Matrix Decomposition: Analysis of an Access Control Approach on Transaction-based DAGs without Finality

Florian Jacob
Karlsruhe Institute of Technology
Institute of Telematics
florian.jacob@kit.edu

Luca Becker
Karlsruhe Institute of Technology
Institute of Telematics
luca.becker@sunbury.xyz

Jan Grashöfer
Karlsruhe Institute of Technology
Institute of Telematics
jan.grashoefer@kit.edu

Hannes Hartenstein
Karlsruhe Institute of Technology
Institute of Telematics
hannes.hartenstein@kit.edu

ABSTRACT

The Matrix message-oriented middleware[1] is gaining momentum as a basis for a decentralized, secure messaging system as shown, for example, by its deployment within the French government and by the Mozilla foundation. Thus, understanding the corresponding access control approach is important. This paper provides an abstraction and an analysis of the access control approach followed by Matrix. We show that Matrix can be seen as a form of Distributed Ledger Technology (DLT) based on Transaction-based Directed Acyclic Graphs (TDAGs). TDAGs connect individual transactions to form a DAG, instead of collecting transactions in blocks as in blockchains. These TDAGs only provide causal order, eventual consistency, and no finality. However, unlike conventional DLTs, Matrix does not aim for a strict system-wide consensus. Thus, there is also no guarantee for a strict consensus on access rights. By decomposition of the Matrix approach, we show that a sound decentralized access control can be implemented for TDAGs in general, and for Matrix in particular, despite those weak guarantees. In addition, we discovered security issues in popular implementations and emphasize the need for a formal verification of the employed conflict resolution mechanism.

ACM Reference Format:
Florian Jacob, Luca Becker, Jan Grashöfer, and Hannes Hartenstein. 2020. Matrix Decomposition: Analysis of an Access Control Approach on Transaction-based DAGs without Finality. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies (SACMAT '20), June 10–12, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3381991.3395399

1 INTRODUCTION

The starting point of our analysis is a specification of communication protocols and behavior of a decentralized publish-subscribe middleware with integrated history and state tracking, called Matrix.[1] Currently, Matrix represents the basis for a popular decentralized messaging tool with a higher ambition to interconnect arbitrary platforms for near real-time communication. The use of Matrix by the French government [7] and the Mozilla foundation [10], among others, and the discussion of its use in organizations like the Federal Defence Forces of Germany [8] demonstrates its relevance. Since the Matrix approach is evidently used, or intended to be used, in deployments with strict security requirements, there is a natural interest in understanding and assessing the underlying access control approach. However, the Matrix approach is an unconventional one, resembling elements of distributed ledger technologies and blockchains. Thus, an analysis of the access control system requires an analysis of the decentralized approach itself. In this paper, we address the following three questions:

(1) What type of decentralized system is given by Matrix?
(2) Are the foundations of such a system sufficient to provide a valid access control approach?
(3) What access-control-related aspects need to be addressed before a 'mission-critical' deployment can be recommended?

In this paper, these aspects are addressed by 'decomposing' the Matrix approach. The first question will be answered by showing that the Matrix approach to decentralization can be seen as a variant of a distributed ledger, using a Transaction-based Directed Acyclic Graph (TDAG) without finality as transaction store. This form of abstraction does not only generalize results, but is essential for understanding the system. While the notion of a Transaction-based Directed Acyclic Graph without finality is clarified in the following, it basically translates into rather weak guarantees upon which an access control system is built. Surprisingly, as answer to the second question, the decomposition shows that those weak guarantees seem to be indeed sufficient to build a sound access control system. However, in answering question three, we show that some further actions are required.

Decentralized trust management has been a topic of research for quite some time, see e.g. [2]. Decentralized access control for publish-subscribe systems has been proposed, e.g., in [17]. However, previous work has either not analyzed access control approaches implemented on top of TDAGs or similar distributed ledgers, or finality is assumed, like in the IOTA Tangle [18]. Our contribution is the analysis of a TDAG-based access control approach without finality. In this context, we solely focus on policy specification, policy information, and policy enforcement.

Distributed ledger technologies implement distributed, highly-available, append-only databases. In their earliest form, transactions to the database are collected into blocks and linked together to form a linear chain, the so-called blockchain concept. In contrast, the Transaction-based Directed Acyclic Graph concept links transactions together to form a Directed Acyclic Graph instead of a linear

[1]See https://matrix.org/. Please note that we use the term "middleware", commonly used in distributed systems, now for decentralized systems.

