VRIJE UNIVERSITEIT BRUSSEL

Bruface
Brussels Faculty of Engineering

OBJECT CAPABILITIES FOR PERMISSIONED BLOCKCHAINS

JANNES DE METS

2021-2022

Promotor: Jens Nicolay

Wetenschappen en Bio-ingenieurswetenschappen

Table of Contents

Abstract.

Businesses are using **permissioned blockchains** for secure and distributed storage and computing. These networks require an **access control model** to enforce security policies. Traditionally, a model called **access control list** is used in this domain, which has several limitations and problems. Another model called **object capability** has promising characteristics that provides solutions to the problems of access control lists. An in-depth comparison between these two models concludes in six properties which are desired for our proposed solution. Related work in the form of former research and state-of-the-art implementations highlight the common use of access control lists in solutions aimed towards enterprises. In addition, existing capability-based solutions lack the desired properties of the 'true' object capability model. To overcome these shortcomings, we propose an object capability model that manages access control in business-to-business permissioned blockchains networks. This model uses a 'capability-aware' gateway for the distribution of capability objects which are the 'access control 'bookkeepers' of the system. Our research is supported by a proof-of-concept called Jannes' Object Capability-Based Access Control [1] (JOCBAC), the prototype of the proposed model implemented in Hyperledger Fabric. This model demonstrates that object capabilities for business-to-business permissioned blockchain networks provide a more user- and administrator-friendly model as opposed to ACLs. In addition, the JOCBAC model has a fine granularity of authorisation options and respects the principle of least authority.

1 Introduction

1.1 Motivation

The need for transparency and security in a digital world are some of the main reasons why **blockchain** is becoming a popular technology to use, both in public and private sectors. This technology presents transparency and security, it solves problems with sharing data in environments where trust is not guaranteed and can provide distributed computation services. Therefore, companies are using permissioned blockchains to share data with partners and executing business logic in a safe manner. Custom blockchain networks are build using frameworks such as Ethereum [2], Hyperledger Fabric [3] and many more. The attribute 'permissioned' implies that these blockchains require some form of access control to manage access to the network.

Access Control (AC) has many forms and shapes. In general, it enforces security policies to ensure that only authenticated users can perform the correct set of operations. This paper will introduce traditional models along with a more detailed description on two groups of Access Control Models (ACMs), called Access Control Lists (ACLs), and capabilities. **Capabilities (caps)** are an approach for managing access control and provide more dynamic and fine-grained models as opposed to ACLs. Because of their promising features, this paper provides an evolution on capability-based models and explores differences between the traditional ACL and the proposed object capability model.

1.2 Problem

Many strategies have been proposed or developed to manage AC, each with their advantages and disadvantages in different use-cases. Most of the existing capability models are targeted towards an Internet-of-Things (IoT) -environment, meaning scalability and limited hardware are key factors. This results in abandoning certain features leading to models that do not conform to the object capability principles desired for this thesis. Instead, this thesis is aimed towards the business-to-business domain which allows more freedom when designing an ACM, as opposed to the IoT domain. These businesses can include many partners resulting in complex hierarchical structures but is still not comparable in size to models meant for IoT or public networks. In our scenario, ACL is most commonly used but this model has issues and limitations for which the object capability model provides solutions. The manner in which these problems are addressed is described in the next section.

1.3 Approach

Enterprises have a need for a flexible and dynamic access control system. To solve the problems previously summarised, an object capability ACM for permissioned blockchains is proposed. In this model, capabilities are the access control 'accountants' themselves, which means that they handle access control logic and provide direct access to the blockchain network. This solution consists of a 'capability-aware' gateway that connects subjects to the network and manages the distribution of capabilities. The underlying blockchain platform is used for storage and basic governance of capabilities. In this way, applications can use the capabilities they receive as the API of the system, as all the operations they perform on the network are initiated through these objects.

1.4 Overview

This thesis is structured as followed: chapter 2 explains fundamental concepts such as blockchain, access control in general and access control models such as the capability model required to understand the thesis problem. It also includes an in-depth comparison between ACL and caps and concludes with presenting related work. Chapter 3 formulates the thesis scope and problems, followed by the conceptual approach in chapter 4. The implementation called JOCBAC [1] is introduced in chapter 5, including information about frameworks used to build permissioned blockchains and an overview on the framework called Hyperledger Fabric. Chapter 6 validates the implementation and more information regarding related work is presented in chapter 7. Chapter 8 contains an overview of the findings presented in this paper along with future work and to conclude this paper, the used references are listed.

2 Background

This chapter introduces background information required to formulate the thesis problem and approach. It starts with the basics of a blockchain (2.1) which covers topics such as distribution and smart contracts (2.1.1), and ends with a few practical examples (2.1.2). Proceeding is a definition and a brief history of access control (2.2). This section also includes a description of the access control list model (2.2.1) and the evolution of the capability model (2.2.2). These two models are further compared to underline the advantages of capabilities (2.2.3) and the segment regarding access control is concluded with a summary on the discussed models and properties in Table 3.

2.1 Blockchain

Organisations are using blockchain platforms for storing data and executing business logic. A blockchain is a distributed ledger and is realised by cryptographically linking blocks together. In the basic data structure, each block contains data, a hash, and the hash of the previous block. Storing the previous hash is what creates the chain and the only block without a previous hash is the first block, which is called the genesis block. The hash is a unique fingerprint defined by the content of a block. If the data in a block would change then the hash will change as well, resulting in an incorrect link with the next block, thus invalidating all upcoming blocks. This is great for detecting changes and as a first layer of security. But with today's computation power in mind, it would still be possible for e.g., a potential attacker, to insert a tampered block and recalculate all upcoming blocks. That is why a consensus mechanism exists in blockchain, meaning that an agreement in between participants must be reached before a block becomes valid, i.e., reaching consensus. Considering the example of a democratic vote as a consensus mechanism: an attacker now needs to recalculate the blockchain as well as take over 50% of the network to declare his tampered blocks as valid for everyone. This allows for a secure way to store and share data in an untrusty environment. Essentially, a blockchain stores the transaction history that leads to its present world state, which holds all current values of all objects in the system.

2.1.1 Blockchain Properties

A blockchain is a **distributed** data structure, this means that data is decentralised and divided in a peer-to-peer network. A copy of the ledger is stored on each peer resulting in a fault-tolerant system in case of outage and allowing participants to verify the integrity of the ledger for themselves. In addition, a blockchain is transparent and secure because consensus must be reached in between participants when adding or modifying data and the complete transaction history is always available. In contrast, a non-decentralised structure depends on a central authority to manage data and is vulnerable to outages and security risks (more on this in section 2.2.3.2: Distribution).

Smart contracts are immutable, self-executing programs that are stored on a blockchain and trigger when certain conditions are met. In most permissioned blockchain platforms they are developed using Turing-complete programming languages, meaning they theoretically can express any computation. The main purpose of smart contracts is to automate business logic, e.g., arrangements between parties at an increased speed, accuracy, and lower cost. This is even more evident when compared to manual labour. In addition,

smart contracts are encrypted on the blockchain-level to keep documents safe, thus increasing the transparency and security of the network. To conclude, one could say that smart contracts extend the blockchain technology from a distributed database to hybrid distributed storage and computing.

2.1.2 Types of Blockchain and Examples

In general, there are three types of blockchain: public, private, and **permissioned**. A public chain allows anyone to join the network and perform transactions, whereas the private network only allows certain nodes to participate in the network and it usually depends on some form of central authority to manage and/or define these. In between is the permissioned blockchain, which can be compared to a public blockchain but with an access control layer on top of it. Sometimes public blockchains are also referred to as permissionless, for they do not demand any permissions to interact with the network. Hybrid blockchain is another term that is occasionally used for describing a permissioned blockchain that is not private. Finally, Consortium blockchain stands for a private blockchain that is controlled by a group as opposed to one authority.

Blockchains are commonly used in sectors that deal with sensitive information including, but not limited to, finance, business, and hospitals. An example of a public blockchain is the famous cryptocurrency Bitcoin [4], created by an anonymous person or group and used as a virtual currency. Ethereum [2] is also a public network, but it has the option to create permissioned networks as well, for example the Quorum Blockchain Service [5] is an Ethereum fork that allows for creating business networks.

Wrapping up, it is difficult to give a singular definition of what a blockchain is because every implementation defines and translates the basic concepts in its own ways. In general, blockchains are used to store and verify data transparently and securely, and to keep an immutable record of transactions. It solves problems for sharing data in an untrusty environment and provides distributed computing solutions.

2.2 Access Control

Access Control (AC) manages authentication and authorisation, respectively meaning it oversees *who* has access to *what* resources. When a permissioned blockchain is used by organisation(s), not all participating subjects should have access to all resources in the system, this is where access control comes in. Take for instance an object that certain subjects want to obtain, authentication stands for confirming the identity of the subject and verifying if it is allowed access to the object or not, whereas **authorisation** defines the

granularity of actions the subject can perform on that object. Aside from these two, confidentiality and accountability are also crucial factors, which respectively mean keeping sensitive information private and being able to trace activities to a specific subject. In this way, there is no discussion possible about whether some transaction took place or not without exposing sensitive information. Authorisation is the most principal factor for this study, i.e., achieving a high granularity in options without diminishing on security.

Access Control Models (ACMs) are structures that define how exactly the access control is enforced. This is usually done in an access control matrix containing all objects in the system related to subjects with their respective rights. It is rarely stored directly in this form because that would consume too much memory, instead its sparse and repetitive properties are exploited to shard the matrix using different techniques. These methods are generally dividable in two groups: access control lists and capabilities. Each stores access control policies in a separate way resulting in different models to manage access control.

