWorkPermit transaction to check if (Alice) has work permit: should succeed
INFO [Functions] main -> Result: OK: bd812e141df735826f41a4fef41108f429877246d04150fd4c79fba0207dc794 1128725 1967023 14626 18511
INFO [Functions] main -> Invoking PersonDie transaction to register (Alice) as dead: should succeed
INFO [Functions] main -> Result: OK
```

23. Figure The results of the function testing suite

To verify that the threshold encryption work, we can utilize an already ready implementation [54] of the algorithm, to decrypt the data.

In another test run, the enclave return the response:

```
main -> Invoking hasWorkPermit transaction to check if (Alice) has work permit: should succeed
main -> Result: OK: 57fd4ed6d69189cf1c398210798ca63c4306c4c6c42f3cb110cbeb159d37e233 3821773 3099428 12947 21234
```

Inputting this result into the decryption algorithm (where we have artificially compute partial decryptions, we successfully get the original message back.

```
print(bcolors.WARNING + "[+] Encrypted message: {}".format(encryption))

# get partial results
d = [get_partial(encryption, key, g, p, q) for key in selected_keys]

# reconstruct secret to check whether shamir share works
print("[+] Reconstructed secret: {}" .format(combine_key_shares(selected_keys, q)))

# reconstruct message out of partial results
message = combine_shares(encryption, verificationKey, d, p, q)
print(bcolors.OKGREEN + "[+] Decrypted message: {}".format(message) + bcolors.ENDC)
```

✓ 0.0s

```
[+] Encrypted message: ('57fd4ed6d69189cf1c398210798ca63c4306c4c6c42f3cb110cbeb159d37e233', 3821773, 3099428, 12947, 21234)
[+] Reconstructed secret: 15972
[+] Decrypted message: has work permit
```

24. Figure Successful threshold decryption

Validation testing suite: In this testing suite, the tested components of the architecture are the ones implementing the validation logic. Not to be confused by access control, validation logic is at the core, a predicate that determines whether a Chaincode invocation with its arguments are valid or not in the void of entities and permissions. To continue the example from the previous testing suite, validation logic only tests whether an issue health examination function

invocation was invoked to an existing person, with a valid date, and that the result of the health examination, namely the systolic and diastolic??? measurements of the person was in a medically acceptable range. Important to mention that this aspect of the protocol implementation is critical to well-functioning system. Zooming out of the details, these validation functions will determine the state of the ledger by enabling or denying transactions to happen. Since users may not be able to read confidential data on the ledger, they must root their trust in the publicly agreed upon chaincodes as well as the security of the protocol.

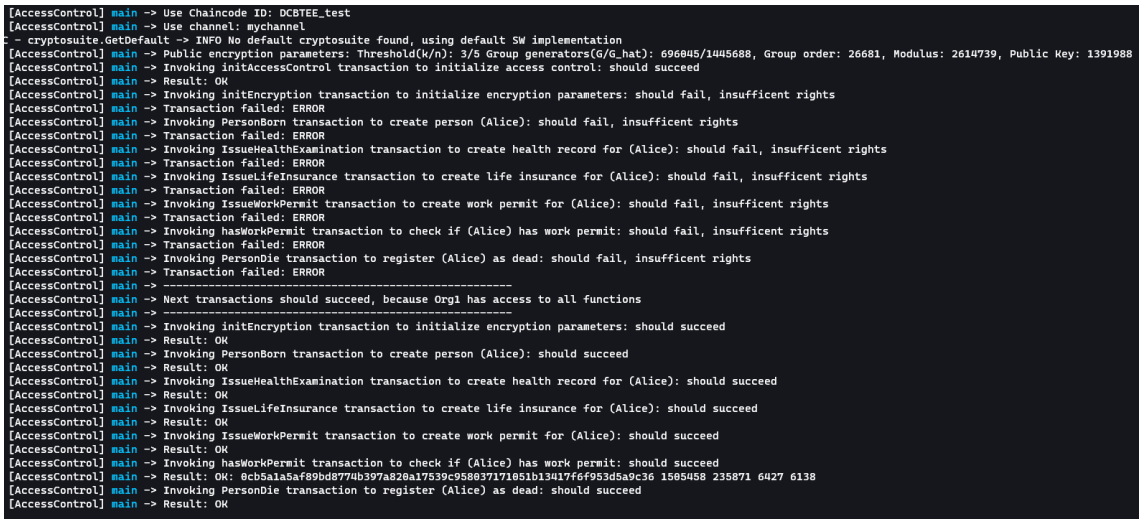
```
[VALIDATION] main -> Use Chaincode ID: DCBTEE_test
[VALIDATION] main -> Use channel: mychannel
- cryptosuite.GetDefault -> INFO No default cryptosuite found, using default SW implementation
[VALIDATION] main -> Public encryption parameters: Threshold(k/n): 3/5 Group generators(G/G_hat): 696045/1445688, Group order: 26681, Modulus:
[VALIDATION] main -> Invoking initAccessControl transaction to initialize access control
[VALIDATION] main -> Result: OK
[VALIDATION] main -> Invoking initEncryption transaction to initialize encryption parameters
[VALIDATION] main -> Result: OK
[VALIDATION] main -> Invoking PersonBorn transaction to create invalid person: should fail
[VALIDATION] main -> Transaction failed: ERROR: Invalid request
[VALIDATION] main -> Invoking PersonBorn transaction to create valid person (Alice): should succeed
[VALIDATION] main -> Result: OK
[VALIDATION] main -> Invoking PersonBorn transaction to create valid person (Bob): should succeed