SACMAT '20, June 10–12, 2020, Barcelona, Spain
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7568-9/20/06…$15.00
https://doi.org/10.1145/3381991.3395399

chain, which reduces the inherent ordering from a total to a partial order [15]. Distributed ledger technologies can be characterized by their trade-off decision between decentralization, consistency, and scalability, also known as the DCS triangle[2] [23]. Similar to the CAP theorem [4, 3], the DCS triangle's yet to be proven conjecture is that only two out of three properties can be fully achieved simultaneously, and that there is a gradual trade-off between those properties. The system under consideration, Matrix, trades consensus on a total ordering of conflicting transactions and on which transactions can be considered final (i.e. finality), for a high degree of decentralization and scalability. Without the need to solve a consensus problem, there is also no need for, e.g., proof of work or similar mechanisms as used in the Bitcoin blockchain. However, as a consequence there is no system-wide consensus on access control.

This paper is structured as follows. In Section 2, we classify what type of decentralized system the Matrix middleware is. Furthermore, we explain and classify the fundamental data structure used by Matrix as a TDAG. We elaborate on the challenges associated with access control based on the weak guarantees given by TDAGs as used by Matrix. We also define the requirements that a decentralized access control system based on partial order has to fulfill to be considered secure. Fundamental related work is also provided in Section 2, however, as this paper touches various fields (Matrix specification, access control models, publish-subscribe systems, distributed/decentralized system) we also give references to related work in all of the following sections. In Section 3 we analyze how Matrix deals with partial order and non-finality. In Subsection 3.1, a conceptual model for access control for the given interface and guarantees is described based on Lattice-based Access Control (LBAC) and Attribute-based Access Control (ABAC). As concurrency can give rise to conflicts in causal relationships, we analyze in Subsection 3.2 a conflict resolution mechanism required to provide an attribute store for the access control system. While the previous sections only assumed an implementation of the interface and guarantees, we describe in Subsection 4.1 architecture and mechanisms of a truly decentralized system based on TDAGs. We assess the decentralized access control system implementation proposed by Matrix for its security in Subsection 4.2, describe several security issues and present insights gained from the assessment. Finally, we conclude in Section 5 that decentralized, secure implementations of the analyzed class of access control systems seems possible with the given guarantees, but emphasize the need for formal verification of the conflict resolution mechanism and the need to understand the 'characteristics' of an access control approach based on TDAGs without finality.

2 CONCEPTUAL OVERVIEW & DECOMPOSITION

2.1 System Overview and Terminology

Matrix is a specification[3] for a decentralized publish-subscribe middleware with integrated history and state tracking. Its most commonly used application is a messaging system: Messaging is based on "rooms", which are conversation groups on a theme, consisting of an arbitrary number of users that can join, read, participate in and leave the conversation. Rooms have a history of current and past communication messages as well as a state that is represented by a set of currently valid attributes of the room. A user's participation in a room is modeled in form of a membership relation, which is augmented by attributes as well. Attributes are not only used as "cosmetic" metadata on rooms and memberships, like their name and avatar, but also for moderation and access control in general. Attributes can represent whether a room is a public channel free for anyone to join or a private, invite-only group. Similarly, attributes can specify whether anybody is free to speak in a room or only a subset of users is allowed to send messages. For access control, it is particularly important to note that administrative permissions, i.e. permissions to change policy attributes, are represented as attributes as well.

One user can have multiple devices associated with their account, which are not required to be online all the time in order to receive messages. Instead, users associate themselves with a so-called "homeserver", a server which acts as a representative for them in the Matrix network, and is in charge of relaying user actions from and to the homeservers of other users. Users either use a homeserver provided by a third party or operate their own.

For each room, homeservers of joined users form a federation in which they exchange new messages and attribute changes. Each room is strictly independent of other rooms, i.e. there is no protocol-level interaction between rooms. For access control decisions, only the attributes of the concerned room and memberships are relevant. For the remainder of the paper, we assume the presence of a single room and a single federation only. However, all considerations can be generalized to multiple rooms existing in parallel.

We will use the terminology of publish-subscribe and access control systems instead of Matrix terminology as follows. We consider the union of all devices of a user to be a *subject* of the access control system, which *subscribe* to and *publish* to *topics* instead of sending to and receiving from rooms. While being subscribed to a topic implicitly grants receive permissions, a subject is not required to exercise them, although a subscription is required for publishing to a topic. Subjects publish messages to a topic's history or attribute changes on the topic or a topic's memberships. Messages and attribute changes always have a *type*, which specifies the semantics of the content. Matrix therefore is a topic-based, typed publish-subscribe system.