2.2.1 Access Control Lists

The Access Control List, (ACL), stores information regarding access by row such that objects are related to subject's rights. Hence, for each object there exists a list containing which subject has what type of access to this object. An example shown in Table 1 is the entry Object1 that contains Subject1 and Subject3, both having read and execute permissions (r-x) but only Subject1 also has write permissions (-w-). As an example, ACL is commonly used in routers and switches as a light-weight version of a firewall.

ACL	
Object1	[(Subject1 ; rwx) , (Subject3 ; r-x) , …]
…	…

Table 1: Illustrative example of an Access Control List

From ACLs grew the ideas of grouping subjects by roles or adding attributes to them for more control and less memory use. This results in two models that are commonly used as general-purpose ACMs, RBAC and ABAC. The **Role Based Access Control** (RBAC) model uses roles to simplify its AC-matrix and is based on the traditional ACL model. Instead of storing all subjects as separate entities in an ACL, they are grouped together in roles which define the authority relating to objects. Instead of relying on identity grouping with roles, another model called **Attribute Based Access Control** (ABAC) uses policies containing objects and context (e.g., location, time...) for a more fine-grained control. This is usually

implemented using the ACL method, as resources are still related to subjects but with extra context. Custom attributes provide this extra context and are attached to subjects. These features result in a model with more possibilities for designing an AC-system. However, both models share problems inherent to ACLs. Next section discusses the model in which we are most interested in.

2.2.2 The evolution of Capabilities

The object capability (ocap) Access Control Model (ACM) distributes objects called capabilities (caps) over subjects. These capabilities manage access control decisions and provide direct access to the underlying system. Through the years of research, many capability-based ACMs have been designed. However, most of them misinterpreted or wrongly implemented critical concepts resulting in discussions around the effectiveness of using caps as opposed to ACL. This section portrays a brief history on the cap model which ends in the definition of true object capabilities.

2.2.2.1 Capabilities as Lists

In a capability list (cap-list), each subject is associated with a set of rights to objects, swapping the rows and columns of an ACL's matrix. These models are remarkably similar on data level, but in software engineering it is a significant difference. Referring to Table 2, entry Subject1 is now related to a set of objects in combination with the permissions to them.

Capability List	
Subject1	[(Object1 ; r - x) , (Object2 ; r – x) , …]
…	…

Table 2: Illustrative example of a Capability List

2.2.2.2 Capabilities as Keys

Using caps in a list manner does not unlock their full potential. A different manner of modelling caps is representing them as capability-keys (cap-key), or non-forgeable tokens in which access rights are stored. The ownership of a key grants access rights to the holder and it is possible for subjects to confine, delegate, and revoke keys to other subjects. The main difference between the list and key form is the subject's selection of which key to use when performing access, which is not required in a cap-list. The capability-as-keys model can be compared to the 'real world' representation of owning a key for a certain car. Ownership of the key grants access to the car and the key can be passed around or even be

copied, but not reforged. The owner of the key is responsible for the delegation and should trust the subject to whom it shares its key with. Even though this is an appealing analogy, the key model is not an accurate representation of "true capabilities", or object capabilities.

2.2.2.3 Capabilities as Objects

The object capability model (ocap) uses object references instead of tokens, allowing subjects to send a message directly to the object by invoking the reference. In the ocap model, if Alice wants access to a certain object, she is required to own a capability that locates that object. Contrasting with the previously mentioned car key analogy in which the key does not locate the car, it only provides access to it. In addition, the cap-key model has distinct operations for access and authorisation; access defines an interaction from subject to resource, whereas authorisation is an action from subject to subject. In the ocap model there is no distinction between subjects and objects, resulting in uniform access and authorisation methods.

The evolution on different capability-based models and their shortcomings are further described in "Capability Myths Demolished" [6] and in addition, as the title insinuates, it debunks myths that raised around the use of (object) capabilities. While the paper received sceptical reactions [7] *e.g., nit-picking on minor details*, it is still an interesting addition in the evolution of capability-based systems.

Delegation and Revocation

Advancing with object capabilities, delegation, revocation, and confinement are all possible in this model. Where delegation means sharing of capabilities with other subjects, confinement is the conversion of a capability into a less-privileged version, and revocation means renouncing already delegated access rights. These two concepts are clarified with the following example.

One manner for realising **delegation** properties is illustrated in Figure 1. Alice is a subject that owns a capability to Carol, Alice wants to delegate her capability to another subject Bob, allowing Bob to also access Carol. To successfully share authorisations with Bob, Alice must already have access to both Bob and Carol since a delegation cannot introduce a new link in between subjects. If Bob was not to be trusted, access to him should be denied such that delegation of authority is not possible. Otherwise, Alice can share her capability allowing Bob to use the reference to Carol for access.

Revocation is also possible in different manners; the basic idea is explained in a similar example as the previous one and shown in Figure 2. If again Alice wants to grant Bob access to Carol but also wants to revoke this access when required, then two facets F, for forward, and R, for revocation, are added. Alice grants Bob access to the first facet F, which forwards access to the second facet R which she keeps for herself. This facet then forwards the reference to Carol, if Alice wants to revoke access she drops facet R, rendering F useless since it now points to nothing. In this way, the capability itself might not be revoked but access to Carol is, resulting in a mechanism which also works for delegated authorities, i.e., if Bob passed the reference to F further down. This mechanism of revocation is invented by D. D. Redell (1974) [8], and first used in the CAP-III system [9].

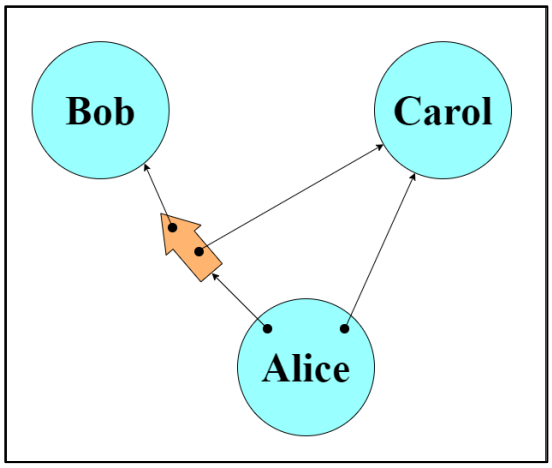

Figure 1: Delegation of a capability

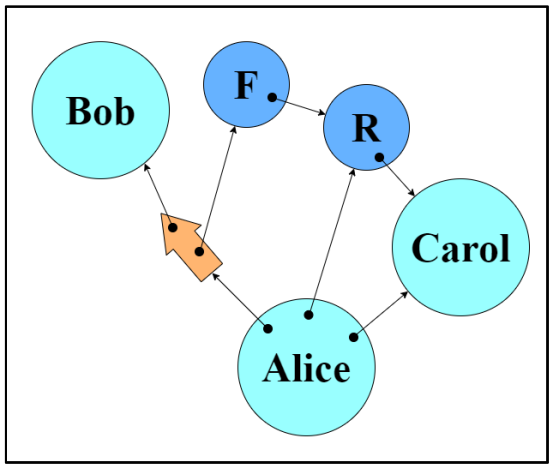

Figure 2: Revocation of a capability

To conclude, object capabilities provide promising characteristics for developing access control models and unless stated different, further use of the term 'capability' will refer to an object capability. An overview of the previously discussed models along with their distinctive properties is depicted in Table 3. An example of ocaps in use is the E object-capability programming language developed by Mark Miller, which is a framework for writing distributed capability-based smart contracts [10, 11]. There are multiple programming languages currently under support that are based on or inspired by the E programming language, thus supporting object capabilities. One example is the Hack programming language [12], developed by Meta as an alternative to PHP.

2.2.3 Capabilities vs Access Control Lists

Access Control Lists (ACLs) and object capabilities (ocaps) are compared to showcase the underlying problems that arise when using ACLs and is summarised in the next table. The columns are an outline of the abstract ACMs that are discussed in the previous chapter, namely ACL (2.2.1) and the evolution of capability models (2.2.2). The rows are the properties that are introduced throughout this section.

	ACL	**Cap-List**	**Cap-Key**	**OCap**
(2.2.3.2) Distribution	Impractical	Possible	Possible	Possible
(2.2.3.4) Least Privilege	Impractical	Possible	Possible	Possible
(2.2.3.3) Dynamic Subjects	No	Yes	Yes	Yes
(2.2.3.5) No Ambient Authority	No	No	Yes	Yes
(2.2.3.1) Unified Designation and Authority	No	Possible	No	Yes
(2.2.3.6) Controlled Delegation	No	No	No	Yes

Table 3: Difference between the Access Control List and Capability models in relation to six key properties.

The following six segments each start with introducing a certain problem regarding ACL and presenting how caps propose a solution *in italic*, followed by a more detailed explanation on the matter.

2.2.3.1 Unified Designation and Authority

An ACL demands designating the resource and deducing what kind of privileges the subject has before access control decisions can take place. Whereas a capability both serves as a designator of what object to access and provides the authorities to do so.

This property is evident in the perspective of the subject and is also the difference between the cap-key and ocap models. Ocaps contain a link to the resource and the related privileges at once, while a key in the cap-key model does not locate the resource, it only provides access to it. In an ACL, the object must first be located before the privileges are accessible thus this property is not satisfied. Having a unified location pointer and set of

rights has an impact on security and how this is managed and becomes more evident when combined with other properties discussed next.

2.2.3.2 Distribution

The centralized nature of ACLs makes them vulnerable to a single point of failure (SPoF), which is defined as a weak point that potentially could bring down an entire system. Whereas object capabilities have a decentralised nature mainly because they are distributed over subjects.