[VALIDATION] main -> Result: OK
[VALIDATION] main -> Invoking IssueWorkPermit transaction to create work permit for invalid person
[VALIDATION] main -> Transaction failed: ERROR: Person does not exist or is not eligible for work permit
[VALIDATION] main -> Invoking IssueWorkPermit transaction to create work permit for valid person (Alice) should fail, no life insurance
[VALIDATION] main -> Transaction failed: ERROR: Person does not exist or is not eligible for work permit
[VALIDATION] main -> Invoking IssueLifeInsurance transaction to create life insurance for invalid person should fail
[VALIDATION] main -> Transaction failed: ERROR
[VALIDATION] main -> Invoking IssueLifeInsurance transaction to create life insurance for valid person (Alice) should fail, no health record
[VALIDATION] main -> Transaction failed: ERROR
[VALIDATION] main -> Invoking IssueHealthExamination transaction to create health record for invalid person should fail
[VALIDATION] main -> Transaction failed: ERROR: Invalid request
[VALIDATION] main -> Invoking IssueHealthExamination transaction to create health record for valid person (Alice) should succeed
[VALIDATION] main -> Result: OK
[VALIDATION] main -> Invoking IssueWorkPermit transaction to create work permit for valid person (Alice) should fail, no life insurance
[VALIDATION] main -> Transaction failed: ERROR: Person does not exist or is not eligible for work permit
[VALIDATION] main -> Invoking IssueLifeInsurance transaction to create life insurance for valid person (Alice) should succeed
[VALIDATION] main -> Result: OK
[VALIDATION] main -> Invoking IssueWorkPermit transaction to create work permit for valid person (Alice) should succeed now
[VALIDATION] main -> Result: OK
[VALIDATION] main -> Invoking IssueHealthExamination transaction to create health record (unhealty) for valid person (Bob) should succeed
[VALIDATION] main -> Result: OK
[VALIDATION] main -> Invoking IssueWorkPermit transaction to create work permit for valid person (Bob) should fail, no life insurance
[VALIDATION] main -> Transaction failed: ERROR: Person does not exist or is not eligible for work permit
[VALIDATION] main -> Invoking IssueLifeInsurance transaction to create work permit for valid person (Bob) should fail, not healthy
[VALIDATION] main -> Transaction failed: ERROR
[VALIDATION] main -> Invoking PersonDie transaction to invalid person dies, should fail
[VALIDATION] main -> Transaction failed: ERROR: Person does not exist or already dead
[VALIDATION] main -> Invoking PersonDie transaction to valid person (Bob) dies, should succeed
[VALIDATION] main -> Result: OK
[VALIDATION] main -> Invoking IssueHealthExamination transaction to create health record (healthy) for valid person (Bob) should fail, dead
[VALIDATION] main -> Transaction failed: ERROR: Invalid request
```

25. Figure Validation testing suite

Although it does not test every combination of valid and invalid arguments and ledger state, it verifies that the most important validation functionalities are at place. The enclave only return error messages, to demonstrate the cause of the error for development purposes only, this must be turned off for a production use case.

By reading the output line by line, we can read what the client attempts to do, and the result of the attempt.

Access control testing suite: In the access control testing suite, the tested components decide whether the entity making a transaction invocation or query has the rights to the action. This happens before validating arguments of the invocation, meaning if a life insurance entity attempted to register an invalid health examination result for a non-existing person to the ledger (which it does not have rights to do so) the invocation would be first rejected for not having permissions to invoke certain Chaincode function.



