Fine-grained Access Control for a Blockchain-based Healthcare System

Sem Duveen , Mauro Conti , Chhagan Lal
TU Delft

***Abstract*—Large volumes of medical data (MD) are continuously generated by the healthcare domain. When sharing these data, issues arise regarding privacy and security. To solve these issues, a permissioned blockchain (BC) can be used, but since blockchains do not have access control (AC) as a default feature, the integration of an access control system (ACS) is necessary to ensure the confidentiality of the medical data. The main question that we aim to answer is: How can access control techniques (ACT) be incorporated into a BC-based medical data sharing system (MDSS)? To answer this question, we created an access control system (ACS), based on HyperLedger Fabric, after evaluating existing techniques, with the use of a set of questions, that were chosen specifically for this purpose. Our ACS uses a smart contract, called the Access Contract to restrict access, based on access levels and permission queries, which are stored in the state ledgers of HyperLedger Fabric's world state. The Access Contract defines the necessary transactions for an ACS, in which these variables are used. Our ACS satisfies more metric questions than the related works's average and is thus optimal. We found that AC can be incorporated into a BC-based MDSS, by utilizing smart contracts to define the needed transactions that use access levels and permission queries to restrict the access of users.**

1 Introduction

Healthcare is a domain that continuously generates large amounts of MD from various sources. This makes healthcare a data-intensive domain [10]. This large volume of data needs to be shared among different health facilities. However, healthcare systems are facing several challenges concerning the privacy and security when sharing MD [18]. Solving these challenges is important, because the shared data contain personal information of patients and should therefore remain confidential. According to Dubovitskaya et al. [6], these challenges can be solved by using a permissioned BC to create this MDSS.

A big problem that arises when trying to use BC to create such a system is the lack of AC as a default feature of BC [12]. A MDSS without any ACT would allow people to access data that they are not authorized to. This is unacceptable for a MDSS, since a patient's data should always remain confidential.

Currently, there are many projects that propose a BC-based MDSS. For example, Guo et al. [12] propose a hybrid architecture that uses both BC and edge nodes in order to make AC based on the attributes of the user possible for MD sharing. Other examples are Ancile [3], MedBloc [14] and MedRec [1]. Each of these projects has its own way of integrating AC into their system.

They are, however, not perfect yet. Each of them uses different ACTs and provides a proof, with different metrics, for their optimality. This makes it unclear which ACT is the better one to use for a BC-based MDSS.

The main question is: *How can ACTs be incorporated into a permissioned BC to create such a BC-based MDSS?* To answer this question, we will present the design of an ACS that can be integrated into a permissioned BC that uses the framework HyperLedger Fabric.

This paper's main contributions are that it gives a set of metrics that can be used to evaluate an ACS for MD sharing, shows the strong and weak points of existing works and finally, uses this information to give a design of a fine-grained and adaptible ACS that can be used to integrate AC into a BC-based MDSS.

The structure of the remainder of the paper is as follows. After this introduction, the second section gives the background information needed in order to understand the subject of the paper and discusses related work. First, the third section shows an evaluation of these related works and the metrics that are used. Next, this evaluation is used in order to create a new or improved ACT. Finally, the new ACS is evaluated, based on the same metrics as chosen in section 3. The last three sections contain the discussion and future research, conclusions and responsible research.

2 Background and Related Work

2.1 Blockchain

A BC consists of a chain of data packages, which are called blocks [19]. Each of these blocks stores multiple pieces of data called transactions. A block can be seen as a data structure that consists of two parts, the header and the data. The header contains the hash value of the previous block in the chain, a timestamp, a Merkle Root and some other information. The data part contains the relevant data of the transaction. Since the BC is extended by each block, the BC can

be seen as a complete ledger of the transaction history. This BC is duplicated across the distributed network, where every node has its own copy [25].

2.2 Smart Contracts

A smart contract is a piece of code that governs transactions and defines the conditions of a mutually agreed contract [5]. Smart contracts run and are stored on the BC and are executed by all the participants of the network. Since they function on a publicly shared ledger on the distributed network, smart contracts are immutable, once they have been agreed to run [25].

2.3 Access Control

Currently, there are many different ACTs used for AC. Some noteable examples are Discretionary AC (DAC), Mandatory AC (MAC), Role-based AC (RBAC), Attribute-based AC (ABAC) and Relationship-based AC (ReBAC).

Discretionary Access Control

DAC is a way to restrict access, based on the identity of a user and/or the group they belong to [9]. When using DAC, the owner of an object is able to decide which users should be allowed to access the object and what type of actions they are allowed to perform on the object. [16]. Due to the owner having full authority and control over their object, DAC is also known under the name, owner-based AC [26]. An implementation of a DAC model usually utilizes an Access Control List (ACL), AC Matrix or an AC Capability List [2]. All three are different ways of storing the access levels that are granted to users. DAC is the most commonly used ACT in commercial operating systems, such as UNIX and Linux [2].

Mandatory Access Control

When using MAC, the AC policies are fully managed by the system administrator [24]. This is the exact opposite of DAC, where every user manages the AC policies of their own object. Multiple ACTs exist that use MAC as a base. An example of this is Lattice-based AC. For this MAC model, access is restricted, based on the sensitivity of the information that an object contains [9]. This sensitivity is represented by a label. Each user is also given a label. This user label is checked against the object label to see if it has the same or a higher level of sensitivity. These sensitivity labels generally include TS (top secret), S (secret), C (confidential), R (restricted) and U (unrestricted) [2].