SPoF is a general problem with centralized structures and can already be partially decentralized when using a blockchain and smart contracts to store and manage capabilities or other AC-related data. In contrast to the centralised SPoF problem, when nodes in the blockchain network are suddenly down, the network should still be active as explained in section Blockchain (2.1). ACL models implemented with smart contracts inherit the same problems that come with the traditional ACLs resulting in a less dynamic system or in an 'administrative nightmare' to maintain. Capabilities on the other hand are easier to distribute over users as one capability object serves one subject. Research concerning the complete distribution of capability-based AC has been conducted, for instance, J.L. Hernández-Ramos et al. (2016) [13] designed a decentralized capability-based access control model called DCapBAC (7.2.1) to demonstrate that it is possible to completely decentralise a cap-based ACM without using a blockchain. The papers mentioned in CapBAC (7.2.2) also conclude that the simplicity and fine granularity of cap-based systems smoothly integrate into a blockchain.

2.2.3.3 Dynamic Subjects

ACL is rigid and requires a lot of overhead when dynamicity of subjects is important, thus reducing scalability and resulting in a less dynamic model. In contrast, object capabilities define and distinguish subjects at the instance level with types such as objects, classes, processes, or programs, allowing for increased granularity in the definition of subjects.

An increase in granularity implies more choices when creating AC rules. For instance, an Access Control Model (ACM) that can use context such as a subject's time of the day to make decision has more granularity than a system that does not consider context. Thus, a higher granularity comes with more expressiveness when developing ACMs. The flexibility of defining subjects is also present in the cap-list and cap-key versions. It is true that Ocaps also suffers from scalability issues in very large-scale use-cases (e.g., for the IoT domain), but that is less important for this thesis which is mainly interested in exploiting the

fine-grained authorisation that comes with the use of capabilities in a business-to-business network.

2.2.3.4 Principle of Least Privilege

ACL models are challenged when administering the principle of least privilege, which states that only the minimum required rights for a subject to perform its actions should be granted and nothing more. Object capabilities do follow this concept as they have confinement and delegation techniques, and the dynamic properties required to adhere to it.

This principle is one of the five basic design principles for information protection as stated by J. H. Saltzer et al. (1975) [14] and helps with minimising the possible damage that can be dealt to a system in combination with preventing incorrect usage of privileges. ACL systems lack the granularity to always conform to this principle without creating an administrative nightmare. This is partly since the power to edit a resource's rights usually comes with the rights of changing multiple parts of the ACL, resulting in too many privileges being granted. Ocaps have the Dynamic Subjects property implying the ease of defining a variety of subjects without resulting in an overcomplex system, even when subjects are extensively creating new subjects as well. In addition, it is easier to restrict a subject's editing of authorities to the bare minimum requirement, resulting in a system that can preserve the principle of least privilege.

2.2.3.5 Ambient Authority

ACL systems have ambient authority due to the location designator and the subjects' rights being separated, whereas object capabilities do not.

A subject accessing a certain object uses ambient authority when it is only required to give the name of the object along with the operation it wants to execute for rights to be granted. This can lead to the abuse of a more privileged subject in the system, i.e., the confused deputy problem discovered by N. Hardy (1988) [15]. This problem occurs when a third party succeeds in fooling a more privileged component, the deputy, into misusing its authority. This enables the third party to invoke actions it is normally not allowed to do, resulting in an indirect violation to access rights. An example to illustrate the confused deputy problem is the web-browser: it has the authority to oversee private session information, but it can also run malicious Java-scripts that tamper with this data without the intent to harm the user. In this scenario, the browser is the confused deputy fooled by the attacker's script. This problem can be solved by having no ambient authority in the system. In that case, the browser cannot use the authority meant for the cookies in relation with

the script which originates from a different subject. The cap-list model does not require users to give extra information whereas cap-key requires to pass on a valid key as well, thus removing ambient authority.

2.2.3.6 Controlled Delegation

*ACL models lack flexible **delegation**, **revocation**, and **confinement** techniques which object capabilities do offer.*

The ocap-structure paves the way for delegation to be possible. These techniques add to the overall granularity of cap-based systems and in combination with e.g., providing context, results in a lot of freedom to define precise restrictions for subjects. To give an example of a system that possess these properties, S. Bistarelli et al. (2019) [16] have implemented a proof-of-concept ACM based on the CapBAC system by S. Gusmeroli et al. (2013) [17] on top of a blockchain where delegation and revocation of rights are possible (7.2.2). ACL oriented systems that have delegation possibilities do exist, usually in the form of extracting and passing on parts of the ACL. However, the storing and transferring of rights is done in such diverse ways which usually result in a lack of granularity and a scarcity of confinement and revocation possibilities. The ocap model's unification of subjects and objects in combination with the required capability for designating resources results in the delegation of capabilities being regulated. I.e., there is no way of delegating a certain capability without having the required permissions to do so.

2.2.3.7 Conclusion: Object Capabilities vs ACLs

The previous chapter discussed the topics permissioned blockchain and access control focussing on the Access Control List (ACL) and capability models. These models were further compared to discover underlying problems with ACL and find out how capabilities offer a solution. The previous section signifies that the object capability model is easier to distribute over subjects, has more granularity due to context and dynamic subjects, has controlled delegation, revocation, and confinement techniques, and that ocap preserves the principle of least authority without creating ambient authority. Now that the background information is established, the problems and approach of this thesis are formulated in greater detail.

3 Problem Definition

Most Access Control Models (ACMs) used for business-to-business permissioned blockchain networks utilize some form of Access Control List (ACL). The ACL model has issues and limitations for which the capability model provides solutions (2.2.3). After surveying existing solutions and comparing them with the properties of the desired object capability model, the following two problems are identified and are further underlined with concrete examples of developed capability ACMs (3.1).

(1) The applications or domains of state-of-the-art capability-based solutions are not aimed towards enterprises, resulting in abandoning desired features.

A substantial number of research is focused on distribution for large scale IoT scenarios, where there is a demand for heterogeneity, to connect a large number of devices and for the use of lightweight cryptography schemes such that hardware-constrained devices can also participate. However, this thesis domain is the private use of enterprises. These enterprises may include many partners which can result in complex hierarchical structures but is still not comparable in size to the models meant for IoT or public networks. To further illustrate the shortage in solutions aimed towards enterprises, the literate review by Bashir Dar A. et al. (2020) [18] (7.1) only has 2 of their total 76 papers (2.6%) targeted towards enterprises. This difference in target domain results in state-of-the-art abandoning certain desired features, further described in the next problem.

(2) The state-of-the-art capability-based solutions do not conform to the desired object capability principles.

A capability is an object that should administer both the authorisation for its owner and provide direct access to the network when it has permission to do so. Many of the existing capability-based ACMs for blockchains use tokens or keys instead of objects. The differences between these, presented in Table 3, are the 'Unified Designation and Authority' and 'Controlled Delegation' properties. In addition, the state-of-the-art models also rely on a central manager for verifying those keys, invoking smart contracts, or designating resources, which also contradicts the 'Unified Designation and Authority' property.

3.1 State-of-the-art examples

Four state-of-the-art capability-based ACMs are further analysed to underline the absence of object capability-based Access Control Models (ACMs) and is summarised in Table 4. This table relates the models to five characteristics that are most relevant for this thesis. The last column contains JOCBAC, our prototype of the desired object capability ACM and

is further touched upon starting at chapter 5: Implementation. The research mentioned in the columns each also contain an implementation or simulation as a proof-of-concept and are described in chapter 7: Related Work. The values such as high and medium are relative to the other models.

	DCapBAC [13]	**CapBAC** [16, 17, 19]	**BlendCAC** [20]	**IoT-CCAC** [21]	**JOCBAC** (5.2)
Blockchain	-	HL Sawtooth	Ethereum	BigchainDB	Hyperledger Fabric
Application (domain)	IoT	IoT	IoT	IoT & Enter-prise	Enterprise
Granularity	Medium	Medium	Medium	High	High
Delegation & Revocation	No	Yes	Yes	Yes	Yes
Object Capa-bilities	No	No	No	No	Yes

Table 4: Description of mentioned implemented systems related to five properties.

The DCapBAC (7.2.1) implementation (mentioned in 2.2.3.2) comes at the cost of abandoning delegation mechanics followed by a loss in granularity. This model is the only one listed in Table 4 that does not use a blockchain platform but even so, it demonstrates the potential of the capability model for distributed solutions. However, most implemented capability models are not truly distributed as there remains some form of central entity that manages a certain part of the system. To continue, the CapBAC (7.2.2) models do not consider enough context for AC decision making resulting in a lower amount of granularity than desired for this thesis. Similar to BlendCAC (7.2.3), these system also do not use 'real' object capabilities but instead they use tokens or keys. Continuing with the BlendCAC model, only administrators can perform revocation on a capability and the implementation suffers from latency of the underlying Ethereum network. IoT-CCAC (7.2.4) also generates capability tokens instead of actual capability objects and the token itself does not directly provide access to a resource. This is because the validity of the token and fulfilment of access control conditions are managed externally by an Identity Management process, not through the cap itself. Next chapter explains how a true ocap model design is approached for permissioned blockchains using gateways and smart contracts.

4 Approach

This thesis proposes a true **object capability**-based **access control model** for a permissioned blockchain to overcome the limitations that come with the traditional ACL models and thus filling the gap of ocap models in the business-to-business domain. What makes this approach unique is the use of distributed object capabilities that perform the access control decision themselves and can provide direct access to the network. The proposed solution consists of a gateway that distributes capabilities over subjects and uses the underlying blockchain platform for distributed storage and management of the capabilities. The next paragraphs describe an overview on the mentioned conceptual building blocks of the proposed model. This is done by illustrating the basic structure of the approach in Figure 3, the (red) labels' values are in respect to the following subsections *(i.e., section 4.label)*.