We can also compare the Matrix middleware to distributed ledgers. More specifically, each topic in Matrix is considered an independent distributed ledger. According to Zhang et al., distributed ledgers can be split up into three major components:

- The append-only data structure that stores transactions
- The peer-to-peer-network that distributes transactions
- The consensus mechanism that provides conflict resolution to conflicting transactions

In this paper, we do not address aspects of the peer-to-peer network as a distribution mechanism (see, e.g., [11] for a monitoring study of the Matrix network). In contrast, this paper addresses the data structure and analyzes the "consensus mechanism" component with respect to its ability to support access control enforcement.

[2]To the best of our knowledge, the idea was first found in https://blog.bigchaindb.com/the-dcs-triangle-5ce0e9e0f1dc

[3]https://matrix.org/docs/spec/

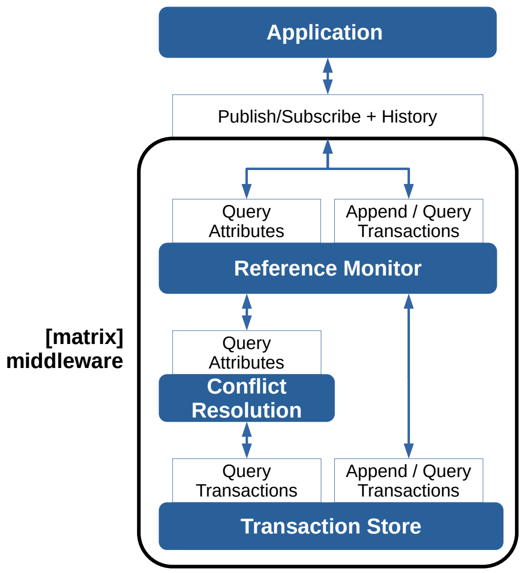

Figure 1: Layer model for the Matrix middleware. Rounded, blue-filled boxes are layers, rectangles are interfaces provided by the lower layer and used by the upper layer. The layers are abstracted from decentralization aspects, i.e. reduced to a single instance.

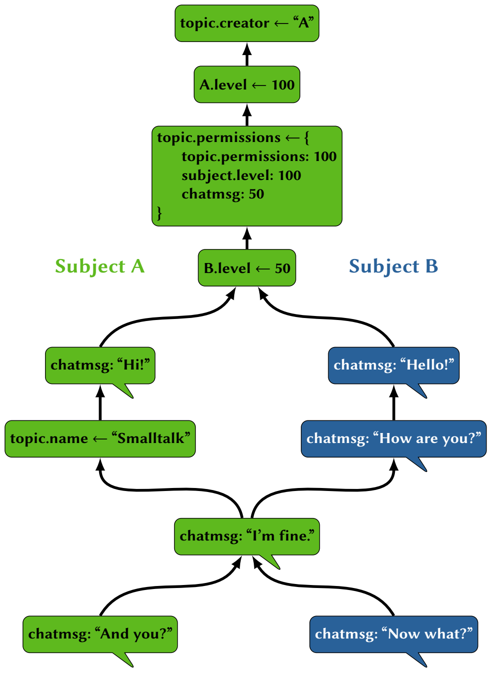

Figure 2: Example of a Matrix Transaction-based Directed Acyclic Graph storing all transactions for a topic. Both message and attribute transactions are stored in the same data structure.

Figure 1 shows the inner working of the Matrix middleware: At its core lies the per-topic Transaction Store, which only allows appending new and querying past transactions. To the application layer, however, Matrix provides a publish-subscribe interface with history access. This is achieved by a translation between the internal and external interface, which maps the publication of a message to appending a *message transaction*, and the publication of an attribute change to appending an *attribute transaction*. The internal transaction-based interface is more powerful, as it allows the specification of a causal relation of a new transaction with respect to other transactions, which is not possible using the publish-subscribe interface. Utilizing all attribute transactions in the Transaction Store, the Conflict Resolution layer resolves conflicts between concurrent or contradicting attribute changes. The Conflict Resolution layer derives a consistent total ordering based on the causal ordering of transactions, and, thus, is able to provide an interface to query the resolved attributes. The attribute interface and the transaction interface together enable provisioning of the publish-subscribe interface with history access. Both interfaces are intercepted by the Reference Monitor layer that is responsible for evaluation and enforcement of access control. The Reference Monitor depends on the conflict-resolved attributes provided by Conflict Resolution as sole policy information source to make an access control decision.