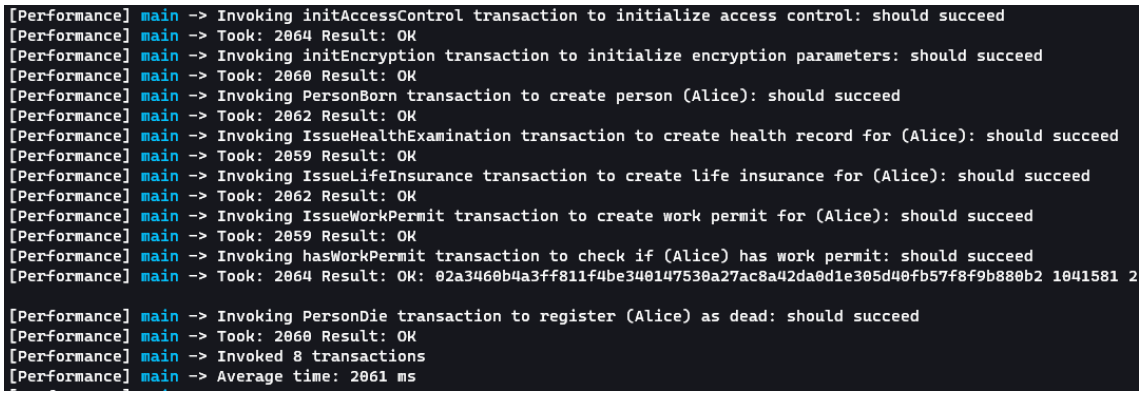
26. Figure The acces control tests

The results clearly show that whenever there was not sufficent permissions for a user, the transaction was rejected. (The access control scheme is initialized with a chaincode call at the beginning.)

5.2.1 Performance and scalability

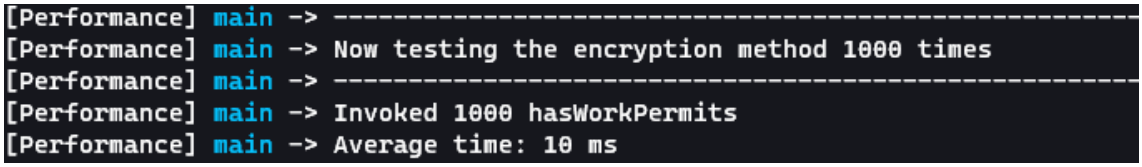
Performance testing suite: Performance and scalability is usually an important requirement, so it was essential for the testing suite to reliable and consistently measure response times from the system. The implementation is efficient in the sense that the change in access control scheme does not result in the need re-encrypt any data, which is a common drawback of cryptographic solutions [Section 2.2.1] Critical point to mention that the tests were run in a simulated SGX mode, this was done for multiple reasons, but one these reasons is that Intel SGX is no longer provided in consumer, and as mentioned earlier, bootstrapping an SGX enabled Hyperledger Ledger Fabric network is resource and time intensive, and not practical for testing. Since I am trying to measure the scalability of the algorithms, as long as we provide a fixed hardware and software environment (blockchain configuration) for the tested software, we can get useful insights. The performance testing suite includes the use of the functionality testing, by invoking all Chaincode functions, and measuring response times. It also includes invoking a person being born transaction several times to assess the effect of more data in the system to transaction speeds, but as shown in the data, there seem to be no slowing down of the system. (The prototype implementation uses a linearly scaling algorithm for storing and retrieving the data, but this is due to the limitation of the FPC Lite, and therefore of this prototype implementation.) Results of testing is shown here.

Invoking the main functions of the system results in a consistent ~2000ms transaction.



27. Figure Speed of invocations

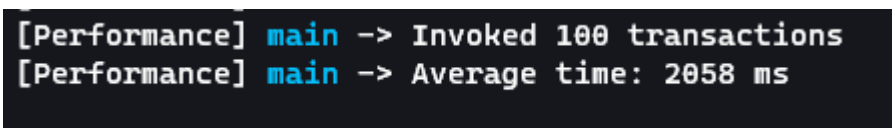
Next, I tested how the encryption of an object in the chaincode translates to transaction times. To test this, I invoke the encryption query 1000 times, and average out how much time it takes for the queries to finish.



28. Figure Benchmarking the encryption

The results showed a very consistent 10 ms execution time for the on chain validation and encryption. Note that invocations require more time queries in the Fabric ecosystem, because the unlike the invocations, the queries do not have to reach a consensus or get included in a block. The protocol suggests against enabling queries for the encryption process, because this might damage the audability requirement.

Finally, I tested how the data model reacts to the growing size of the data structure. For this, I invoked 100 transactions to create 100 persons on the ledger, and measured the response time of the system. Our assumption is that the prototype implementation's algorithm scales linearly with the amont of data in the system, because it has to parse longer and longer data structures as the system grows. During the testing, since we are using standard C++ code doing computationally non-expensive operations, I could not detect a worsening of performance. First transaction took 2056 ms and the last 2058 ms, but the longest transaction was the 6th one, with the duration of 2066 ms.



29. Figure Invoking 100 transactions

Testing environment: The testing was done in a Docker container without resource restrictions running on a Windows 10.0.19045, with 16GB of DDR3 RAM, an Intel® Core™ i7-9700K CPU, and minimal to no background activities and simulated enclaves.

5.3 Use case evaluation

After discussing the prototype implementation and the testing of my approach, I now evaluate the results based on the informal running example I detailed in [Section 3.1].