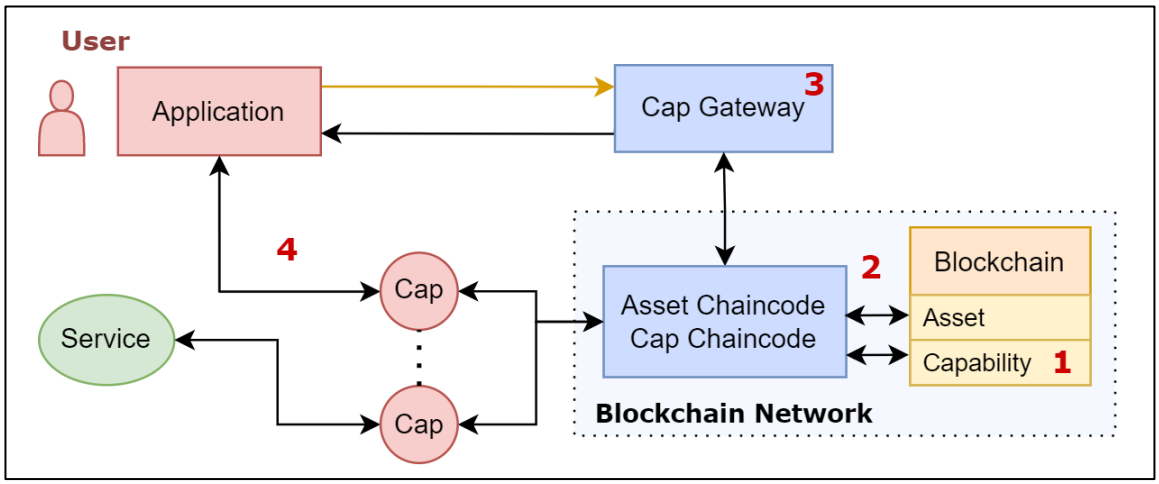

Figure 3: Basic structure of the proposed Object Capability model

First the life cycle and purpose of capability objects for the proposed solution are discussed (4.1), followed by the functionality of the blockchain and smart contracts (4.2). Afterwards the role of the 'capability-aware' gateway (4.3) and how applications can use the cap objects to perform actions on the network (4.4) are explained.

4.1 Capabilities

Capabilities are objects distributed over subjects that grant access to the blockchain network for a certain set of permissions. Capabilities perform the AC-bookkeeping themselves by containing the owner's set of privileges to make access control decisions. In addition, once capabilities are obtained by subjects, they provide direct access to the blockchain network by encapsulating a gateway object. In essence, the public methods of a capability object provide the API for users of the system as any interaction happens through

the capability itself. An example of such an interaction are the delegation and revocation techniques. If a delegation request is granted, the calling cap will create a new cap owned by the receiver and keep a link to the newly created cap. If a revocation request is granted, this link will make sure that the revoked capabilities are removed from the blockchain storage thus declaring any possible further use as invalid.

4.2 Blockchain and Smart Contracts

The underlying blockchain of the network is used to store assets and capabilities. An asset is a digital representation of any resource and the abstract notion of what an asset can convey is maintained throughout the model and has no direct effect on it. An example of assets, e.g., in a file system, are files, directories, and executables. The blockchain's smart contracts manipulate data on the ledger through various functions. The proposed model only adds one contract that manages the capabilities for the system.

4.3 A Capability-aware Gateway

Gateways are the bridge between 'low-level' blockchain network details and 'high-level' functionality. They serve as the primary connection point for applications by directing requests to the right peer in an everchanging network topology. In our solution, the gateway is responsible for distributing capabilities. To provide capabilities with direct access to the network, the outgoing cap-objects are encapsulated with another minimal gateway object that contains the right settings for the requesting user.

4.4 Applications

For subjects such as applications that connect to the network, the capability objects that are returned by the gateway are used for accessing and performing operations on the network. An example depicted in in Figure 3 is the delegation of a certain capability to another subject 'Service'. This is possible by retrieving the list of reachable subjects and then sharing the capability with the desired subject. This will create the delegated capability in the system and if Service requests its capabilities or (re)connects to the network, it will obtain the delegated capability.

5 Implementation

We implemented a prototype of our approach called JOBAC: a capability-based Access Control Model (ACM) for permissioned blockchains [1]. Writing such an ACM requires a framework with enough customisability to be able to represent this model. Segment 5.1 discusses two different frameworks and why Hyperledger Fabric is chosen for this implementation. This is followed by information regarding JOCBAC such as the setup and structure of the model (5.2). The chapter ends with an overview on the main components of a Hyperledger Fabric network and how access control is natively adopted (5.3).

5.1 Permissioned Blockchain Frameworks

Many permissioned blockchain platforms exist in which access control is customisable. Based on the target domain and quality of documentation, Hyperledger Fabric and Ethereum were considered for implementing the object capability access control model.

Ethereum is an open-source platform for writing decentralised applications. It makes use of the flexible programming language Solidity, which is similar to JavaScript and Python. In addition, Ethereum provides ways for writing and testing applications making it easier to build custom blockchain solutions. However, the core model suffers from privacy**,** latency, and the prices (gas fees) accompanied with using the network.

On the other hand, **Hyperledger Fabric** (Fabric or HLF for short) is an open-source framework for building permissioned blockchain networks and applications hosted by the Linux Foundation. It is a solid option for business-to-business models which do not rely on a (crypto-) currency to power the network. HLF is also flexible and modular, but instead it uses general purpose programming languages such as Go or JavaScript, resulting in a familiar way of developing smart contracts. In addition, there is no possibility of ledger forks due to the available consensus mechanism in Fabric being deterministic, meaning any output is final. In addition, HLF is known to offer an efficient transaction scheme, resulting in a throughput estimated around thousands of transactions per second. In comparison with Ethereum, HLF does not require a fee for performing transactions on a network. The previously mentioned arguments are in line with the reasons why related work commonly use Ethereum and HLF as their framework for implementing a proof-of-concept, according to the literature review by Bashir Dar A. et al. (2020) [18] (7.1).

The welcoming features of Hyperledger Fabric such as documentation, customizability, easy smart contract development and the target of enterprise applications determined the researcher's choice for using this framework.

5.2 JOCBAC

The Proof-of-Concept called '**Jannes' Object Capability Based Access Control**' (JOCBAC) [1] is an object capability-based access control model implemented in Fabric to protects data and executable logic stored on a permissioned blockchain. The model is developed for multiple organisations collaboratively working together operating on such a blockchain infrastructure. This blockchain platform requires a way to enforce access control in a flexible manner without creating a major overhead for system administrators. This flexibility means that JOCBAC should be able to manage organisations, users, services, etc joining or leaving the system, whilst maintaining a high and extendable number of available options when deciding on how access is granted.

This chapter starts with discussing the assumptions that JOCBAC makes on a more technical level (5.2.1). Afterwards, the underlying network that is used for simulating an active environment for JOCBAC called Test-Network is explained (5.2.2) and is followed by an overview of JOCBAC's structure (5.2.3). The section on the implementation's structure is analogue to the Approach (4) but with added implementation details.

5.2.1 Assumptions

JOCBAC is a proof-of-concept or prototype, not a functional end-product. This means that the implementation is not fully secured nor designed for a specific production environment. In this model it is assumed that businesses have some form of identity management for registering and enrolling their users to the network with valid X.509 certificates. In this way, companies can still make use of Fabric's built-in ACL for defining a hierarchical structure of their organisation including clients, peers, admins, and organisational units such as affiliations or departments (5.3.1). These roles are mainly used for defining endorsement policies (5.3.2) but can potentially be integrated in the capability system if required, resulting in a hybrid model. To continue, JOCBAC does not include ways for managing or editing blockchain-level configurations, for example adding a new organisation or changing endorsement policies through capabilities. In addition, the way that subjects are discoverable to each other remains untouched because this is dependent on a practical environment where a set of rules are present that define which subjects can reach which subjects. For the same reason, custom options such as a time limit are also not implemented. The use of immediate object references such that accessing a resource is literally invoking the reference is not possible since references of the blockchain's internal state are meaningless on a higher level of the network. To work around this issue, JOBAC uses

abstractions such that the properties of object capabilities do remain. Fundamentally, JOCBAC is used to verify that design concepts of the object capability model are feasible, i.e., a proof-of-concept.

The set-up used for experimenting contains an Intel® Core™ i5-7200U @ 2.50GHz x 4 CPU, with 8 GB of RAM running on the Ubuntu 20.04.4 LTS 64-bit operating system. The specific versions of used software are listed next.

- Hyperledger Fabric version 2.4
  - Fabric-CA-Client 2.2.4^
  - Fabric-Network 2.2.4^
- Docker version 20.10.12, build 20.10.12-0ubuntu2~20.04.1
- Docker-compose version 1.29.2, build 5becea4c
- NPM version 8.5.5
- Node version 16.14.0

5.2.2 Fabric Test-Network

This paragraph discusses the 'low-level' mechanisms for setting up and starting the Fabric project called 'Test-Network', which is used as the underlying blockchain infrastructure for showcasing the JOCBAC model. There is no need to design a completely new network, as this is part of the internal blockchain functionality and management of an organisation's internal structure and are therefore out of the scope of this thesis. This means that no changes were made to the Test-Network endorsement policies, settings of the Fabric Certificate Authority for managing identities, scripts for packaging and deploying chaincode, scripts for managing channels on the network, etc. This results in JOCBAC being separated from the blockchain network's underlying functionality as much as possible. Research on the mentioned processes has been conducted to acquire an understanding on where and how changes should be made when designing the capability-based ACM.

