David Steiman, 338864

Design, Development, and Evaluation of a blockchain-based Decentralized Access Control Mechanism for Microservices

Bachelor thesis expose

Version 5

I. Context

In a microservice / federated architecture, there are different demands on security. The focus of today's microservice sources is mostly for the question: how to ensure that at request time all information needed for access decisions is available for processing. In details, the topics cover authentication (e.g., OpenID, IAM, SAML) to identify the requesting subject and authorization (e.g., OAuth 2, JWT) to gather all information about the authorization of the subject. The next question is about, how to make a decision. For this purpose, there are different approaches to access control enforcement, like role-based access control (RBAC) or attribute-based access control (ABAC). These concepts are simple, but lack on expressibility in complex access control requirements. For fine-grained access control, a well-established way is using "access control lists" (ACL) for a detailed definition of policies, which often occur in monolithic applications (and does not consider distributed usage). While tools like JWT make it unnecessary to store a state of a session, ACLs have to be persisted in a way accessible to all microservices. Moreover, someone has to manage the ACL configuration. In particular, in federated systems, where services belong to a certain party, every member has to trust this policy administrator. Consequently, the core idea of ACLs lacks in two points: decentralization and trust.

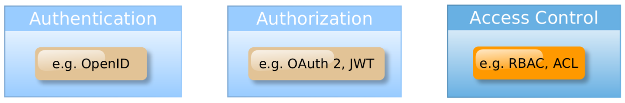


Figure 1. security demands in microservices.

There also exists an approach of using ACLs in distributed systems. This method uses the term "domain" what Yialels summarizes as "grouping of objects in hierarchical structures for management of large systems" [1] based on Slomans definition of domain as "an object which maintains a list of references to objects that have been explicitly grouped together for the purposes of management" [2]. This definition is compatible with Evans definition of a domain as "the realm of knowledge [..] of the software system being designed" [3], what is the definition of "domain" used in microservices. Yialelis' concept, which is close to a system this thesis aims for, is designed with ACLs. An entry of the list contains the requesting subject, its domain, the requested target, and access rules defined for those. A reference monitor enforces policies. Although his idea has a focus on distributed systems, the concept assumes the presence of a policy administrator, which is performing the overall security configuration. So this access control mechanism requires trust in a central authority.

This thesis introduces a "decentralized access control list" (DACL) mechanism. Assuming the presence of an identity management and authentication, DACL describes how to enforce authorization policies, being fully decentral and eliminating the need for a central authority (trustlessness).

In favor of decentralization, DACL requires the underlying persistence be managed in a distributed manner, and that stored data cannot be changed later. For trustlessness, DACL supposes a smart contract, deployed on a blockchain, is coordinating all state-changing operations.

In favor of decentralized governance of microservices, the mechanism itself does not consist of technical details, to leave the implementation open to the service. The mechanism will be applied to an exemplary scenario to show, how these mechanisms should be used and implemented using Ethereum and IPFS.

II. Example scenario

The scenario describes a federated eHealth application platform for diagnoses and prescriptions. During treatment, a doctor declares several diagnoses and prescribes prescriptions for his patients. In this scenario, two parties are maintaining the eHealth platform, which consists of a diagnosis and a prescription platform. As medical data of a patient is highly sensitive, the access control policies are complex. On the one side, the patient should always be able to supervise the permission on his data, but allow exceptions for his doctor, e.g., to open access to diagnoses to collaborating doctors. On the other side, both doctors and

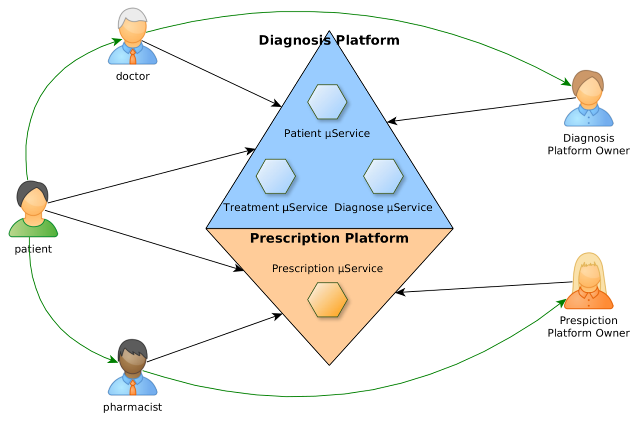


Figure 2. architecture of an eHealth Platform.

pharmacists might become patients as well, so approaches like RBAC do not fit well.

The green arrows in figure 2 show the permission grant direction, while the black arrows show the common requests done by the different users. As data relating to a user should only be under the user's control, maintaining a central policy authority is not acceptable. Using centralized ACL mechanism also means, one node (or a centrally managed group of nodes) is responsible for storing the policies and allows this central authority to bypass the policies and manipulate the data.