Role-based Access Control

According to Bai et al. [2] DAC and MAC, due to the continuous development of computer applications, hardly begin to satisfy AC demands like adding, canceling and merging of enterprise departments, title promotion, duty change and the leaders' responsibility seperation and restriction. In order to solve these problems with DAC and MAC, RBAC was proposed. To restrict access with RBAC, various roles are distinguished, after which every user of the system is granted one or more of these roles [2]. Access rights are given to each of these roles. Users are only allowed to access an object if they have one or more of the needed roles.

Attribute-based Access Control

ABAC can define permissions based on almost any user characteristic, relevant for security. These characteristics are known as attributes and can have three different types. Subject attributes, which define the identity and characteristics of subjects; resource attributes, which define the characteristics of entities that are acted upon by a user; and environment attributes, which describe the technical, situational and operational environment [4]. Each object is associated with rules, regarding certain attributes. When a user tries to get access to an object, their attribute will be checked against each of these rules. Access will only be granted when all of the rules are satisfied.

Relationship-based Access Control

An ACS that uses ReBAC [11] restricts access based on the relationship that the user, who is trying to access an object, has with the owner of said object. The word owner, when using ReBAC, does not mean the same as it does when used in DAC. In this case the owner does not necessarily have full control over the AC policies of the object. When a user tries to access an object, their relationship with the object owner will be checked. The access request will be rejected, if their relationship is not one of the authorized relationships or if it does not exist.

2.4 Related Work

Guo et al. [12] propose an ACS for a BC-based MDSS. This system has an architecture that is a hybrid between a HyperLedger Fabric BC and edge nodes. HyperLedger Fabric includes a so called ACL, in which permissions can easily be specified. The proposed system uses ABAC as ACT. Since there can be a lot of conditions that need to be met to be able to access the data, ABAC allows for very fine-grained and secure AC. This high number of conditions is also its biggest limitation, since a higher number of rules means that the AC policy will take up more space. The proposed system chooses high security over being more efficient with space. Another limitation is that this project does not only use BC, but also needs edge nodes in order to work.

Ancile [3] is an Ethereum-based [7] permissioned BC, designed for the management of MD. It utilizes smart contracts in order to achieve heightened AC for users of various roles, like patients, providers and third parties. The ACT that Ancile uses is owner-based, which is better known as DAC. Dagher et al. [3] commit to the idea that patients own their data and that this data is not a currency that can be exchanged. A limitation that Ancile has is that it uses access levels, which, unless specifically specified as an access level, makes it impossible to only retrieve parts of the data. When retrieving data, it always returns the full file. In addition, the authors of the paper do not give experimental results and only give a comparative analysis against other projects [5]. Lastly, it is only possible to grant permission per user, instead of a group of users at once.

MedChain [5] is another project, that is very similar to Ancile. The smart contract structure that is used for AC is almost identical. Because of this, MedChain has the same strengths and limitations as Ancile when it comes to AC.

MedRec [1] is a project that uses an ACT that is a hybrid between DAC and ReBAC. This means that the data that needs to be accessed on the BC is stored in a smart contract that represents the relationship between a patient and a provider. However, the permissions are still only managed by the owner of the data. This smart contract stores a query link to the patients data and a permission query link, which will only display the data that allowed to be accessed. This makes it possible to only retrieve parts of the patient's data that is allowed to be seen, instead of always returning the full file. Since these query links can get complicated, storing it takes up more space than specifying an access level [23]. Just like Ancile, another limitation of MedRec is its incapability to specify permission for a group, rather than an individual. This is the case, because the permissions are stored per relationship. The authors of the paper do not explicitly explain any AC policies that allow third-parties to access the MD. Currently, MedRec is still a prototype.

Shah et al. [23] propose an Ethereum-based MDSS with integrated AC. It builds upon and improves the previously mentioned project, MedRec [1]. The main improvement regarding AC is that it is capable of specifying permissions for groups, which the patient can specify themselves. Since it builds upon MedRec, it has the same limitations regarding AC, besides not being able to specify permissions for groups.

MediChain [22] is a project that uses a HyperLedger Fabric BC in order to create a MDSS with AC. The ACT it uses is DAC, because the authors believe that the owner should have distcretionary authority over the data they own. MediChain is made with a patient-centric user experience in mind. The AC policies are stored in HyperLedger Fabric's ACL. The security and privacy of MediChain has not been tested yet. The authors of the paper state that, in the future, they would like to test MediChain against various security attacks, which implies that this has not been done yet.

Ramani et al. [21] propose a system that allows a patient to fully control the access policies for their data. It is, however, unclear if the authors of the paper use DAC or a combination of RBAC. These access policies are enforced by using smart contracts, which are stored on the BC. The project does not allow patients to give access to other patients, in order to have them edit their access policies. Patients can only receive read-access from other patients. When an unregistered person tries to access the data, access will automatically be denied. This is done by checking the registrar contract, which contains information on all registered users. This does create problems, since this makes it impossible for third-parties to access the data.

EACMS [20] is a HyperLedger Fabric-based BC that does not only integrate AC into a BC-based MDSS, but, through the combined use of RBAC and DAC, it also makes it possible for emergency doctors to receive access to a patient's data in case of an emergency. It does this by allowing an emergency doctor to access the data for a set amount of time when a case is deemed to be an emergency. A big problem that arises with this project is to discern emergencies from non-emergencies. This is an ethical question that does not have an explicit answer, and is therefore out of the scope of this research project. Even though this is the case, it still can not be ignored as a limitation.

Yang et al. [25] propose a system where the AC is only handled by two smart contracts. The Summary Contract and the Record Relationship Contract. Multiple ACTs are used in order to restrict access. The system uses a hybrid between MAC, RBAC and ReBAC. The AC policies are only managed by the data provider, but since this can get complicated, a permissions field is present in the Record Relationship Contract, where permissions for roles can be specified. The permissions field is not used, but present in case it is needed. The other permissions are stored in the summary contract by multiple access rights fields.