The Test-Network consists of two organisations called 'org1' and 'org2', and an orderer organisation called 'exampleOrderer'. Each organisation has an CA-admin, peer, user, and an org-admin, but the orderer organisation only has a CA-admin and admin. The CA-admins are in charge of handling certificates and the other admins are used for registering and enrolling peers to the network and managing the MSPs (more details on these components in 5.3.1.2). Docker is used for development and simulation of a distributed network and uses Containers to function. These Docker Containers are virtual microcomputers that

conduct a specific task and are created for peers, ca's, the cli interface, and the orderer when starting the network.

The JOCBAC model is deployed by running the 'custom-network.sh' script. This will first initiate housekeeping and afterwards run the Fabric test network start-up script called 'network.sh', with the correct set of arguments for initializing the capability model. The 'network.sh' script carries out the following steps:

- Fabric-CA setup:
    - Generate MSPs and create certificates for the organisations;
    - Enrol and register users to the network.
- Create a genesis block and channel (endorsement of channel config);
- Install custom chaincode (endorsement of chaincode).

After these steps are completed successfully, the network runs on localhost with a channel called 'mychannel'. Valid users and applications of the participating organisations can now join this channel and invoke chaincode. New channels can also be added and JOCBAC will continue to work after the correct chaincode is installed.

5.2.3 Structure

This paragraph contains an overview on how JOCBAC manages capabilities and the lifecycle of capability objects. This is done by explaining the basic structure of the implementation analogue to the Approach (4) and thus also illustrated in Figure 3, the (red) labels' values are in respect to the following subsections *(i.e., section 5.2.3.label)*. All code is written in JavaScript, GO is also a possibility for writing chaincode in Fabric providing a statically typed programming language. Since the gateway and application logic is written in JavaScript, the researcher instead opted for a homogeneous way of programming, i.e., everything in JavaScript.

5.2.3.1 Capability class

Table 5 describes the Capability class used by the JOCBAC model. The following paragraph contains some technical descriptions because it is important to express how the system is implemented to further validate how the theoretical properties translate in a practical environment. *Strings are used for ids and identifying users, the type Any is used for the content of an asset, and Arrays of Booleans represent rights.*

Capability		
Field	*Type*	*Description*
`ID`	`String`	Unique identifier.
`Owner`	`String`	Name of the owner.
`RR`	`Map<String, Boolean[]>`	Resource-Rights: the name of a resource mapped to the rights of this capability (read, write, execute).
`CapRights`	`Boolean[]`	Privileges depicting creation of assets and delegation/revocation of the capability.
`Options`	`Options`	Custom options
`DelegatedChildren`	`String[]`	IDs of capabilities that have been delegated from the current one.
`#NetworkObject`	`NetworkObject`	Contains chaincode and gateway objects for the capability to access the network and invoke contracts.

Table 5: JOCBAC's Capability Class

An instantiation of the capability class represents one capability object that grants certain permissions on the network. The Capability class consists of the fields listed in Table 5 starting with the 'ID,' which is a unique identifier for a capability object. In addition, the class stores the name of its owner for basic access control such as to counter misuse of a certain capability by other subjects. The field 'RR', which stands for Resource and Rights, contains a 'Map', or dictionary, for all the resources this capability provides access to, mapped to the actions that are allowed to be performed on that asset. To continue, the field 'CapRights' contains two Booleans which represent the privileges to create an asset and if delegation of the current capability is possible or not. This field is used to increase the amount of confinement options when delegating a capability, conforming to the principle of least privilege. The 'Options' field and type represent custom features or extensions that can be implemented. For example, if a time restriction is needed on a capability, the options field could contain a timestamp and maximum added time before the capability deletes itself or certain privileges. The field called 'DelegatedChildren' keeps track of capabilities that are created by delegating the current capability, by storing their ids in an Array. This is used for effectively revoking access to the delegated capabilities when its requested. The '#NetworkObject' is a private field, *indicated by a '#' in JavaScript*, and is empty by default. A capability instance is defined and stored on the blockchain without the value of the '#NetworkObject' field, but when in use by a subject it contains chaincode

and a gateway object. These objects are used by the capability to access the network and invoke contracts through this connection.

Capability Methods

There are public and private methods in a capability class. In essence, the public methods provide the API for users of the system as any interaction happens through the capability itself. The public methods manage logic for access control using the capabilities various field's values and the owner's identity information. They then call private methods that invoke contract functions, or they invoke chaincode themselves. When a capability does not have the rights to perform a certain action, for example updating an asset, the cap owner can still call the method '`updateAsset`', but it will not succeed as the capability will deny the operation.

Capability Delegation and Revocation

The mechanism for a controlled delegation and revocation of privileges is realised as followed. When delegating rights, a new capability for the provided owner is created and the correct entries of the caller's `RR` (Rights and Resources) field are copied over to the new capability. The id of the newly created capability is stored in the original capability's '`DelegatedChildren`' field for later possible revocation. The choice of subjects to share a capability with is acquired by calling the method '`getUsers`' on that capability. Revocation is realised by deleting the delegated capability from the blockchains private state after recursively deleting his delegated children and so forth. If a subject still owns a copy of a previously deleted capability, it will not be able to use it since the capability chaincode verifies if the used capability is valid by comparing it to the stored version.

5.2.3.2 Blockchain and Chaincode

The underlying blockchain of the network is used by the gateway to store capabilities and by the system for storing assets and distributed logic. Chaincode, Fabric's term for smart contracts (5.3.1.3), manipulates data on the blockchain through various functions. JOCBAC uses two smart contracts, one is for managing assets and the other is for managing capabilities. It is possible to combine these into one contract or file, but they remain separated for abstraction purposes. Both contracts contain functions for basic CRUD operations on their respective data types. These functions are requested by the capability-gateway for initialisation and afterwards by capability objects for manipulating data. The chaincode responsible for simple operations on assets is called '`assetTransfer.js`' and

is based on asset transfer chaincode examples provided by the Hyperledger Fabric Samples repository [22]. The operations on assets are as a matter of course and are not discussed in further detail. The capability chaincode does have other functionalities, for example functions that manage RR-entries and the delegation and revocation of capabilities.

5.2.3.3 Capability Gateway and Applications

The capability gateway serves as the primary connection point for users. It is also ressponsible for initialising the ledger with assets and capabilities for simulating an active network state. When a certain subject wants to connect to the network and therefore interacts with the gateway, the following steps are performed by the gateway.

1. Build an instance of the 'Fabric-CA Services Client' based on the network configuration (CA and MSP management);
2. Setup or read the subject's wallet. *A wallet in Fabric holds identities, not currency*;
3. Create the gateway object, connect to requested channel and receive channel's smart contracts;
4. When requested, initialise ledger with assets and capabilities;
5. Return the subject's capabilities that are stored on the blockchain.

For subjects such as applications, the capability objects that are returned by the gateway in step 5 are used for accessing and performing operations on the network if authorisation is satisfied. To provide capabilities with direct access to the network, the gateway encapsulates any outgoing cap-object by setting the '#NetworkObject' field with the correct parameters based on the incoming subject's information. Capabilities manage access control decisions and are used by applications for directly accessing the network. For simulation purposes, the programmed applications called 'user*x*.js' interact with the network by performing hardcoded actions because there is no GUI implemented as this is out of this thesis' scope.

5.3 Hyperledger Fabric

This chapter contains an outline on the basic components required for a Fabric blockchain network to function such as channel, orderer, gateway and more (5.3.1). Followed by critical information regarding access control in Fabric, including the several types of policies and how custom access control is natively approached (5.3.2).

5.3.1 Components of Hyperledger Fabric

A high-level overview on the different components of Fabric and how they work together to manage a blockchain network is presented in the next paragraph. This overview

is based on the documentation provided by all pages of the Hyperledger Docs [23] but mostly the 'Key Concepts' page. Bear in mind that this research uses version 2.4 of Hyperledger Fabric which is structurally different from previous versions. An overview on the exact differences can be found in the 'What's new in HLF v2.*x*' page of the Hyperledger Docs [23].

5.3.1.1 Channel and Orderer

In Fabric, a network utilizes **channels** to communicate with other organisations. Each channel has a configuration block and one or more **ordering** services. The configuration block contains endorsement **policies** about the channel and the initial set of organisations that are allowed to participate. Endorsement policies are a collection of rules that define the structure of decision making, they exist on multiple levels in a network and are explained in more detail in section 5.3.2. An ordering service is always managed by a certain organisation, provides basic endorsement access control for a channel through these policies, and collects verified transactions from applications to order them into transaction blocks.

5.3.1.2 Certificate Authority & Membership Service Provider

Certificate Authorities, or CAs for short, are present in a network for identification and can/should be different for each organisation. It creates and manages certificates which identify a participant from a certain organisation. For testing purposes, Fabric comes with a build-in CA called 'Fabric-CA' which manages certificates in a test environment. The **Membership Service Provider** (MSP) is tied to a root CA to identify entities created by this root and thus defining an organisation and its roles in a tree structure. Every actor on a network has a digital **identity** using X.509 digital certificates that decide what endorsement permissions the identity has. Identities in Fabric can contain organisational units i.e., a set of roles such as peer, peer-admin, channel-admin, which can be configured at organization, node, and channel levels. Organisational units are a way for Fabric to define a separation between identities that are created by the same Certificate Authority. An identity also includes additional attributes for HLF to govern permissions which are called principals. In essence, the CAs and MSPs define and conduct the basic operations for creating, updating, and verifying certificates and identities along with organisational units and principals.

5.3.1.3 Gateway, Chaincode

Peer nodes are essential to a blockchain network and provide the **Fabric Gateway** as a service through a set of APIs. If a peer with a valid certificate joins a channel, it receives a copy of the ledger and physically hosts this copy. Gateways are also responsible for evaluating, endorsing, and committing transactions on a blockchain network. Changes to the channel configuration can only be made if there is an active ordering service on the channel. Other operations require peers to install **chaincode**, which is Fabric's structure for one or more smart contracts packaged together and therefore the terms are used interchangeably. Chaincode contains an endorsement policy and if not, it is inherited from the channel's configuration file. This policy describes what organisations or set of members need in order to verify transactions before their output is accepted by others.