The example starts with a list of organizations that have different responsibilities, potentially conflicting interests, and the need to collaborate. This paper suggests the use of HyperLedger Fabric to manage the identities of different organizations and its participants. There are more than a hundred Fabric based applications[57] in the real world that require this functionality on a day-to-day basis, proving that it is usable for this purpose. Continuing with the example, there needs to be a way for the organizations to execute a domain specific logic, like validating data, which can also be handled by HF. What is also needed, is a shared data among participants, that they can append data to, for instance the result of a health examination, but only if the business logic allowed that to happen, this is also enabled by HF, and the blockchain nature of the framework, to serve as the shared database. The concept of the use case that requires more than the use of HF is the need for organizations to use others' data without actually having the ability to read that data. In other words, verify something that they have no access to. My approach uses trusted execution environments and FPC to establish a database that is encrypted for everyone, apart from the trusted enclaves. This makes sure that while trusted business processes implemented as chaincodes can use other organizations' data in a predefined way (only chaincodes that were agreed upon by all parties can be executed), general, ad-hoc access to sensitive date is not possible. In our example, the life insurance company needs a way to verify that a customer is "healthy" according to medical organization, without having sharing any specific sensitive data about the persons medical record. Since the medical record had to be put on the blockchain with a transaction, that validated domain specific aspects of the data, the life insurance company can be sure that what it reads out from the blockchain, as a result of a transaction has not been tampered with. With this, we also touched on the point of not having the ability to tamper data on the shared state, provided by the blockchain nature of the system as well as the fact that the only way to modify the blockchain is with commonly agreed upon chaincodes. What is critical to this use case, is the ability to dynamically change the access control of the organization. For instance the insurance company is sold to another owner, therefore the rights of the previous owner has to be revoked, and the rights should be transferred to the new

owner. My approach can easily handle this situation, by modifying the ledger data that stores the access control scheme. Upon changing this data, the users can not influence the state of the blockchain, since even if they refuse to acknowledge the new access control scheme, the blockchain nature of the architecture does not allow their version of the ledger state to be created, and data can not be extracted by them anymore, because every output of sensitive data requires other organizations to participate. To solve the issue of participants wanting to read data from the blockchain occasionally, we provided a chaincode implementation concept to only allow threshold encrypted data to ever leave the enclave, therefore no participant can read out data by itself, only if other parties collaborate to handle the read. This means that if an organization wants to read out a sensitive data like whether a participant has a work permit in the country or not, this can only be done with collaborating with other parties. This also achieves the challenge of aiding legal and compliance matters, because if the only way to read data is for participants to collaboratively use the blockchain, it is not possible to not leave a trace either on the blockchain, or elsewhere about the event. Unless every participant of the encryption process is malicious, this use case remains fulfilled.

My implementation of the approach provides an example for how all of these informal requirements that is the "running example" can be fulfilled.

5.4 Requirement evaluation

After informally evaluating the running example, I will continue with systematic assessment of the constructed requirement system [Section 3.2].

Fulfilling functional requirements

F1: The proposed approach uses HF detailed earlier, which verifiably follows a distributed ledger pattern, with the support for transaction validation and a ledger state managed by the consortium, with the use of a consensus mechanism.

F2: To handle personal data, we use HF's key values storage to store document like data to store attributes about a person, and its relationship with other data.

F3: The handle domain specific validation logic for the creation, update, and deletation of data, we use chaincodes with the design explained in [Section 4.1]. Since we are using one HF channel in the approach, the key value storage can only be modified/read by verified FPC chaincodes, if we require implementation to only allow validated access, there is no way for the data to be edited in different ways, which is a guarantee of the FPC.

F4: To allow the dynamically updatable authoritative data access, we build on the guarantee, that if the implementation satisfies F3, the access control scheme should also be only modified if a validation logic is satisfied, manifesting in form of a chaincode. More specifically, if we established, that we can store data that has to undergo a validation process, we can confidentially use this in the context

of the blockchain to store authoritative data access schemes, and the machine parsable descriptions thereof.

Confidentiality requirements

C1 - read protection: The reading of sensitive data is multi-party collaborative process in my approach. For a party to read out sensitive data without colluding with other parties is not possible in our architecture, because the only way sensitive data can leave the enclave, is in an encrypted state, encrypted by a threshold encryption schemes. It is provided, that transaction may only happen on top of committed ledger state, meaning a participant can not insert their own public key to later use this for the encryption, because this ledger state can not be committed (unless multiple organizations collide). We come to the conclusion, that only encrypted data can leave the enclave, so even if an attacker uses outdated access control scheme to read out a sensitive data, it can only view that data in an encrypted form. This satisfies the requirement, unless the participant breaks well established encryption schemes. (As mentioned earlier, it is advised to use large primes in the cryptosystems to ensure computational difficulty)

C2 – operation attempt based information leakage: My approach describes a way for implementing validation logic, that reveals minimal information amid attempting transaction invocations. If there are possible chains states (s), this chain state also includes the authoritative data access scheme (ac). If participants of the network agreed upon chain codes that use certain validations, we can easily see that if a participant speculatively attempts transaction on the current ledger state s, it might only extract information that is already available to it, by the "contract" of the chaincodes. This is ensured by the fact the approach prohibits the use of returning information about the source of an invalid request, as well as the fact that chaincode can only run on ledger states, that were committed. [Section 3.5.4]

C3 – observation-based information leakage: In our approach, since we are using a distributed ledger, every node of the network has a copy of the ledger, this requirement states that a participant should gain no sensitive information about the ledger state by using techniques to observe its local encrypted ledger state. Since the approach uses the enclaves state encryption key to store data on the ledger, nodes cannot understand what is being changed or modified on the ledger. However, the proposed FPC technology does not encrypt keys of the stored data (in the key value storage), so observers could gain information about system if keys contain meaningful data. The reason why this is not a problem at the moment, is the fact that the approach uses one key for its private storage, which means that apart from the fact that the state of the ledger "has changed", observers cannot infer any meaningful information about the ledger state.