The thesis claims to show, how DACL helps to construct a decentralized access control mechanism, to ensure patients keep control over their data by implicitly changing the security policies of the eHealth platform.

III. Problems

A. Trustlessness

The first core problem of DACL is trustlessness. Trustlessness means, no one has special privileges over the state of the policies. In other words: there is no need to trust someone. So the subsequent problems here are:

1) Network members and proposal handling: The DACL smart contract allows a set of participants (e.g., Ethereum wallets) to be part of a network, where every member is permitted to interact with the smart contract. As there is no network administrator, there should be mechanisms how new members can join or how to remove malicious participants.

As there is no central policy authority as well, which can perform changes on its behalf, the mechanism should consist of a policy change handling, where policies are committed after the network reached consensus. If the basic idea says, every participant can propose changes and vote on proposals; then there are subsequent problems of the voting system design. In a naive approach, all participants are forced to vote on every incoming proposal. However, there are use-cases, where a participant is not required to participate in the voting process for some reason. There should be proper conditions when change proposals are accepted.

In the case when multiple policy change proposals are pending the network must end up in a state, where all valid policy change proposals are checked in.

2) Access policy enforcement: Comparing to the reference monitor from Yialelis' domain security model, DACL should define, how policies are enforced. Assuming technical requirements violates the microservice approach of decentralized governance the mechanism should specify, how the policies are applied. Optionally it could be discussed, how other members can prove the policy enforcement.

3) After-the-fact mandate changes: Although only participants known by the smart contract are permitted to interact with the policies, there still are business use-cases to make exceptions on these mandates. To cover similar exceptions on a "consortium blockchain", the Hyperledger requirements work group released a set of 8 requirements [4], where the following are relevant to the DACL design:

- a member(anchor) shall be able to sign over its authorization to another member (guardian) after the fact. This requirement accords to the following two scenarios:
  - anchor loses its authorization
  - anchor maintains its authorization over the smart contract
- the network shall be able (e.g., via a blockchain consortium) to sign over the authorization over a smart contract from anchor to another member
- it should be possible to re-assign the autho-

rization

- all assignments shall not require handling over private keys

As the DACL does not define a consortium, the other requirements do not fit well here.

4) Policy storage: Agnostic to the storage design, the owner of a node should not be able to change the data. Manipulating the value of the stored data should make the data invalid.

B. Decentralization

The second problem of DACL is about reducing the responsibility of a single participant and coordinating all tasks around communication and persistence completely in a distributed manner. In particular:

1) Policy storage: No node should form a single point of failure. As of Brewers CAP theorem, there is no way to guarantee consistency, availability, and partition at the same time. For availability, An access control decision must be possible at any time. There should be a tradeoff between consistency (ensure new policies are always available right after their commit), or partition (avoid storing a complete policy configuration state on every node).

2) Costs of operation: Putting data on the blockchain is expensive, so it must be clarified, which part of the data is "on-chain" and which "off-chain". In particular, how to refer to the off-chain-stored data.

Considering the core component of DACL is an Ethereum smart contract, every writing operation costs some fees in Ether (the Ethereum cryptocurrency) to be paid by a participant. Also, the complexity of the contracts final implementation is relevant for the running costs, as these depend on the number of instructions processed in Ethereum. When a participant runs out of gas, he becomes unable to communicate with the network, so the contract must regulate this costs with methods like auto-refilling and define how the network responds to frozen participants. There are ways of inventing health checks, but these also increase the running costs. For example, forcing every participant to write the current timestamp on the blockchain can prevent freezes, but leads to more cost-intensive write operations.

3) Performance: The performance of DACL can be narrowed down to the response time for a request, which involves an access decision, and the duration for new proposals getting committed and propagated to all participants. The entire state can be stored in a single document, containing the full policy configuration. Consequently, a proposal consists of a whole state, too. Handling these policies unstructured may heavily affect the time a participant needs to iterate over the proposal. In a different approach, a proposal consists only of a set of changes, so the current state is the result of all accepted changes applied after each other. Without caching, accumulating the changes may cause many requests, what affects the response time.