Finally, there is MedBloc [14], which is very similar to the other projects. It uses ACLs to let the patient specify who is allowed to access their data. MedBloc uses the DAC technique, which makes their pros and cons mostly the same as other systems that use DAC. The authors of the paper conducted experiments, that show that when more access policies are used, the system does not perform worse than normal. This experiment was not perfect, since they used arbitrary policies, which will not be used in real life applications.

Some limitations that are shared among many different projects are because of ACTs. An important limitation for DAC, for example, is that it is weak against Trojan horse attacks and buggy programs with malicious input [17]. DAC assumes that all programs are benign and correct, which is not the case in actuality. MAC is also not perfect, since all permissions are managed by one administrator. For larger systems, this can cause them to have more work than they can handle, which makes them more prone to mistakes. For ABAC the biggest limitation was already mentioned before: A file can have a large amount of rules that need to be satisfied in order to access the data. This large amount of rules needs to be stored, so a larger amount of space is needed in order to have ABAC. The needed amount of space may outnumber the available space, if the number of rules becomes too high and/or the rules become too complex. RBAC has some limitations as well. Permissions can, for example, only be given to roles. It is impossible to give permission to a user or even an operation. This makes RBAC less flexible than some other ACTs. ReBAC's biggest limitation is that it is impossible to grant access to users that one has not yet been in contact with, since they need to already have established a relationship. When using ReBAC, emergency doctors, ambulance staff, third-parties and similar users, are not allowed to access any data.

3 Methodology

3.1 Review of techniques

In order to create a new or improved ACT for the given healthcare system, the existing ACS's will first have to be reviewed, based on a chosen set of metrics that best fit the requirements of a MDSS.

Metrics

The set of metrics that will be used in order to review the ACS's will be a subset from the set of metrics that is given by Hu et al. [13]. Hu et al. give a set of metrics that can

Project	Framework	Techniques	Advantages	Limitations
[12]	HyperLedger	ABAC	Very fine-grained and secure AC	Inefficient use of space, needs edge nodes
Ancile	Ethereum	DAC	Patient owns their own data	Always gives full health record, no experimental results, no groups
MedChain	Ethereum	DAC	Same advantages as Ancile	Same disadvantages as Ancile
MedRec	Ethereum	DAC + ReBAC	Allows giving access to part of health record	Query takes up space, no access for groups, prototype
[23]	Ethereum	DAC + ReBAC	Improves on MedRec, group access now possible	Same disadvantages as MedRec
MediChain	HyperLedger	DAC	Patient-centric	Not well tested against security attacks
[21]	Ethereum	DAC + RBAC	Only registered users can get access	Third party access impossible and only read access for other patients
EACMS	HyperLedger	DAC + RBAC	Emergency doctors have access in emergencies	Unclear when a case is an emergency
[25]	Ethereum	MAC + RBAC + ReBAC	Access managed by one administrator	Administrator can get overwhelmed
MedBloc	HyperLedger	DAC	Patient manages their own permissions	Not well tested

Table 1: Related work

be used in order to evaluate ACS's, but they do not give any indication of which metrics can be used in order to evaluate an ACS that is used for MD sharing. This is why only a subset of these metrics are useful for the evaluation of such a system. After thoroughly reading and studying the given metrics, we have chosen a set of questions that will be used for evaluation. Each metric can be described by their own set of questions. These metrics and the corresponding questions are as follows.

Auditing: Having a log is significant for the ACS, because it allows administrators to check if no-one is using their access at times that they should not be accessing the data. It makes it possible to manually check if everyone is obeying AC rules that can not be specified digitally.

- **Q1:** Are system failures logged by the ACS?
- **Q2:** Are denied access requests logged by the ACS?
- **Q3:** Are granted access requests logged by the ACS?

Ease of privilege assignments: The amount of steps needed to assign a permission, group or permission inheritance need to be as small as possible for the system to have the optimal efficiency. Therefore, this metric shows the efficiency of projects, based on the amount of steps needed for these actions. The smaller, the more efficient.

- How many steps are needed in order to:
  - **Q1:** Add a permission?
  - **Q2:** Remove a permission?
  - **Q3:** Change a permission?
  - **Q4:** Create a user group and their relations?
  - **Q5:** Create a data group and their relations?
  - **Q6:** Add a permission with inheritance?

Policy management: It is important for a user to be able to manage their policies well. This management is made more fine-grained and flexible when a user can assign a policy to a specific target, like all doctors of a certain hospital. The expiration date allows for users to assign policies that delete themselves at the time they specify. This way a user does not have to delete the policy manually, if it is known until when it is needed beforehand. This is significant, since people can be forgetful.

- **Q1:** Is policy expiration assignment allowed by the ACS?
- **Q2:** Is policy target assignment allowed by the ACS?

Delegation of administrative capabilities: Administrating permissions can be a lot of work if it is done by only one administrator. If the workload becomes too much, the administrator will be more prone to mistakes. This can be solved by allowing the delegation of administrative capabilities.

- **Q1:** Does the ACS allow policy administration delegation?

Bypass: There are a lot of situations in healthcare that require immediate access to a patient's health records. There might not always be someone to provide access at a time like that. Even though the question, of when to use a bypass, is hard and ethical, a bypass, with tolerable risk, is important.

- **Q1:** Is the ACS able to bypass AC rules for critical decisions?
- **Q2:** Is the risk of bypassing AC rules tolerable?

Least privilege principle support: When working in the medical field, one should not get access to more data than necessary. For example, an eye doctor needs access to data regarding your eyes and should for that reason not have access to more than that.

- **Q1:** Is the ACS able to enforce the least privilege principle?