Other requirements

M1 – minimally disruptive changes to control list: An important maintainability requirement of my system is to efficiently handle changes in the

access control scheme. Purely cryptographic solution often require the entire re-encryption of the private data to maintain confidentiality, but this is very inefficient. Since my approach uses my multiparty reading process, as soon as the new access scheme is committed to the nodes, there is nothing else left for the nodes to do. Changes to the access control scheme are just regular transactions, that change the behaviour of dependent chaincode functions.

AU1: For every read access to be auditable, similarly to M1, since a predefined threshold number of parties need to collaborate to read a data from the ledger, unless all of these parties are corrupt, we can assume that either one of the participants will a submit a transaction to the ledger with the partial decryption invocation (queries are not allowed by the chaincode design description), or keep another record of the request from the other party. If all parties collide, this requirement can be broken, even with by following my approach.

I1: To maintain the integrity of distributed ledgers, the system has to be resistant to Byzantine [58] fault/attack behaviour, since this is not entirely possible, my approach has to maximize the tolerance against attacks. To ensure this I advise the operators of the system to deploy as many nodes as they need. Proving that this is a mitigation to the problem is out of scope of this paper.

6 Conclusions

Emerging technologies like blockchains and trusted execution environment are opening the possibility of infinitely complex and useful possibilities. However as they are fairly new, their research is only in its early stages, in the grand scheme of things. To aid the exploration of the new possibilities, we need to first construct requirement models that could be useful in our world, then examine, analyse and test these technologies and concepts to gain insight about them. Finally, we can work with the now clear "puzzles pieces" that we have collected and combine them in new, exiting ways, and create protocols and restrictions to mitigate the effects of technological or conceptual limitations that we learned about.

In the paper, I attempted to do the previously described abstract process. Researched the emerging technologies deeply, constructed a requirement system for a useful unsolved use case, and then created a system/protocol to fulfil the requirements of the use case.

I conclude that trusted execution environments are promising way to handle data comprehensibility on blockchains, with minimal use of cryptography and following a protocol, we can construct a system that is sufficiently complex and secure according to my knowledge. Only large limitation is the trust that we offset to the developers of TEE technologies like Intel SGX. By breaking the integrity of the enclave, this solution can no longer promise confidentiality and privacy.

7 Bibliography

[1] S. Chakrabarti, T. Knauth, D. Kuvaiskii, M. Steiner, and M. Vij, 'Chapter 8 - Trusted execution environment with Intel SGX', in *Responsible Genomic Data Sharing*, X. Jiang and H. Tang, Eds., Academic Press, 2020, pp. 161–190. doi: 10.1016/B978-0-12-816197-5.00008-5.

[2] E. Androulaki *et al.*, 'Hyperledger fabric: a distributed operating system for permissioned blockchains', in *Proceedings of the Thirteenth EuroSys Conference*, in EuroSys '18. New York, NY, USA: Association for Computing Machinery, prilis 2018, pp. 1–15. doi: 10.1145/3190508.3190538.

[3] M. Brandenburger, C. Cachin, R. Kapitza, and A. Sorniotti, 'Blockchain and Trusted Computing: Problems, Pitfalls, and a Solution for Hyperledger Fabric', *arXiv*, May 2018, doi: 10.48550/arXiv.1805.08541.

[4] S. Nakamoto, 'Bitcoin: A Peer-to-Peer Electronic Cash System'.

[5] M. Chase, 'Multi-authority Attribute Based Encryption', in *Theory of Cryptography*, S. P. Vadhan, Ed., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 515–534. doi: 10.1007/978-3-540-70936-7_28.

[6] S. J. De and S. Ruj, 'Efficient Decentralized Attribute Based Access Control for Mobile Clouds', *IEEE Trans. Cloud Comput.*, vol. 8, no. 1, pp. 124–137, 2020, doi: 10.1109/TCC.2017.2754255.