5.3.1.4 Application and Private Data

An **application** is identified by the organisation it is operating on and is used to invoke transactions. This is only possible after it has been connected to a channel via the gateway service and chaincode is deployed on that channel. To store the states of peers, Fabric has the option to use either LevelDB or CouchDB for flexibility in the type of database management system, i.e., relational or graph. The HLF docs make the statement that if the JSON format is used for storing data, the CouchDB option is better since it can then provide indexes for chaincode, resulting in a more adaptable and structured way of querying data especially when using large datasets.

Fabric has a notion of protected **private data collections**. This is used when certain data should not be accessed by all organisations on a certain channel, without creating a new channel just to access and protect that private data. In a Fabric network it is also possible to invite new organisations to already existing channels and to concurrently connect with multiple channels. Fabric provides abstractions though Software Development Kits, or SDKs, for implementing business logic and smart contracts. Examples are the classes and interfaces such as Channel, Identity, Gateway, Wallet, and Contract.

5.3.2 Writing an Access Control Model in Hyperledger Fabric

After an overview on the method used to design a custom ACM in Fabric (5.3.2.1), this chapter discusses Fabric's native access control mechanisms and with an important distinction between policies for endorsement (5.3.2.2) and for accessing resources (5.3.2.3). This is followed by the different types of policies that govern access control on different

levels of a network by means of an ACL (5.3.2.4) and the chapter concludes with how AC is natively approached in Fabric (5.3.2.5) relating to the JOCBAC model (5.3.2.6).

5.3.2.1 Overview

An understanding of the Hyperledger Fabric framework and how to implement a capability access control model is partly acquired through hands-on code experience. This experience is obtained starting with the 'Getting Started - Install' page from HLF's 'readthedocs' [23]. This page runs through the prerequisites required to develop a Fabric network and uses the 'Fabric-Samples' repository [22] to display its functionality. The 'Test-Network' project from these samples introduces how a basic network is setup, run, and shut-down, containing default values for MSPs, CAs, etc. The tutorial regarding this testing network explain how to create a channel and perform basic channel configuration, along with how to deploy chaincode on a channel. Running the 'Test-Application' project introduces application-related concepts such as connecting through a gateway and how to perform CRUD-operations on assets. To analyse how Fabric manages access control and how this can be customized in-depth, the page 'Developing Applications' from HLF docs uses the 'Commercial-Paper' example and is the first complex business scenario introduced in the docs. In this example, key concepts that introduced the Fabric blockchain framework are explained in more technical detail, such as how data is represented, the life cycle of smart contracts, how to develop a gateway, how to manage policies, MSPs etc.

5.3.2.2 Endorsement

The endorsement of a transaction means verifying its validity by collecting signatures from a certain set of members before any output is accepted as final on the ledger. A member returns a signature when evaluating the transaction returns the same output as the one that is proposed. The number of required signatures and which set of members participate in this process are defined in the endorsement policies. This process is similar as reaching consensus, but the latter is the final phase and implies that all intermediate endorsement steps have been completed.

5.3.2.3 Resources

Access control for resources can be enforced through endorsement policies as they contain the possibility to define rules that govern which (type of) peer can or cannot invoke specific chaincode functions, essentially mapping subjects onto chaincode functions. Access control is then situated in the endorsement process. However, since Fabric stores these policies in ACLs the discussed problems that arise when using traditional ACLs are

now introduced, e.g., new participants and new rules need to be configured in policies manually which does not result in a dynamic model (2.2.1 and 2.2.3). Thus, endorsement-sided policies are not relevant for this thesis as they are not about access control for resources, at least not in an optimal manner, and are primarily used for managing the internal functionality of a blockchain network.

5.3.2.4 Types of Policies

Three types of policies are used in a Fabric network and are shown in Figure 3, each one enforces rules on a different level of a network.

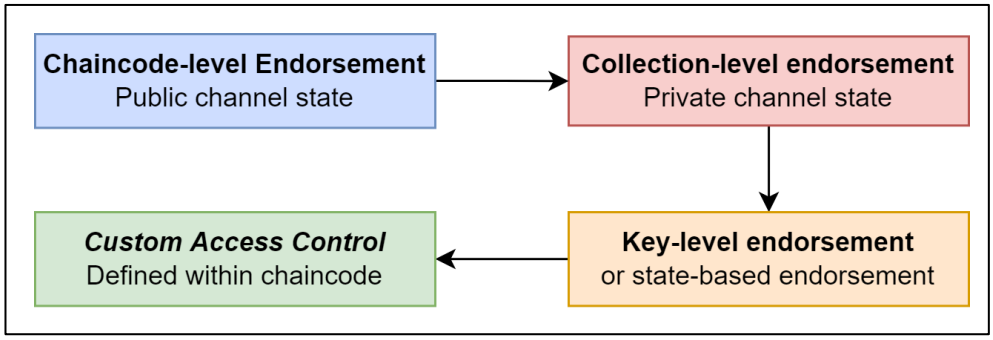

Figure 4: Three levels of endorsement policies in Hyperledger Fabric

- The 'chaincode-level endorsement' policies are defined on a certain channel. These policies enforce rules for the public channel state, with public including everyone that has access to that channel. *Deployed chaincode operates on channel-level, thus the same as chaincode-level.*
- The policies on the 'collection-level endorsement' are also defined on the channel and override the previous 'chaincode-level' policies. This is used for any key in the private data collection or private channel state.
- 'Key-level' or 'State-based' policies are managed within chaincode and override the previous mentioned policies for specific keys. An example is to create an asset that is owned by only one organisation. *It is possible to deploy this chaincode on different channels, thus not chaincode nor channel-level.*
- Any other access control option, rule, policy, or mechanism is also defined within chaincode. Consequently, this is the level where most extensions of the basic Fabric access control structure are situated, e.g., ABAC, RBAC and CapBAC implementations.

5.3.2.5 Custom Access Control

According to the HLF docs, the manner in which access control to manages resources is promoted to be written is by writing logic in chaincode, o*r as previously mentioned by writing ACL rules in the endorsement policies to explicitly define which subjects can invoke which chaincode functions.* The proposed chaincode logic collects the user's identity from transaction context to make the access control decisions. This identity contains basic information regarding the user for example the user's name, organisation, location, certificate, and other possible attributes. The example that the Fabric docs provide is to store the user's identity as a property of a resource and verify the equivalence when access is requested [24] [page Tutorials, Writing Your First Chaincode, Chaincode Access Control]. Another possibility is to design an ABAC-based ACM using the (Fabric-) CA to modify certificates with custom attributes.

5.3.2.6 Conclusion Access Control in Fabric

While it is easier and faster to setup a basic ACL, RBAC or ABAC model, the overhead of manual editing rules, code, and/or attributes grows considerable in size with the complexity of the system. The term 'complexity' includes the number of users, files, organisations, affiliations etc, and in what degree these can change. To minimise long term overhead, JOCBAC extended Fabric's functionality with an object capability access control model without relying on the underlying endorsement ACL nor custom identity attributes for access control decision making. In this way, organisations can still make use of the underlying ACL and attributes for defining a hierarchical structure of their organisation in combination with the capabilities from JOCBAC.

6 Validation

JOCBAC functions how it should as described in Approach (Chapter 4). The underlying blockchain is used for distributed storage of capabilities and one smart contract is added to manage this. The capability-gateway distributes capability objects for the subjects in the system. These objects make access control decisions themselves separated from the underlying ACL and are usable by subjects to interact with the system. If the correct permissions are satisfied, the capability object can invoke smart contract functions through his encapsulated gateway and provide direct access to the blockchain. The JOCBAC model also includes safe delegation, revocation, and confinement techniques. The section involving the structure of JOCBAC (5.2.3) is organised parallel to the approach and explains the

following features with added implementation details. Therefore, JOCBAC is a 'true' object capability model since the properties that distinguish object capabilities from other models all hold. These properties are mentioned in section 2.2.3: Access Control Lists vs Capabilities and summarised in Table 3. The next paragraph explains the mentioned properties in relation to the prototype JOCBAC (6.1), followed by an example that portrays the usefulness of JOCBAC in comparison to the default ACL model (6.2).

6.1 Object Capability Model

This paragraph compares the mentioned properties to the JOCBAC model in more detail, to verify that JOCBAC classifies as an Object Capability model in Table 3.

6.1.1 Unified Designation and Authority

This property is fulfilled in JOCBAC, as a capability both serves as an object designator and provides the permissions for that object at once, i.e., it contains the asset's id and the rights concerning that asset. If a subject or asset is not accessible through a certain capability, the asset or subject will not be present in the capability's fields (2.2.3.1: Unified Designation and Authority).

6.1.2 Distribution

The use of a blockchain for storage and for executing smart contracts distributes data and business logic over peers in the network. Capabilities that are in use by subjects are objects on the client-side, meaning they are automatically distributed over the subjects by design. The capability-gateway is an access point used for initialisation of the ledger and for initial connections to the network and is the only centralised part of JOCBAC, therefore this model is highly distributed (2.2.3.2: Distribution).

6.1.3 Dynamic Subjects

Any type of authenticated subject can potentially use the capability-gateway to connect to the network and receive capabilities for usage, this includes applications, users, services, processes, etc. This results in a dynamic possibility of subjects in the system (2.2.3.3: Dynamic Subjects).