Safety: An ACS is always at risk of failing. This risk may be incredibly small, but it is always important to have some extra safety constraints just in case. The more extra safety constraints there are, the better.

- **Q1:** How many extra safety constraints does the ACS posses?

Conflict resolution or prevention: In some ACTs, it is possible to create multiple permissions that are in conflict with eachother. For example, a user can deny access of all doctors, but allow access for a specific doctor. These conflicts should either be prevented, or resolved.

- Is the ACS capable of:
  - **Q1:** resolving conflicts between policy rules?
  - **Q2:** preventing conflicts between policy rules?

Operational/situational awareness: There are many special cases regarding ACS's for MD. For example, when a patient is under eighteen, their parents should have access until they turn the right age. But in some cases the patient can be unable to manage their own access because of medical reasons. In this case, parents should remain in control. The system needs to be aware of these situations.

- **Q1:** Is the system capable of specifying and enforcing operational/situational awareness control?

Expression properties: ACS's are more easily maintained when they allow for the use of standards and rule specification languages. The same holds for the use of one or more ACTs. This metric is significant for the developers of the system.

- **Q1:** Does the system allow the use of existing AC standards?
- **Q2:** Does the system allow the use of existing AC rule specification languages?
- **Q3:** Does the system follow an existing ACT?
- **Q4:** Does the system allow the combination of different ACTs?

Adaptability to the implementation and evolution of ACTs: In a hospital, ACTs can change and evolve fast. It might, for example, be useful to first grant all doctors access, but later only a specific one. This is why the system should be adaptable.

- **Q1:** Is the system capable of handling future changes in the ACTs?

Evaluation

The ten projects, stated in section 2.4, will be evaluated, based on the questions that were previously chosen. Each of these questions will be answered for all of the projects and afterwards these answers will be discussed in order to find out how an ACS can be made that satisfies the largest subset of questions. To satisfy ease of privilege assignment and safety, the number needs to be better than or equal to the average.

The following tables each represent a question and their corresponding answers for each project. The abbreviations that are used are: No (N), yes (Y), unspecified (UNS), not applicable (N/A) and question x (Qx) where x is a number.

Auditing: A BC, by nature, is a log of all the transactions that have been made. However, this log does not contain log entries regarding system failures, granted requests and denied requests. Table 2 shows the presence of these features.

Project	Q1	Q2	Q3
[12]	N	Y	Y
Ancile	N	N	N
MedChain	N	Y	Y
MedRec	N	N	N
[23]	N	N	N
MediChain	N	N	N
[21]	N	N	N
EACMS	N	N	N
[25]	N	N	N
MedBloc	N	N	N

Table 2: Auditing

Ease of privilege assignment: The amount of steps that need to be taken in order to add, change and remove permissions should not be too big, since this would lessen the performance of the BC. The same holds for the amount of steps needed in order to create user and data groups and to create permissions with inheritance. Table 3 shows the amount of steps needed for each.

Project	Q1	Q2	Q3	Q4	Q5	Q6
[12]	2	3	3	2	N/A	N/A
Ancile	3	4	4	N/A	N/A	N/A
MedChain	3	4	4	1	N/A	N/A
MedRec	2	2	2	N/A	N/A	N/A
[23]	2	2	2	2	N/A	N/A
MediChain	2	3	3	N/A	N/A	N/A
[21]	3	4	5	N/A	N/A	N/A
EACMS	2	3	UNS	UNS	N/A	N/A
[25]	3	4	4	N/A	N/A	N/A
MedBloc	4	UNS	UNS	N/A	N/A	N/A

Table 3: Ease of privilege assignment

Policy management: It may be preferable for a patient to only give permission for a certain amount of time or to doctors, but only of a certain hospital. Table 4 shows if these features are allowed.

Project	Q1	Q2
[12]	N	Y
Ancile	N	N
MedChain	Y	N
MedRec	N	N
[23]	N	Y
MediChain	N	N
[21]	Y	N
EACMS	Y	N
[25]	N	N
MedBloc	N	N

Table 4: Policy management

Delegation of administrative capabilities: Since the administration of permissions can get quite exhausting, it might be preferable to be able to delegate the administrative capabilities to another user. Table 5 shows if this is possible.

Project	Q1
[12]	N
Ancile	Y
MedChain	Y
MedRec	N
[23]	N
MediChain	UNS
[21]	N
EACMS	UNS
[25]	N
MedBloc	N

Table 5: Delegation of administrative capabilities

Bypass: In case of an emergency or other similar situations where critical AC decisions need to be made, it is useful to be able to bypass the AC rules when the risk is tolerable. This is shown in table 6

Project	Q1	Q2
[12]	N	N/A
Ancile	N	N/A
MedChain	N	N/A
MedRec	N	N/A
[23]	N	N/A
MediChain	N	N/A
[21]	N	N/A
EACMS	Y	Y
[25]	N	N/A
MedBloc	N	N/A

Table 6: Bypass

Least privilege principle support: A user should have the minimal amount of access that is possible, while still being able to do their jobs. Table 7 shows if this is the case.

Project	Q1
[12]	N
Ancile	N
MedChain	N
MedRec	N
[23]	N
MediChain	N
[21]	N
EACMS	N
[25]	N
MedBloc	N

Table 7: Least privilege principle support

Safety: In order to avoid leaks, a system can have safety constraints besides the ACS. Table 8 shows the amount of constraints.

Project	Q1
[12]	2
Ancile	1
MedChain	2
MedRec	0
[23]	0
MediChain	0
[21]	1
EACMS	1
[25]	1
MedBloc	1

Table 8: Safety

Conflict resolution or prevention: In some cases, conflicting permissions can be added. These conflicts should be prevented and/or resolved. Table 9 shows if this is the case.