[7] V. Goyal, O. Pandey, A. Sahai, and B. Waters, 'Attribute-based encryption for fine-grained access control of encrypted data', in *Proceedings of the 13th ACM conference on Computer and communications security*, in CCS '06. New York, NY, USA: Association for Computing Machinery, Október 2006, pp. 89–98. doi: 10.1145/1180405.1180418.

[8] H. Qian, J. Li, Y. Zhang, and J. Han, 'Privacy-preserving personal health record using multi-authority attribute-based encryption with revocation', *Int. J. Inf. Secur.*, vol. 14, no. 6, pp. 487–497, Nov. 2015, doi: 10.1007/s10207-014-0270-9.

[9] X. Liu, Y. Xia, S. Jiang, F. Xia, and Y. Wang, 'Hierarchical Attribute-Based Access Control with Authentication for Outsourced Data in Cloud Computing', in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, Jul. 2013, pp. 477–484. doi: 10.1109/TrustCom.2013.60.

[10] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou, 'Scalable and Secure Sharing of Personal Health Records in Cloud Computing Using Attribute-Based Encryption', *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 131–143, 2013, doi: 10.1109/TPDS.2012.97.

[11] A. Sahai and B. Waters, 'Fuzzy Identity Based Encryption'. 2004. Accessed: Oct. 29, 2023. [Online]. Available: https://eprint.iacr.org/2004/086

[12] A. Shamir, 'Identity-Based Cryptosystems and Signature Schemes', in *Advances in Cryptology*, G. R. Blakley and D. Chaum, Eds., in Lecture Notes

in Computer Science. Berlin, Heidelberg: Springer, 1985, pp. 47–53. doi: 10.1007/3-540-39568-7_5.

[13]U. Gupta *et al.*, 'Chasing Carbon: The Elusive Environmental Footprint of Computing', in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 854–867. doi: 10.1109/HPCA51647.2021.00076.

[14]S. Jahid, P. Mittal, and N. Borisov, 'EASiER: encryption-based access control in social networks with efficient revocation', in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, in ASIACCS '11. New York, NY, USA: Association for Computing Machinery, Március 2011, pp. 411–415. doi: 10.1145/1966913.1966970.

[15]R. Shraddha and T. Bharat, 'Enhancing Flexibility for ABE through the Use of Cipher Policy Scheme with Multiple Mediators', in *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014*, S. C. Satapathy, B. N. Biswal, S. K. Udgata, and J. K. Mandal, Eds., in Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2015, pp. 457–464. doi: 10.1007/978-3-319-11933-5_50.

[16]H. Yousuf, M. Lahzi, S. A. Salloum, and K. Shaalan, 'Systematic Review on Fully Homomorphic Encryption Scheme and Its Application', in *Recent Advances in Intelligent Systems and Smart Applications*, M. Al-Emran, K. Shaalan, and A. E. Hassanien, Eds., in Studies in Systems, Decision and Control. , Cham: Springer International Publishing, 2021, pp. 537–551. doi: 10.1007/978-3-030-47411-9_29.

[17]'Zero knowledge proofs of identity | Proceedings of the nineteenth annual ACM symposium on Theory of computing'. Accessed: Nov. 01, 2023. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/28395.28419

[18]W. Li, H. Guo, M. Nejad, and C.-C. Shen, 'Privacy-Preserving Traffic Management: A Blockchain and Zero-Knowledge Proof Inspired Approach', *IEEE Access*, vol. 8, pp. 181733–181743, 2020, doi: 10.1109/ACCESS.2020.3028189.

[19]A. E. B. Tomaz, J. C. D. Nascimento, A. S. Hafid, and J. N. De Souza, 'Preserving Privacy in Mobile Health Systems Using Non-Interactive Zero-Knowledge Proof and Blockchain', *IEEE Access*, vol. 8, pp. 204441–204458, 2020, doi: 10.1109/ACCESS.2020.3036811.

[20]J. Boyar and R. Peralta, 'On the concrete complexity of zero-knowledge proofs', in *Advances in Cryptology — CRYPTO' 89 Proceedings*, G. Brassard, Ed., in Lecture Notes in Computer Science. New York, NY: Springer, 1990, pp. 507–525. doi: 10.1007/0-387-34805-0_45.

[21]V. Costan and S. Devadas, 'Intel SGX Explained'. 2016. Accessed: Oct. 28, 2023. [Online]. Available: https://eprint.iacr.org/2016/086

[22]J. M. Rushby, 'Design and verification of secure systems', *ACM SIGOPS Oper. Syst. Rev.*, vol. 15, no. 5, pp. 12–21, Dec. 1981, doi: 10.1145/1067627.806586.