2.2 TDAG-based Ledgers without Finality and their Guarantees

In this section, we classify the Transaction Store used by Matrix as a Transaction-based Directed Acyclic Graph (TDAG) and explicate the (weak) guarantees that it provides: partial (causal) order, eventual consistency, and no finality.

Partial (causal) order. New local transactions can be in a causal relationship to the past transactions currently available to the local replica of the data structure, but not in causal relation to concurrent or past transactions from remote replicas not yet known to the local replica. This potential causal relationship is known as the "happened before" relationship as defined[4] by Lamport [16], which leads to a partial order on all transaction, the so-called *causal order*. New transaction only need to reference existing transactions without descendants due to transitivity: If transaction x happened before y and y happened before z, then x happened before z.

In Matrix, the causal order is established by the homeservers: When a user creates a new transaction and forwards it to their homeserver, the homeserver will append it to all local transactions that are not yet in a "happened before" relation with newer transactions, which keeps the number of references minimal. Using the fact that every partial order can be presented as a directed acyclic graph (DAG), homeservers use a DAG to store all transactions and their causal order. Such a DAG is called a TDAG (c.f. [15]). An example of such a TDAG is given in Fig. 2.

Eventual consistency. Every homeserver with users subscribed to a topic will try to synchronize its full TDAG with every other subscribed server. Due to synchronization not being instantaneous, different servers can append concurrent transactions to their TDAG replica, which are later synchronized and result in parallel transactions that cannot be compared causally. In graph terms, concurrent

[4]Note that Lamport defines x "happened before" y as $x \longrightarrow y$. In this paper, we actually use the converse relation $y \longrightarrow x$, so that new transactions can reference old transactions and the metadata of transactions can be kept immutable. It follows that for $y \longrightarrow x$, we say x is the parent transaction of y.

transactions are a fork in the DAG and create independent causal chains. When synchronization is not possible for an extended period of time, e.g., during a network partition, chains of transactions independently grow and replicas will be in an inconsistent state until synchronization is possible again. New transactions after the synchronization will reference the most recent transaction from both chains, which will lead to the independent chains being merged again. When reading the data structure, it is up to the reader to interpret the independent transactions of different chains. The TDAG as described above, thus, provides eventual consistency, which means that replicas can get inconsistent temporarily, but will eventually reach a consistent state when the partition is resolved.

No finality. To be able to resynchronize after an arbitrary long time of network partition, a design principle of Matrix is to have no upper limit on the time required to eventually reach consistency between replicas: Transactions always get accepted and are never lost regardless of the duration of network partitions. This means that replicas have to accept new transactions regardless of their "happened before" relation to other transaction in the TDAG. Consequently, no parts of the TDAG can ever be considered final.

The TDAG used by Matrix is constructed in an append-only fashion, which especially means that the metadata of transactions is immutable. For referencing, each transaction is assigned a unique identifier by means of a cryptographic hash function based on its immutable parts. This identifier is used for encoding the "happened before" relation, and can be used to verify the integrity of a full transaction given its hash. The fact that Matrix is based on an append-only, fully replicated data structure for transactions, which are linked by the "happened before" relationship using transaction hashes to verify their integrity, suggests that Matrix can be considered a distributed ledger. However, in contrast to other popular distributed ledger technologies, Matrix realizes a distributed ledger without finality.

Overall, only the following weak guarantees are provided by the Transaction Store of Matrix that is implemented as a TDAG:

- the ordering consistency error is lower-bounded to causal order
- the staleness consistency error is lower-bounded to eventual consistency
- no finality: no upper limit is enforced on the resynchronization time for eventual consistency between independent replicas.

In the following, we will analyze the characteristics of an access control system built on these weak guarantees.

2.3 Course of Analysis and Requirements

We follow a two-step approach in our analysis. First, we start in Section 3 by assuming that Transaction Store, Conflict Resolution and Reference Monitor (c.f. Fig. 1) are provided by a central trusted third party, but with the same guarantees as described in Subsection 2.2. This means that the Reference Monitor has to cope with a Transaction Store that only provides a partial order on the stored transactions and no finality. Second, in Section 4, we replace the trusted third party with an architecturally [5] and politically decentralized system. The Reference Monitor, Conflict Resolution and Transaction Store layers are distributed over multiple Matrix servers cooperating to provide the layer, without a single party being in control of all replicas.