Project	Q1	Q2
[12]	N	N
Ancile	N	N
MedChain	N	N
MedRec	N	N
[23]	N	N
MediChain	N	N
[21]	N	N
EACMS	N	N
[25]	N	N
MedBloc	N	N

Table 9: Conflict resolution or prevention

Operational/situational awareness: An ACS should be able to make AC decisions, based on operational/situational variables. Table 10 shows if this is the case.

Project	Q1
[12]	N
Ancile	Y
MedChain	Y
MedRec	N
[23]	N
MediChain	N
[21]	Y
EACMS	Y
[25]	N
MedBloc	N

Table 10: Operational/situational awareness

Expression properties: There are many useful standards, languages and techniques for AC. Table 11 shows the presence of these properties.

Project	Q1	Q2	Q3	Q4
[12]	Y	Y	Y	Y
Ancile	N	N	Y	N
MedChain	N	N	Y	N
MedRec	N	N	Y	N
[23]	N	N	Y	Y
MediChain	Y	Y	Y	N
[21]	N	N	Y	N
EACMS	N	N	Y	Y
[25]	N	N	Y	Y
MedBloc	N	N	Y	Y

Table 11: Expression properties

Adaptability to the implementation and evolution of ACTs: It is preferable for an ACS to be adaptable to future changes in the ACTs, since these tend to change relatively often. Table 12 shows if this is the case.

Project	Q1
[12]	N
Ancile	Y
MedChain	Y
MedRec	N
[23]	Y
MediChain	N
[21]	N
EACMS	N
[25]	Y
MedBloc	N

Table 12: Adaptability to the implementation and evolution of ACTs

Amount of satisfied questions: Table 13 shows the amount of satisfied metric questions per related work.

Project	Amount
[12]	12
Ancile	5
MedChain	9
MedRec	4
[23]	8
MediChain	6
[21]	4
EACMS	9
[25]	4
MedBloc	3
Average	6.4

Table 13: Amount of satisfied questions

3.2 System Design and Implementation

Now that all metrics have been used to evaluate the related works, a new ACS can be created that satisfies the largest possible subset of questions by looking at how the other works achieve or fail to achieve this.

Access levels

The users of the system can perform various actions on a patient's data. These actions are restricted through the use of access levels. These access levels have a hierarchy and all allow a set of actions. The access levels are as follows.

READ: When a user is granted this access level by a data owner, they are allowed to read the health records of this patient. Any other actions are prohibited.

WRITE: A user that is granted this access level can perform the same actions that a user with the READ access level can perform and more. Besides reading the health records, this user is also allowed to edit the health records.

OWNER: The top of the access level hierarchy is the OWNER access level. A user with this access level has access to all actions. This includes the actions of both the READ and WRITE access level. Besides these actions, this user is also allowed to change the access level of other users, change the permission query and give access to entirely new users.

Permission query

This system uses permission queries, instead of using normal queries that retrieve a patient's full health record. This permission query only returns the parts of the data that a user is allowed to access. A permission query can, for example, only retrieve a patient's name, email and address, instead of retrieving the full file with more sensitive information. When a user has the OWNER access level, they can edit the corresponding permission query, so it retrieves a different part of

the health record. The permission queries are encrypted, to ensure that they can be shared securely.

Transactions

Each user is able to perform a predefined set of transactions. These transactions are as follows.

requestAccess: A user of the system is able to request access to the data. This does not mean that this access request will be granted. The user provides a requested access level, role and institution, so that the system can check if the user has access, based on these attributes. This request can be granted or denied.

updateAccessLevel: Users can try to update the access level that a user has given to another user. If the access level of the updating user is OWNER, this user will be allowed to change the target access level. If the access level is lower, they will not be able to perform this action.

updatePermissionQuery: Users can also try to update their permission query. This permission query holds a query for a part of the health record, for which the data owner has given someone access. This update of the permission query can, just like an update of the access level, only be successfully done when the updater has an access level of OWNER.

grantAccess: users with the OWNER access level is able to grant access to a specific user. They will have to provide an access level and permission query for the target user or group of users.

revokeAccess: Finally, users with the OWNER access level can revoke access. They can remove any permissions, provided they have the appropriate access level.

World state

HyperLedger Fabric consists of two parts that are closely related, but very distinct. These two parts are the blockchain and a so called world state [15]. The world state is a database that holds the current values of a set of ledger states. This makes it possible to easily access the current value of a state, instead of having to traverse the entire transaction log, in order to calculate it. The ledger states are expressed as key-value pairs, whose structure can be {Key=K, Value=V} version=x or {Key=K, Value={KV}} version=x. In this case x is a number, K is a key, V is a value and {KV} is a key-value pair. The values that the ACS uses are stored in this world state in the form of these state ledgers. The state ledgers that are used for the ACS are structured as follows.

Permission: {*Key=K, Value={Level: level, Query: query, expires: bool, (expiration: date)}*}
Access level: {*Key=level, Value=rank*}
Data: {*Key=patient_id, Value=ptr*}

K represents a key value that can have three different values. The key value is of a concatenation that always first includes the patient's ID. Following straight after is the data requester ID or the ID of a role that a user can have. In case of a role, an institution ID can follow. Level is an access level and query is the permission query. Expires is a bool that shows if an expiration date is present. Expiration is the expiration date and is optional. For the access level state ledger, level represents an access level and rank represents a number that signifies the hierarchy of the access level. For the data state ledger maps a patient id to a pointer that points to their health record.

Smart Contract

The transactions used by the ACS are defined within smart contracts. In hyperledger, these smart contracts are stored in something that is called, a chaincode. For this ACS, only one chaincode with one smart contract is needed. This smart contract is called the Access Contract.