[23]A. Ltd, 'TrustZone for Cortex-M – Arm®', Arm | The Architecture for the Digital World. Accessed: Nov. 01, 2023. [Online]. Available: https://www.arm.com/technologies/trustzone-for-cortex-m
[24]'AMD Secure Encrypted Virtualization (SEV)', AMD. Accessed: Nov. 01, 2023. [Online]. Available: https://www.amd.com/en/developer/sev.html
[25]'PowerDVD - Award-Winning Blu ray & 8K Media Player for Windows'. Accessed: Nov. 01, 2023. [Online]. Available: https://www.cyberlink.com/products/powerdvd-ultra/spec_en_US.html
[26]'SAP : Security vulnerabilities, CVEs'. Accessed: Nov. 01, 2023. [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-797/SAP.html?page=1&order=1&trc=1395&sha=e62d8a2c3b36760650ea919de531215046da3593
[27]A. S. Rajasekaran, M. Azees, and F. Al-Turjman, 'A comprehensive survey on blockchain technology', *Sustain. Energy Technol. Assess.*, vol. 52, p. 102039, Aug. 2022, doi: 10.1016/j.seta.2022.102039.
[28]'Fabric Private Chaincode | IBM Research'. Accessed: Sep. 21, 2023. [Online]. Available: https://research.ibm.com/projects/fabric-private-chaincode
[29]'Complete Your Web3 Journey With Decentralized Storage'. Accessed: Nov. 01, 2023. [Online]. Available: https://crust.network/?ref=cms.polkadot.network
[30]'Zondax: Secure Blockchain R&D Solutions'. Accessed: Nov. 01, 2023. [Online]. Available: https://zondax.ch
[31]'Automata - Modular attestation layer for Web3'. Accessed: Nov. 01, 2023. [Online]. Available: https://www.ata.network/
[32]A. Brenzikofer, 'Have a TEE with Polkadot', Polkadot Network. Accessed: Sep. 21, 2023. [Online]. Available: https://medium.com/polkadot-network/have-a-tee-with-polkadot-7ea052e4d69a
[33]'integritee-network/substraTEE'. Integritee Network, Feb. 04, 2023. Accessed: Nov. 01, 2023. [Online]. Available: https://github.com/integritee-network/substraTEE
[34]D. G. Wood, 'POLKADOT: VISION FOR A HETEROGENEOUS MULTI-CHAIN FRAMEWORK'.
[35]S. van Schaik *et al.*, 'SoK: SGX.Fail: How Stuff Gets eXposed'.
[36]S. Network, 'Secret Network'. Accessed: Nov. 01, 2023. [Online]. Available: https://scrt.network
[37]'Stale Data Read from xAPIC / CVE-2022-21233 / INTEL-SA-00657'. Accessed: Nov. 01, 2023. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/stale-data-read-from-xapic.html
[38]'INTEL-SA-00615', Intel. Accessed: Nov. 01, 2023. [Online]. Available: https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00615.html
[39]A. Klenik and I. Kocsis, 'Porting a benchmark with a classic workload to blockchain: TPC-C on hyperledger fabric', in *Proceedings of the 37th*

ACM/SIGAPP Symposium on Applied Computing, in SAC '22. New York, NY, USA: Association for Computing Machinery, Május 2022, pp. 290–298. doi: 10.1145/3477314.3507006.

[40]'Introduction to Intel® SGX Sealing', Intel. Accessed: Oct. 28, 2023. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-intel-sgx-sealing.html

[41]'Code Sample: Intel® Software Guard Extensions Remote Attestation...', Intel. Accessed: Oct. 28, 2023. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/code-sample/software-guard-extensions-remote-attestation-end-to-end-example.html

[42]'Deprecated Technologies - 001 - ID:655258 | 12th Generation Intel® Core™ Processors Datasheet, Volume 1 of 2'. Accessed: Oct. 28, 2023. [Online]. Available: https://edc.intel.com/content/www/us/en/design/ipla/software-development-platforms/client/platforms/alder-lake-desktop/12th-generation-intel-core-processors-datasheet-volume-1-of-2/001/deprecated-technologies/

[43]M. Brandenburger, C. Cachin, R. Kapitza, and A. Sorniotti, 'Trusted Computing Meets Blockchain: Rollback Attacks and a Solution for Hyperledger Fabric', in *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, 2019, pp. 324–32409. doi: 10.1109/SRDS47363.2019.00045.

[44]'fabric-rfcs/images/fpc/full-detail/fpc-key-dist.png at main · hyperledger/fabric-rfcs', GitHub. Accessed: Oct. 31, 2023. [Online]. Available: https://github.com/hyperledger/fabric-rfcs/blob/main/images/fpc/full-detail/fpc-key-dist.png

[45]'Rollback protection Extension (aka Trusted Ledger) · Issue #484 · hyperledger/fabric-private-chaincode', GitHub. Accessed: Oct. 31, 2023. [Online]. Available: https://github.com/hyperledger/fabric-private-chaincode/issues/484

[46]'fabric-private-chaincode/ecc_enclave/enclave/shim.h at main · hyperledger/fabric-private-chaincode', GitHub. Accessed: Oct. 31, 2023. [Online]. Available: https://github.com/hyperledger/fabric-private-chaincode/blob/main/ecc_enclave/enclave/shim.h

[47]D. P. Bovet and P. Crescenzi, *Introduction to the theory of complexity*. GBR: Prentice Hall International (UK) Ltd., 1994.