6.1.4 Least Privilege

The Principle of Least Privilege (PoLP) is maintainable on account of property 'Unified Designation and Authority' along with confinement mechanisms and high context possibilities. The confinement of a capability is possible in various ways, a field can be modified to control the creation and delegation privileges of a delegated capability and in addition, a subject can delegate a full capability's privileges or only select the privileges for a specific asset. A large number of in context options for access control decisions also makes it easier to conform to the PoLP, and even more customisability is possible by implementing the required logic, e.g., custom timing options (2.2.3.4: Principle of Least Privilege).

6.1.5 Ambient Authority

Ambient authority is removed from the system due to four reasons: Firstly, since property 'Unified Designation and Authority' is fulfilled, secondly because a capability object accesses the network through itself, thirdly since only one capability is used per action per subject, and lastly because the subject itself is responsible for selecting the 'key' it wants to use, i.e. selecting which capability to use for a accessing a specific asset. Because of this, a subject cannot use a capability's privileges to perform actions with another capability, thus removing the possibility of a confused deputy from the system (2.2.3.5: Ambient Authority).

6.1.6 Controlled Delegation

The controlled delegation property is fulfilled when a working mechanism is present and properties 'Unified Designation and Authority' and 'Dynamic Subjects' are satisfied. This results in no possibility to delegate a certain capability without having the required permissions to do so or without the required access to the subject. Nested delegations can be easily revoked, and confinement techniques help to control the delegation process (2.2.3.6: Controlled Delegation).

6.2 Example JOCBAC vs ACL

The usefulness of JOCBAC is compared against the ACL model by means of an example; A subject called User wants to share its access to a certain asset with another subject called Service. The JOCBAC version is illustrated in Figure 5 and Figure 6, and the ACL version in Figure 7 and Figure 8.

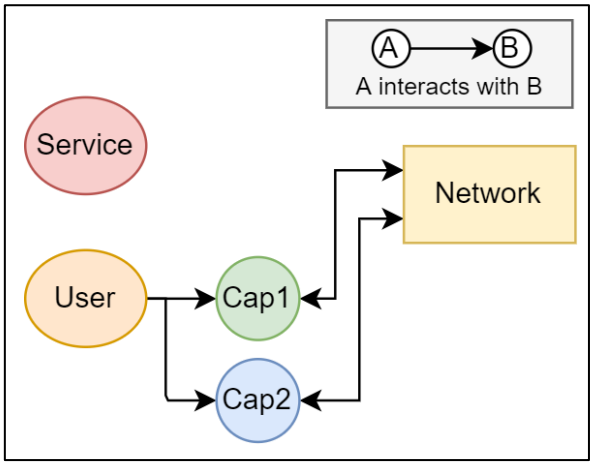

Figure 5: JOCBAC before example delegation.

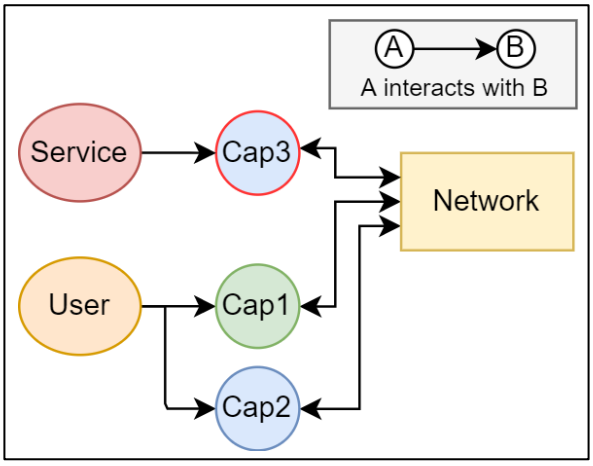

Figure 6: JOCBAC after example delegation.

It is assumed in this example that Service has a valid identity (MSP and CA) such that it can be authenticated by the capability gateway. It is also assumed that User already owns a capability containing a correct set of privileges to access Asset2, a reference to Service, and the rights to delegate this access further on which is the blue capability called 'Cap2' in Figure 5 and Figure 6. Initially, the service does not hold any capabilities in this example. When the user shares his capability, it first requests all subjects it has access to and selects the wanted subject as argument for the delegation request. Both occur by calling methods of the capability object itself. When privileges are correct, the network will store the created capability, called Cap3 in Figure 6. If Service now connects to the network, or requests its currently owned capabilities, it will receive the delegated Cap3 from the gateway and through that object it can now access Asset2.

The previous example is reiterated but now with an Access Control List (ACL) as the Access Control Model (ACM).

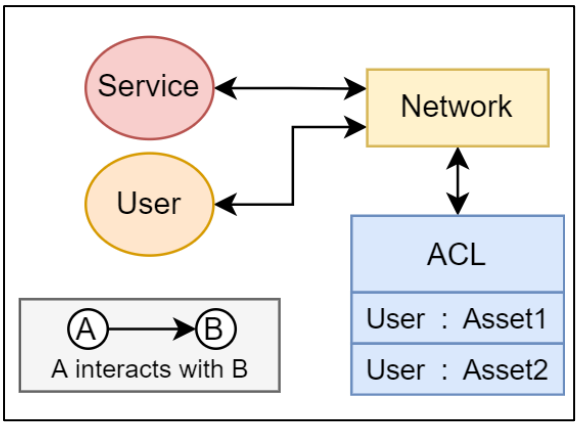

Figure 7: ACL before example delegation.

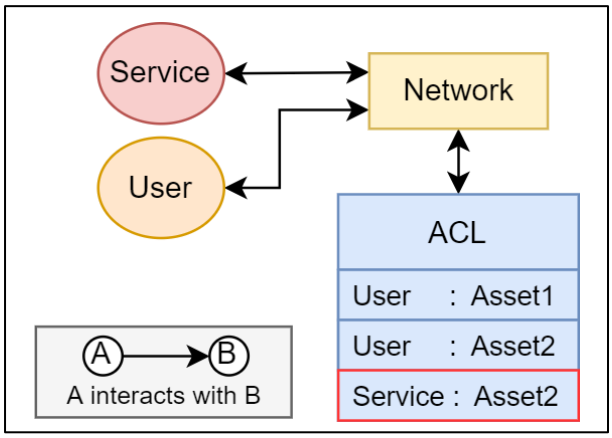

Figure 8: ACL after example delegation.

If in this model the user wants to share his access to Asset2, the existing access rule "User : Asset2" needs to be copied over and then modified with the new owner. By default, the user cannot initiate this sharing of rights to happen thus an external component will have to edit the ACL line by line. The users can be grouped together (RBAC) or individually given more attributes (ABAC) to make the editing of the ACL-matrix less heavy, but this does not solve underlying problems with ACL.

There is a clear difference in underlying functionality between the two models including already discussed distinctions. The foremost difference is the 'side' on which the access control happens is flipped from subjects to network. This means that in the ACL model, or 'the network side', all operations happen through the network which results in a highly centralised model. In addition, there is a mechanism required to manage the ACL-matrix line by line. This is possible either manually or automated, but both require a lot of effort, time, and a large overhead to maintain. Consider the case of revoking the delegated access rule "Service : Asset2" from the previous example in an ACL model: which authority is going to manage the link from an original entry to the delegated entry? How are nested delegations maintained and revoked? The answers usually imply adding more information per line accompanied by a more complex system for managing the matrix. For capability models the answer is the capability object itself. From the user-perspective, JOCBAC is more user-friendly and dynamic, and it automates work that otherwise should be done by an administrator or a more privileged component.

7 Related Work

Capabilities have promising characteristics for designing Access Control Models (ACMs), resulting in plenty of research trying to perfect this idea for different scenarios and problems. As there exists no general best solution, several papers sharing the main ideas of this research are discussed next. A portion of the paper from Miller M. et al. (2003) [6] has already been mentioned and they conclude with a comparison between ACL and cap models related to several properties similar to Table 3 and an in-depth description of the differences described in the previous sections 2.2.1 and 2.2.3.

7.1 Overview

The analytic **literature review** by Bashir Dar A. et al. (2020) [18] served as an excellent starting point for studying related work and consists of a systematic review on blockchain driven access control. They gathered seventy-six papers regarding this subject to describe

and compare them in detail. These systems are based on a wide variety of models, from ACL, RBAC and ABAC to capabilities and other blockchain oriented ACMs.

In this study, almost four out of ten of the collected papers are targeted towards the IoT domain, which they divide in the following subcategories: *"the Internet of Drones, Smart City, Smart Grids, Industrial IoT, Smart Homes, and Smart Buildings"* [18, pp. 9-10]. The second and third most used domains respectively are Cloud and Healthcare applications, each with one out of ten papers targeting them. In their classification scheme, this thesis would be categorized as an 'Enterprise Application', which is currently just under 3% of their collected papers. Research that is more in-line with this thesis, i.e., targeted towards private enterprise use, commonly do not use capability-based models. Instead, they tend to tilt towards attribute or role-based access control (RBAC & ABAC), generally implemented with smart contracts and custom gateways.

The literature review also includes an overview of issues on security and privacy that are frequently present in ACMs, together with proposed blockchain-based solutions from the gathered papers [18, pp. 19, Table 5]. Their findings state that almost four out of ten papers use the Ethereum network as a blockchain platform to conduct implementations or simulations and Hyperledger Fabric (HLF) being the second most used.

7.2 CapBAC models

7.2.1 DCapBAC

DCapBAC [13] is a distributed capability model for the IoT, where access control is embedded on resource-constrained devices themself. This model does not use a blockchain platform but even so, it demonstrates the potential of the capability model for distributed solutions.