The variables used in the smart contract are id, roleId, instId, targetId, patientId, level, query and date. The five id's are used as a key to get values from the world state. id represents a patient-user permission, roleId represents a patient-role permission and instId represents a patient-role-institution permission. TargetId represents the id that one wants to add as key in the world state or the key of the ledger one wants to delete from the world state. PatientId represents the id of a patient. level and query represent the access level and permission query. The date represents the current date and is used to check if a permission is expired. Audit(granted) and audit(denied) represent the part of the contract where granted or denied access requests will be logged. The get, put and delete methods represent actions, performed on the world state.

Access Contract

```
input: id, roleId, instId, level, date, patientId
requestAccess:
    permission = get(id);

    if(permission == null):
        permission = get(instId);

    if(permission == null):
        permission = get(roleId);

    if(permission != null):
        if(!permission.expires || permission.expiration > date):
            reqAccess = get(level);
            access = get(permission.Level);

            if(reqAccess >= access):
               audit(granted);
               return {
                        Pointer: get(patientId).ptr,
                        Query: permission.Query
                       };
            else:
               audit(denied)

        else if(permission.expires):
            audit(access denied);
            delete(permission);
    else:
        audit(denied)


input: id, roleId, instId, query, date
updateAccessLevel:
    permission = get(id);

    if(permission == null):
        permission = get(instId);

    if(permission == null):
        permission = get(roleId);

    if(permission != null):
        if(!permission.expires || permission.expiration > date):

           if(permission.Level == OWNER):
              permission.Level = level;
              put(permission);

        else if(permission.expires):
```

```
        delete(permission);


input: id, roleId, instId, query, date
updatePermissionQuery:
    permission = get(id);

    if(permission == null):
        permission = get(instId);

    if(permission == null):
        permission = get(roleId);

    if(permission != null):
        if(!permission.expires || permission.expiration > date):

           if(permission.Level == OWNER):
              permission.Query = query;
              put(permission);

        else if(permission.expires):
           delete(permission);


input: id, roleId, instId, targetId,
level, query, date
grantAccess:
    permission = get(id);

    if(permission == null):
        permission = get(instId);

    if(permission == null):
        permission = get(roleId);

    if(permission != null):
        if(!permission.expires || permission.expiration > date):

            if(permission.Level == OWNER && expires):
                put({
                    Key=targetId,
                    Value={
                        Level: level,
                        Query: query,
                        expires: true,
                        expiration: date
                        }
                    });
            else if(permission.Level == OWNER):
                put({
                    Key=targetId,
                    Value={
                        Level: level,
                        Query: query,
                        expires: false
                        }
                    });

        else if(permission.expires):
            delete(permission);


 input: id, roleId, instId, targetId, date
 revokeAccess:
    permission = get(id);
    key = id

    if(permission == null):
        permission = get(instId);
        key = instId

    if(permission == null):
        permission = get(roleId);
        key = roleId

    if(permission != null):
        if(!permission.expires || permission.expiration > date):
           delete(get(targetId));

        else if(permission.expires):
           delete(permission);
```

3.3 System evaluation

Our new ACS is created in such a way that it would satisfy the biggest possible subset of the metrics questions that were chosen in section 3.1. This was done by looking at the related work and learning from their mistakes and successes regarding the chosen metrics.

- **Auditing:** Our system logs granted and denied access requests. This satisfies Q2 and Q3 of the auditing metric. Unfortunately, no way could be found to reliably log system failures through the use of smart contracts. This only leaves Q1 unsatisfied.
- **Ease of privilege assignment:** The amount of steps that are needed in order to add, change and remove permissions, in the same order are 4, 3 and 4. Both adding and removing permissions take more steps than the average amount of steps of the other projects, which is 2 and 3. Changing, however, takes the same amount as the average. Finally, creating a user group and their relations takes 4 steps. This is hard to compare to the related work, since most projects don't support groups. The other questions are not applicable. We have chosen for the larger amount of steps, in order to satisfy more metrics. Q2 and Q4 are satisfied.
- **Policy management:** Both policy expiration assignment and policy target assignment are allowed by the system. Policy expiration assignment is made possible by using an expiration date that gets checked for every transaction. Policy target assignment is made possible through the use of ABAC. This satisfies Q1 and Q2
- **Delegation of administrative capabilities:** Policy administration delegation is made possible by allowing any user to give another user the OWNER access level, as long as they posses that access level as well. This way administrative capabilities can be passed on. Q1 is, therefore, satisfied.
- **Bypass:** Bypassing AC rules for critical decisions is not allowed for the ACS. The risks of allowing users to bypass AC rules in a MDSS were too big, since any risk, even small, is not acceptable when it comes to MD. Another reason why bypassing AC rules is not allowed is that deciding when a decision is critical enough to bypass is an ethical question that is out of the scope of this project.
- **Least privilege principle support:** Through the combined use of access levels and permission queries, the least privilege principle is supported. It is possible to grant users access to a small part of data and only allow them to perform a certain action on it. This satisfies Q1.
- **Safety:** Besides the access levels and permission queries, our ACS has two extra safety constraints. These extra constraints are the expiration date and the log. This is one constraint higher than the average of the related works and therefore satisfies Q1.
- **Conflict resolution or prevention:** Conflict resolution is chosen above conflict prevention, because the ability to grant access to both specific users and groups of users was deemed more significant. Conflicts are resolved by prioritising permissions for a specific user over permissions for a group and prioritising permissions for a role in an institution above permissions for a single role. This satisfies the metric, since an ACS needs to have one of the two options. Only Q1 is satisfied.
- **Operational/situational awareness:** Even when looking at related works that have successfully enforced operational/situational awareness, we could not find a way to do

this for the system by using smart contracts within the time that was given for the research.