[48]T. Elgamal, 'A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms', *IEEE Trans. Inf. THEORY*, no. 4, 1985.

[49]V. Shoup and R. Gennaro, 'Securing threshold cryptosystems against chosen ciphertext attack', in *Advances in Cryptology — EUROCRYPT'98*, K. Nyberg, Ed., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1998, pp. 1–16. doi: 10.1007/BFb0054113.

[50]C. Cachin, 'Distributed Cryptography'. 2012.

[51]A. Shamir, 'How to share a secret', *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979, doi: 10.1145/359168.359176.

[52]R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, 'Secure Distributed Key Generation for Discrete-Log Based Cryptosystems', *J. Cryptol.*, vol. 20, no. 1, pp. 51–83, Jan. 2007, doi: 10.1007/s00145-006-0347-3.

[53]M. Naor and M. Yung, 'Public-key cryptosystems provably secure against chosen ciphertext attacks', in *Proceedings of the twenty-second annual ACM symposium on Theory of Computing*, in STOC '90. New York, NY, USA: Association for Computing Machinery, prilis 1990, pp. 427–437. doi: 10.1145/100216.100273.

[54]N. Schmid, 'noahschmid/threshold_elgamal_python'. Oct. 26, 2023. Accessed: Nov. 01, 2023. [Online]. Available: https://github.com/noahschmid/threshold_elgamal_python

[55]'squishytiramisu/DCBTEE: Fulfilling Data Comprehensibility on Blockchain with Trusted Execution Environments prototype implementation', GitHub. Accessed: Nov. 01, 2023. [Online]. Available: https://github.com/squishytiramisu/DCBTEE

[56]'hyperledger-labs/fabric-smart-client: The Fabric Smart Client is a new Fabric Client that lets you focus on the business processes and simplifies the development of Fabric-based distributed application.' Accessed: Nov. 01, 2023. [Online]. Available: https://github.com/hyperledger-labs/fabric-smart-client

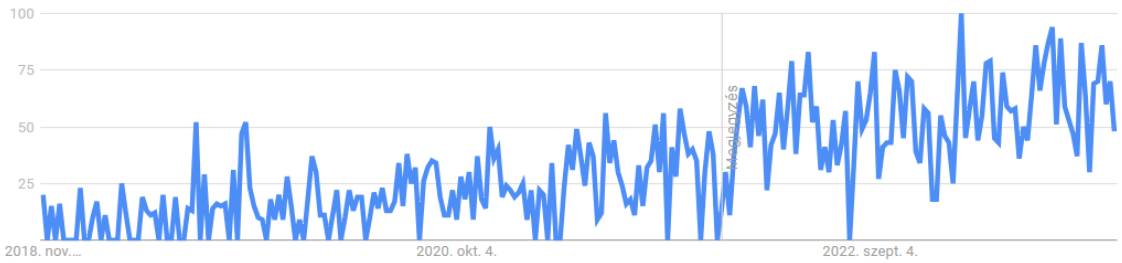
[57]'Use Case Tracker – Hyperledger Foundation'. Accessed: Nov. 02, 2023. [Online]. Available: https://www.hyperledger.org/learn/use-case-tracker

[58]K. Driscoll, B. Hall, H. Sivencrona, and P. Zumsteg, 'Byzantine Fault Tolerance, from Theory to Reality', in *Computer Safety, Reliability, and Security*, S. Anderson, M. Felici, and B. Littlewood, Eds., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 235–248. doi: 10.1007/978-3-540-39878-3_19.

Appendix

Benefits of using blob storage under one key: As mentioned earlier, there are not just drawbacks to using one key pair to store the entire private state of our application. One might argue that in a strict confidentiality application, the key management of storage objects could also create spots where information could unwantedly leak. Since keys are unencrypted even in the privately stored objects, a participant could just look at the key-value storage, and notice that another object has appeared with a certain key. If we are not careful, we might leak sensitive information. For instance, let's imagine that an organization stores a person's organization related data with the key to the persons ID, since we have access to the chaincodes, we obtain this information, and a certain transaction happened that resulted in the person's data changed. This solution completely hides any relationships between keys and data from the view of the key-value storage.

Handling ledger state change visibility: In a model where I only store one key-value pair on the encrypted key-value store, I could simply use a nonce, given as an encrypted argument, or a hash generated from the transaction arguments to modify this unreadable nonce value in the data structure. If we require every transaction to modify this value, a change in the network would not be visible.



30. Figure Google searches for the term confidential computing

Intel SGX Vulnerability mitigation strategies: I recommend [35] Section 3