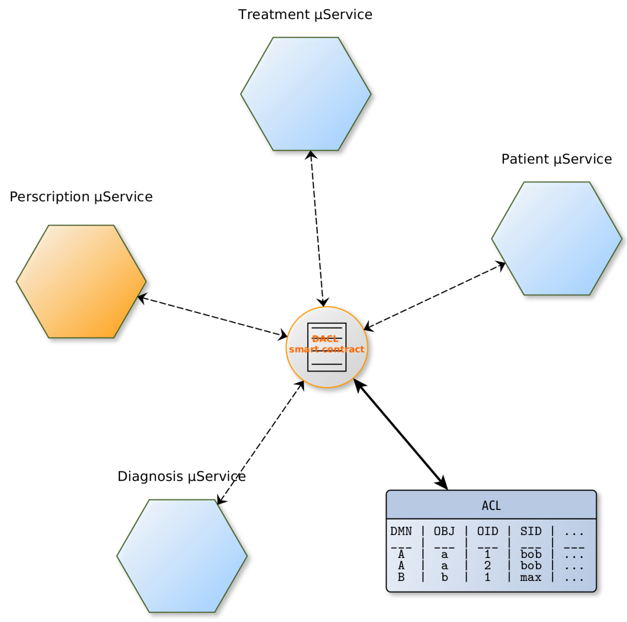
4) Horizontal scalability of the microservices: Microservices are often designed according to the twelve-factor app specification [5]. Processes and Disposability are the key factors for horizontal scalability, as microservices run replicated and might be terminated at any time. Considering DACL is causing a time-intensive overhead for building a valid state, this would harm these two factors and consequently impact the horizontal scaling abilities of DACL secured microservices. A vivid example would be if every microservice must first synchronize with the entire Ethereum blockchain from scratch, what would take hours for a new replica to be running.

IV. Proposed solution approach

The proposed distributed access control list mechanism (DACL) claims to give a solution to the problems. At the core, all members of the DACL defined network interact with a single smart contract, which provides the actual state of the ACL and coordinates changes to this state.

A. DACL core parts

1) Access decision protocol: Building a system in a certain language, which enforces policies directly, would limit the usage of DACL to this language and its ecosystem. Instead, DACL defines an access decision protocol which describes, how a microservice should evaluate the policies, and assumes every microservice enforces them according to that protocol.



DMN	OBJ	OID	SID	...
A	a	1	bob	...
A	a	2	bob	...
B	b	1	max	...

Figure 3. Basic DACL architecture applied to scenario.

The protocol first defines an access control entity $ace \in \mathbb{ACL}$, which is in an access control list $\mathbb{ACL}$, defined by: $\mathbb{ACL} = \{(domain, oid, uid, sid, permission, order) : domain, oid, uid, sid \in \Sigma^*, permission \in \{0,1\}^4, order \in \mathbb{N}\}$ with:

$domain$ = the domain that owns the object

oid = the object identifier of target object

uid = the objects unique identifier

sid = the unique identifier of the requesting subject

$permission$ = 4 bits for create, read, update and delete permission

$order$ = the order of this entry in a set of matched rows

This infinite discrete set can be represented as the following table:

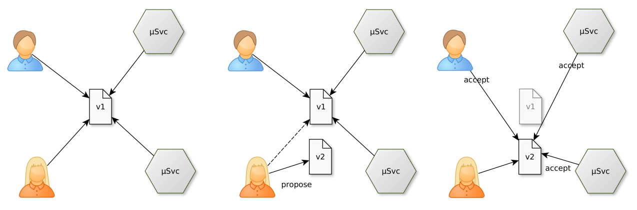
domain	OID	UID	SID	permission	order
...	...	...	...	...	...
diagnoses	diagnosis	123	Alan	0111	1
diagnoses	diagnosis	123	Dr. Bob	0111	2
diagnoses	diagnosis	123	@any	0000	3
prescriptions	prescription.	234	Alan	0100	1
prescriptions	prescription	234	Pharmacist	0100	2
prescriptions	prescription	234	Dr. Bob	0111	3
prescriptions	prescription	234	Dr. Colin	0111	4
...	...	...	...	...	...

Every microservice acts as a participant for a domain (by owning a private key known to the smart contract).

To evaluate this ACL for a given object and subject, a participant iterates over all access control entries for its matching domain in the defined order. The first match is providing the decision if access is granted. A participant is free to use any convention for identifying a single subject, a group of users or a role.

2) Coordination mechanisms: Both, for member and policy administration DACL defines a coordination mechanism, which consists of proposals and voting procedures.

By example, the policy coordination mechanism describes how exactly parties can change the ACL. Given the fact, all communication is done via the DACL smart contract, figure 4 describes how a successful reconciliation should proceed.



(a) initial state (b) state change proposal (c) accepting new policy

Figure 4. example of successful change proposal

Every microservice can read the ACL by interacting with the smart contract. DACL assumes the microservice to implement the access decision protocol. However, interpreting non-owning domain object policies is optional. When a participant wants to change policy rules, he defines a new version of the ACL and reports a proposal to the network. Each participant can either accept the new version, deny with a user-defined reason or abstain from answering. If no of the participants denies the proposal, the smart contract provides the new ACL list.

A similar coordination mechanism will be designed for network member joining/kicking, handling voting conditions, and mandate changes.