- **Expression properties:** Unfortunately, the ACS does not allow the use of existing AC standards and rule specification languages. A way to make this possible was not found, but this might be possible after more research. The ACS follows a combination of ABAC and DAC. Pure RBAC is also possible if desired. This satisfies Q3 and Q4.
- **Adaptability to the implementation and evolution of ACTs:** The ACS can, without a problem, adapt to a change of ACT towards RBAC, MAC and only DAC. This is possible because roles are already used, one user can get the OWNER access levels to all health records and all attributes, besides the id, can be removed. Q1 is, therefore, satisfied.

The amount of satisfied metric questions is 14 out of 24 metric questions. This is a considerably larger number than the average of the related works, which is 6.4 as shown in table 13. It is even 2 more than the highest scoring ACS, which is 12, for Guo et al. [12]. Thus, our ACS is an improved version, according to the chosen metrics.

4 Discussion and Future Research

In the previous section we created a new ACS that satisfies as many questions as possible from the metrics chosen in the same section. Afterwards the system was evaluated by using the same set of questions. Most of the questions were satisfied, except for Q1 of auditing, ease of privilege assignment, bypass, operational/situational awareness and Q1 and Q2 of expression properties. For this reason, there might be a better way to integrate AC into a BC-based MDSS, since satisfying all questions would create an improved version of our ACS. However, in the given time and with our current knowledge, this is the optimal ACS. The ACS enables fine-grained access control by using access levels, permission queries and the default security features that HyperLedger Fabric provides. All values that are needed to create the transactions that the ACS needs are stored in the world state and can be retrieved with a key. To define the transactions, the Access Contract was created. When recreating the ACS, a different researcher might create an ACS with slight differences to our ACS. This does not necessarily mean that it is less optimal, since there can be multiple optimal ACS's.
In the future, more research is recommended on some subjects that have not been researched thoroughly yet, while creating the current ACS. These future research subjects are as follows.

- In the future, more research can be done into ways to restrict access. Currently we use access levels and permission queries, but there might be other techniques that can be combined with the current techniques to make an even better ACS. When more research is done, it might be possible to create an ACS that satisfies more metric questions than the current one.
- The ACS already encrypts the permission query, but this might not be the only part of the ACS that benefits from being encrypted. Moreover, we do not go into depth about the type of encryption that is used. In the future, further research can be done into what parts to encrypt and with what type of encryption. By encrypting the right information in the right way, we can improve the security of the ACS.
- In the system design, it is stated that the ACS has a log that contains granted and denied access requests, but we do not go in depth about how this log should be integrated into the system. In the Access Contract, the places where requests should be audited is shown, but there is no transaction that makes this possible yet. Future research into a transaction or smart contract dedicated to auditing is recommended.
- Since not all metric questions are satisfied with the current ACS, future research can be done into the possibility of satisfying them. If more questions are satisfied, the ACS can be improved, provided that all currently satisfying answers stay the same.

5 Conclusions

The aim of this project is to answer the following question. *How can AC be incorporated into a BC-based MDSS?* To answer this question we proposed an ACS, based on HyperLedger Fabric, after evaluating existing works with the use of current literature on this subject. First, a set of metrics that could be used for the evaluation of an ACS for medical data sharing was chosen. Afterwards, the chosen metrics were used to evaluate existing projects to to discern how the metric questions might be satisfied. Finally, using the information obtained from this evaluation, we created the design of a new ACS. The ACS enables access control by using access levels and permission queries, which are stored in the state ledgers of HyperLedger Fabric's world state. Here they are mapped to a key, which can be a combination between the id's of two users, a user and a role or a user, role and institution. These state ledgers can also contain an expiration date for the permission, but this is not mandatory. The world state also contains a type of state ledger that maps a patient to their data pointer. These values are retrieved from the world state in a smart contract, called the Access Contract. This smart contract defines the transactions that allow users to perform the actions needed for an ACS. The granted and denied access requests are logged, in order to make it possible to manually check that no-one is misusing access. Finally, this ACS was evaluated, which showed that it satisfied 14 of the 24 metric questions. This score is more than double the average for the related works, which was 6.4., and 2 more than the highest score, which was 12. So, to answer the main question, AC can be incorporated into a BC-based MDSS, by utilizing smart contracts to define the needed transactions that use access levels and permission queries to restrict the access of users.

6 Responsible Research

6.1 Ethical aspects

According to [8], the definition of ethics is "pertaining to or dealing with morals or the principles of morality; pertaining to right and wrong in conduct". We have created the ACS according to scientific, ethical principals. There are not many aspects of the research that fit the definition, but there are four

that do. First, in 2.4 we discuss the weaknesses of widely used ACTs. This information could be misused by people with malicious intent, in order to get access to patient's medical records. However, this information can be found by simply doing a google search, but would alone not be enough for the attackers to get access to the data. Next, in section 3.1 we chose the bypass metric to be part of the evaluation metrics. This metric tests if bypassing access rules in case of a critical decision is possible and if the risk is tolerable. To satisfy this metric, an ethical question has to be answered. Namely, when is a decision critical? Especially in a healthcare setting, this question is hard to answer with a justifiably answer and should not be answered by the developers, but by the client instead. Furthermore, the information that needs to be restricted by the ACS has a high level of sensitivity. This makes it significant for the ACS to prevent any data from leaking. It is hard to say with certainty that our ACS is impervious to attacks. The question that arises is: Is it ethical to use this design in practice? The only answer that we're able to give is that it is impossible to say this with certainty for any ACS design, since technology is always changing. When a system that uses our ACS is tested well, the design is useable. Finally, an ACS does not have a conscience. For this reason, it is of significance to have humans manually check if ethical AC decisions are done according to ethical principals. Without this, the ACS might make decision that we, as humans, would not approve of.