7.2.2 CapBAC

To introduce and compare blockchain capability-based ACMs, the mentioned literature review [18], the overview by S. Pal et al. (2021) [25], and the classification by A. I. Abdi et al. (2020) [26], reviewed different cap-based models trends, problems, and needs based on blockchain technology. Most of the research involving the implementations of cap-based ACMs indirectly originated from the previously introduced CapBAC models [16, 17], including the CapBAC model by B. Anggorojati et al. (2012) [19]. These three papers share the same ideas but have different use-cases or implementations. However, each provided an addition to understanding the general capability model. CapBAC overcomes RBAC and ABAC issues regarding scalability and flexibility by utilizing capability tokens for more

granularity of access control rights. Moreover, it has effective delegation and revocation procedures and can ensure the principle of least authority.

7.2.3 BlendCAC

'BLockchain-ENabled, Decentralized, Capability-based Access Control' or BlendCAC, developed by R. Xu et al. (2018) [20], is an addition to the previous CapBac models and also uses a blockchain for storage of tokens with self-enforcing policies. When a user requests access, a capability is generated and stored using smart contracts. However, the proposed system is outperformed by a newer implementation of CapBAC by Nakamura Y. et al. (2020) [27], with almost the same costs and latency. BlendCAC uses two types of tokens and a delegation tree, whereas Nakamura Y. et al.'s proposal only uses one token and a delegation graph for more flexibility.

7.2.4 IoT-CCAC

The 'IoT-Consortium Capability-Based Access Control' model or IoT-CCAC, created by M. A. Bouras et al. (2021) [21], is a distributed capability-based ACM that bridges the gap between poor performance of blockchains and centralised ACMs. In addition to the default capability properties, it introduces a group-capability and achieves high granularity through a wide variety of representable context, such as time, location, used protocol and more. This system is one of the few proposals using a database-specific blockchain called BigChainDB [28], they state that "*a blockchain database can do the same* [as other private blockchains] *with the* [sic] *even better performance*" [21, p. 2].

7.2.5 Conclusion: Related Work

The papers mentioned in this chapter contributed to the understanding of current state-of-the-art technology and research that led to it. In addition, they opened different ways to approach the thesis problem and boundaries. For example, an understanding on how different ACMs operate, such as ACL, RBAC and ABAC is obtained through reading and comprehending the introduction, background, and related work sections of articles [20], [21], [25], [26] and [27]. These articles also describe the transition from traditional access control to blockchain-based access control.

8 Conclusion

This paper proposed an object capability (ocap) Access Control Model (ACM) for business-to-business permissioned blockchain networks that overcome the limitations of the commonly used Access Control List (ACL) model. Related work in the form of former research and state-of-the-art implementations highlight that there is a lack in the object capability ACMs on top of blockchain technology, as opposed to ACL-based implementations and capability models implemented with keys. We delivered an understanding about concepts such as blockchain and access control and learned that blockchains provide a secure way for storing data transparently and that a permissioned blockchain is defined as a chain with an additional AC layer on top of it. We also discovered that the evolution of the capability model includes multiple iterations, each solving certain problems resulting in the true object capability model. Furthermore, the comparison between ACLs and ocaps presents problems that arise when using ACLs and show how ocaps propose a solution. The comparison of the ocap model with state-of-the-art underlines the shortage of ocap models for enterprises. A structure of the proposed solution explains the roles of the components in this model. For example, the gateway distributes capability objects over subjects such as applications. These capability objects are the AC-bookkeepers of the system, providing direct access to the network if the privileges allow them to do so. This results in a distributed and dynamic system in which work is automated that is normally carried out by an administrator or privileged component. To implement JOCBAC, a prototype of the approach, we discussed the Hyperledger Fabric framework and its components resulting in an understanding on how Fabric's elements interoperate and ultimately create a secure blockchain network. A deep dive on the main components involved when designing an ACM such as gateway and chaincode resulted in the setup and structure of the proof-of-concept. JOCBAC proves that the concepts depicted in the approach effectively translate into this environment and that JOCBAC provides a more user- and administrator-friendly model as opposed to ACLs. In addition, the JOCBAC model provides fine granularity of rights, delegation, and revocation properties, and respect the principle of least authority without the possibility of having ambient authority which results in confused deputies.

8.1 Future Work

Because of the complexity involved in this interesting subject and the relative lack in time, two main shortcomings of this thesis are described next. Firstly, the extent to which the implementation in Hyperledger Fabric called JOCBAC is generalisable to other (permissioned) blockchain frameworks such as Corda, Ripple, or database-driven blockchains such as BigchainDB etc, is not untangled. This would be useful since it could add abstraction power to the prototype. Secondly, features that include a more technical mapping of the internal Fabric functionality to the capability system are missing. An example to clarify this functionality is adding the notion of 'network capabilities', which contain privileges for adding organisations, altering endorsement policies, and thus manipulate the internal ACL. These manipulation strategies could help generalise the conceptual model to other frameworks that inherently provide an ACL as ACM, and that potentially do not allow for the freedom in design choices which Fabric does. Another possible feature is a more in-depth 'interface' for subjects to connect to a network, resulting in identity management being part of the model. This feature however could lower the abstraction power of JOCBAC as identity management is more dependent on the framework's internal functionality. Nevertheless, the concepts of JOCBAC are sufficiently abstracted to make designing a new prototype in other blockchain frameworks straightforward.

References

[1] J. De Mets, "JOCBAC's GitHub repository," 2022.

[2] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper,* pp. 1-32, 2014.

[3] E. Androulaki and Et al., "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," *Proceedings of the thirteenth EuroSys conference,* pp. 1-15, 2018.

[4] S. Nakamoto, "A peer-to-peer electronic cash system," *Bitcoin,* 2008.

[5] J. Morgan, "Quorum Blockchain Service," Consensys, 2022. [Online]. Available: https://consensys.net/quorum/qbs/.

[6] M. Miller, Y. Ka-ping and J. Shapiro, "Capability Myths Demolished," 2003.

[7] "[cap-talk] "Capability Myths Demolished" review," eros-os, 2003. [Online]. Available: https://archive.is/20130414162939/http:/www.eros-os.org/pipermail/cap-talk/2003-March/001133.html.

[8] D. D. Redell, "Naming and Protection in Extendible Operating Systems. Project MAC," MIT, Cambridge, 1974.

[9] A. H. James, "A Microprogrammed Operating System," University of Cambridge Computer Laboratory, Cambridge, 1982.

[10] M. S. Miller, "E: Open Source Distributed Capabilities," [Online]. Available: http://www.erights.org/.

[11] M. S. Miller, "E-Rights Wiki: Main Page," 21 April 2018. [Online]. Available: http://wiki.erights.org/wiki/Main_Page.

[12] "Hack," Meta, MIT, [Online]. Available: https://hacklang.org/.

[13] J. L. Hernández-Ramos, A. J. Jara, L. Marín and A. F. Skarmeta Gómez, "DCapBAC: embedding authorization logic into smart things through ECC optimizations," *International Journal of Computer Mathematics,* vol. 93, no. 2, pp. 345-366, 2016.

[14] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE 63,* p. 1278–1308, 1975.

[15] N. Hardy, "The Confused Deputy: (Or Why Capabilities Might Have Been Invented)," *SIGOPS Oper. Syst. Rev.,* vol. 22, no. 0163-5980, p. 36–38, oct 1988.

[16] C. Bistarelli, C. Pannacci, F. Santini, J. Pereira and L. Ricci, "CapBAC in Hyperledger Sawtooth," *Springer,* 2019.

[17] S. Gusmeroli, S. Piccion and D. Rotondi, "A capability-based security approach to manage access," *Elsevier,* 2013.

[18] A. B. Dar, "Blockchain Driven Access Control Mechanisms, Models and Frameworks: A Systematic Literature Review," *SSRN,* 2020.

[19] B. Anggorojati, P. N. Mahalle, N. R. Prasad and R. Prasad, "Capability-based Access Control Delegation Model on the Federated IoT Network," *IEEE Press,* pp. 604-608, 2012.

[20] R. Xu, E. Blasch, Y. Chen and G. Chen, "BlendCAC: A BLockchain-ENabled Decentralized Capability-based Access Control for IoTs," *Cybermatics,* 2018.

[21] M. A. Bouras, B. Xia, A. O. Abuassba, H. Ning and Q. Lu, "IoT-CCAC: a blockchain-based consortium capability access control approach for IoT.," *PeerJ Computer Science,* 2021.

[22] Fabric, "GitHub repository main branch: hyperledger/fabric-samples," Hyperledger, [Online]. Available: https://github.com/hyperledger/fabric-samples.

[23] Hyperledger Fabric, "A Blockchain Platform for the Enterprise," Hyperledger, 2020. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/index.html.

[24] "Fabric Docs, Tutorials, Writing Your First Chaincode, Chaincode access control," Hyperledger Fabric, 2020-2022. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.4/chaincode4ade.html#Chaincode_API.

[25] S. Pal, A. Dorri and R. Jurdak, "Blockchain for IoT Access Control: Recent Trends and Future Research Directions," *ArXiv,* vol. abs/2106.04808, 2021.

[26] A. I. Abdi, F. E. Eassa, K. Jambi, K. Almarhabi and A. S. A.-M. AL-Ghamdi, "Blockchain Platforms and Access Control Classification for IoT Systems," *Symmetry,* vol. 12, no. 10, p. 1663, 2020.

[27] Y. Nakamura, Y. Zhang, M. Sasabe and S. Kasahara, "Exploiting Smart Contracts for Capability-Based Access Control in the Internet of Things," *Sensors,* vol. 20, no. 6, p. 1793, 2020.

[28] T. McConaghy , R. Marques, A. Müller, D. De Jonghe, G. McMullen, R. Henderson, S. Bellemare and A. Granzotto, "Bigchaindb: a scalable blockchain database. white paper,," BigchainDB GmbH, 2016. [Online]. Available: https://www.bigchaindb.com/whitepaper/.