B. Implementation and on-chain/off-chain design

After defining how that mechanism should work in theory, a reference prototype must be implemented using a common language for building microservices. The example will be implemented using NodeJS and Solidity (for developing smart contracts).

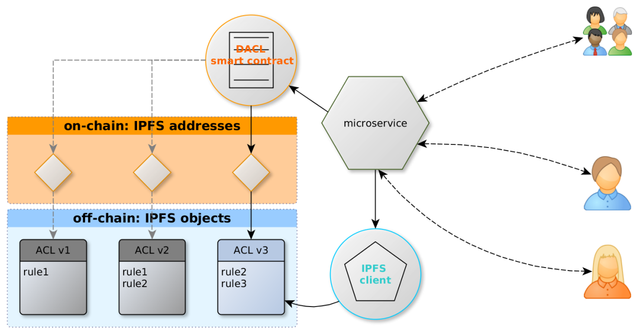


Figure 5. DACL architecture

The smart contract will store IPFS addresses of objects, to keep as little as possible data on the blockchain. In the most basic scenario, every microservice handles coordination directly by interacting with IPFS and Ethereum.

To standardize the coordination, a DACL client can be developed that runs parallel to each microservice and makes the DACL processes available via a well-defined API. It interacts with the smart contract, reads the address from the blockchain and opens the content using an IPFS client.

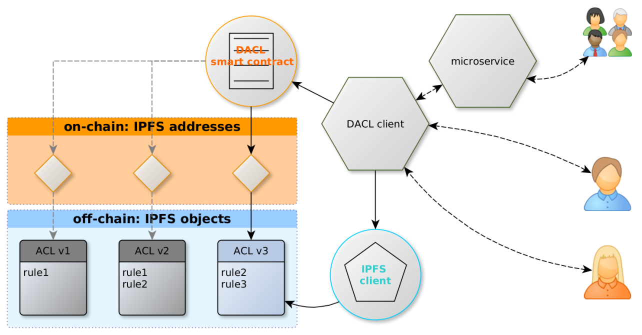


Figure 6. DACL architecture with client

V. Evaluation

After designing the mechanism and implementing a prototype, it is possible to analyze the result regarding trustlessness and decentralization.

A. *Security evaluation*

As trustlessness was defined as the absence of central authorities, the only way of control is the consensus. The voting system offers a strong protection from malicious actions coming from a minority of a network, but several processes take some time, as kicking members or changing policies. So the security evaluation covers the overall security of DACL. In details, it covers how good the privacy promise is satisfied for a patient. In other words: Can a patient trust the platform, which is secured by DACL? Back to the protection claim, the first tests will be about majority attacks. With the permission to add new members to the network, one party could add a greater amount of voters to the network to manipulate voting decisions during proposals. The attack aims to change a patient's access control policy. The next tests will cover ways of performing a minority attacks, as blocking proposals or changing the local values of known policy configurations. Further analysis will try to find exploits, which might be possible during reconciliation if there are some. Assuming the network identifies a malicious participant, one evaluation will analyze the time of excluding the member from the network. Another point of interest is how the system performs in use cases, where mandate changes are required. For example, if a patient loses his private keys, but needs access to the data. Furthermore, the patient can grant a doctor to re-grant his permissions to different parties. Considering one of those parties loses control of his keys, the question is how the patient can revoke those permissions, and how fast he can do that.

B. *Performance evaluation*

The performance evaluation analyzes, how much the mechanism affects the initial performance of the application. The basic design of DACL is optimized for fast reading operations, at the expense of the time for changes being committed. The access decision protocol assumes, there is a version of the policy configuration ready for analyzing, and defines how to make a decision. Nevertheless, the performance of analyzing this state might deteriorate with a growing amount of data. For this purpose, a test case with an enormous amount of data with corresponding ACLs will be performed. In a similar scenario, the next tests will analyze the performance of handling new policy proposals. In details, the eHealth platform owners may decide, that an entry of data is available, as soon its ACLs are available for processing.

Here, a simulation will test a high frequency of new proposals and the target attribute is the time of data being available.

VI. Timeline

- formally design the DACL mechanism
- (possibly) starting implementation of DACL client
- Building a prototype of a microservice based application, implementing the mechanism and using the DACL policies
- Evaluating the prototype as a proof-of-concept, referring to defined claims and new problems, found during the modeling and/or implementation process

References

[1] N. Yialelis, *Domain-Based Security forDistributed Object Systems*, 1996.

[2] M. Sloman, “Journal of network and systems management.”

[3] E. Evans, *Domain Driven Design*, 2002.

[4] M. O. van Deventer and R. Joosten, “Hyperledge - after-the-fact mandate changes.”

[5] The twelve-factor app. [Online]. Available: https://12factor.net/

More papers and references are pending.

CONTENTS