6.2 Reproducibility

When following the exact steps that were done to create this specific ACS, the outcome should be the same. However, this does not mean that it is easily reproducible. In section 3.1, the metrics were chosen, based on their importance for the ACS of a MDSS. This importance can never be fully objective, so a different researcher might choose a different set of metrics. The same holds for the final design of the ACS. The smart contract of a different researcher might have the same functionality as our Access Contract, but this does not mean that the structure is exactly the same. Our ACS is not the only optimal design, since there are small differences that would create an equally optimal ACS.

References

[1] Asaph Azaria, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman. Medrec: Using blockchain for medical data access and permission management. In *2016 2nd International Conference on Open and Big Data (OBD)*, pages 25–30, 2016.

[2] Qing-hai Bai and Ying Zheng. Study on the access control model. In *Proceedings of 2011 Cross Strait Quad-Regional Radio Science and Wireless Technology Conference*, volume 1, pages 830–834, 2011.

[3] Gaby Dagher, Jordan Mohler, Matea Milojkovic, and Praneeth Marella. Ancile: Privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology. *Sustainable Cities and Society*, 39, 02 2018.

[4] Ni Dan, Shi Hua-Ji, Chen Yuan, and Guo Jia-Hu. Attribute based access control (abac)-based cross-domain access control in service-oriented architecture (soa). In *2012 International Conference on Computer Science and Service System*, pages 1405–1408, 2012.

[5] Eman-Yasser Daraghmi, Yousef-Awwad Daraghmi, and Shyan-Ming Yuan. Medchain: A design of blockchain-based system for medical records access and permissions management. *IEEE Access*, 7:164595–164613, 2019.

[6] Alevtina Dubovitskaya, Zhigang Xu, Samuel Ryu, Michael Schumacher, and Fusheng Wang. Secure and trustable electronic medical records sharing using blockchain, 2017.

[7] Ethereum. https://ethereum.org/en/.

[8] Ethical. https://www.dictionary.com/browse/ethical.

[9] Yanfang Fan, Zhen Han, Jiqiang Liu, and Yong Zhao. A mandatory access control model with enhanced flexibility. In *2009 International Conference on Multimedia Information Networking and Security*, volume 1, pages 120–124, 2009.

[10] Ruogu Fang, Samira Pouyanfar, Yimin Yang, Shu-Ching Chen, and Sundararaj Iyengar. Computational health informatics in the big data age: A survey. *ACM Computing Surveys*, 49:1–36, 06 2016.

[11] Philip W.L. Fong. Relationship-based access control: Protection model and policy language. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, CODASPY '11, page 191–202, New York, NY, USA, 2011. Association for Computing Machinery.

[12] Hao Guo, Wanxin Li, Mark Nejad, and Chien-Chung Shen. Access control for electronic health records with hybrid blockchain-edge architecture. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 44–51, 2019.

[13] Vincent C. Hu, Karen Scarfone, Vincent C. Hu, Karen Scarfone, and Scarfone Cybersecurity. Guidelines for access control system evaluation metrics, 2012.

[14] Jack Huang, Yuan Wei Qi, Muhammad Rizwan Asghar, Andrew Meads, and Yu-Cheng Tu. Medbloc: A blockchain-based secure ehr system for sharing and accessing medical data. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 594–601, 2019.

[15] Ledger. https://hyperledger-fabric.readthedocs.io/en/release-2.2/ledger/ledger.html.

[16] Kathrin Lehmann and Florian Matthes. Meta model based integration of role-based and discretionary access control using path expressions. In *Seventh IEEE*

International Conference on E-Commerce Technology (CEC'05), pages 443–446, 2005.

[17] Ninghui Li. How to make discretionary access control secure against trojan horses. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–3, 2008.

[18] Xu Ma, Chen Wang, and Laihua Wang. The data sharing scheme based on blockchain. In *Proceedings of the 2nd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, BSCI '20, page 96–105, New York, NY, USA, 2020. Association for Computing Machinery.

[19] Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. Blockchain. *Business and Information Systems Engineering*, 59, 03 2017.

[20] Ahmed Raza Rajput, Qianmu Li, Milad Taleby Ahvanooey, and Isma Masood. Eacms: Emergency access control management system for personal health record based on blockchain. *IEEE Access*, 7:84304–84317, 2019.

[21] Vidhya Ramani, Tanesh Kumar, An Bracken, Madhusanka Liyanage, and Mika Ylianttila. Secure and efficient data accessibility in blockchain based healthcare systems. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 206–212, 2018.

[22] Sara Rouhani, Luke Butterworth, Adam D. Simmons, Darryl G. Humphery, and Ralph Deters. Medichaintm: A secure decentralized medical data asset management system. *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Jul 2018.

[23] Mira Shah, Chao Li, Ming Sheng, Yong Zhang, and Chunxiao Xing. Smarter smart contracts: Efficient consent management in health data sharing. In *APWeb/WAIM*, 2020.

[24] Fatima Sifou, Ali Kartit, and Ahmed Hammouch. Different access control mechanisms for data security in cloud computing. In *Proceedings of the 2017 International Conference on Cloud and Big Data Computing*, ICCBDC 2017, page 40–44, New York, NY, USA, 2017. Association for Computing Machinery.

[25] Guang Yang and Chunlei Li. A design of blockchain-based architecture for the security of electronic health record (ehr) systems. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 261–265, 2018.

[26] Ke-Jun Zhang and Wei Jin. Putting role-based discretionary access control into practice. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, volume 5, pages 2691–2696 vol.5, 2004.