In a distributed system, eventual consistency without finality means that arbitrarily old transactions can come in at any point in time. Thus, attackers can send artificially aged transactions. While eventual consistency and non-finality of a partial order originate from the distributed nature of a system, we "concentrate" them in the first step: The Reference Monitor interface allows subjects to append new transactions in an arbitrary "happened before" relation, as long as the new transaction does not contradict other transactions in the chosen causal relation. This way, we can concentrate on dealing with partial order and non-finality without accounting for the technicalities induced by decentralization of Transaction Store, Conflict Resolution and Reference Monitor in the presence of attackers.

In the first step, attackers are malicious subjects limited to the interface provided by the Reference Monitor which try to abuse their interaction permissions and the non-finality of the Transaction Store. Although its interface is simple and provides weak guarantees, as detailed in Subsection 2.2, we facilitate reasoning by modeling the Transaction Store using a trusted third party.

Using the trusted third party abstraction, we generalize the access control system employed by Matrix to an access control model for publish-subscribe systems with the given Transaction Store and Conflict Resolution interface. Note that policy decision by the Reference Monitor requires policy information obtained from the Conflict Resolution layer. Conflict Resolution uses the partial order on transactions provided by the Transaction Store and resolves conflicts by deriving a total order on concurrent transactions. While deriving *some* total order from a given partial order is a well-understood and solved task [14], the task here is to derive a *secure* total order. To derive a secure total order, the conflict resolution mechanism proposed by Matrix is inherently tied to properties of the access control model, which is why both are described conjunctively in Section 3.

To maintain security, the Reference Monitor layer has to fulfill the following requirements:

- interrupt all accesses of subjects on Conflict Resolution and Transaction Store
- allow an initially omnipotent topic creator subject to effectively and granularly pass on or restrict regular and administrative permissions
- prevent any privilege escalation that is not directly or indirectly originating from the topic creator subject
- resolve conflicts to the advantage of honest subjects

In the second step (Section 4), malicious subjects are in full control of their replica, but not of the replica of honest subjects. For the system to be considered secure, we require equivalence between the decentralized system and the system using a central trusted third party for all honest subjects, regardless of the presence of an arbitrary number of malicious subjects whose replicas exhibit byzantine faults.

3 DEALING WITH PARTIAL ORDER AND NON-FINALITY

In this section, we assume that a central trusted third party provides the ledger, i.e. the Reference Monitor, Conflict Resolution and the Transaction Store. The central trusted third party can easily provide the required Transaction Store interface and guarantees, as replication and decentralization is put out of scope. We first describe and analyze an access control model generalized from the Matrix system in Subsection 3.1. This model is used for policies which are evaluated and enforced by the Reference Monitor. The access control model and Reference Monitor do not deal with partial order and non-finality itself, but instead externalize it to the Conflict Resolution layer, which is used as authorization database and described in Subsection 3.2. Resolving conflicts in itself is simple: A conflict between two concurrent, partially ordered transactions is, in theory, resolved by deriving any form of total order. The key point is to derive a *secure* total order, which is neither trivial to define nor to implement. We describe the Matrix approach to secure conflict resolution, which is tied to the employed access control model.

3.1 Level- and Attribute-Based Access Control

Matrix primarily employs a reduced variant of Lattice-based Access Control (LBAC), an access control model where security levels or clearances are assigned to subjects and objects [19]. The security levels are expressed as a partially ordered set. Policies define which security level is required for an access, usually differentiating between read and write accesses to control the flow of information. For example, subjects could be allowed write access if their security level is equal or greater than the object's, and read access if their security level is equal or lower than the object's. Another, more commonly used approach is Role-based Access Control (RBAC) [21]: Access permissions are assigned to roles, and roles are assigned to users. In the Hierarchical Role-based Access Control (HRBAC) variant, a role hierarchy is defined as a partial order, where higher roles inherit all the permissions of lower roles. With a role hierarchy, all LBAC variants can be expressed as RBAC [20].

3.1.1 Entities and Operations. In the following, we will introduce a conceptual model for access control on publish/subscribe interactions of subjects and topics, which is depicted in Fig. 3. The relation between subjects and topics, i.e. whether a subject is subscribed to a topic, is explicitly modeled through a *Subscription*. We assume that subjects have direct access to the Reference Monitor interface shown in Fig. 1 and therefore can define the "happened before" relationship at their discretion, i.e. their parent transactions in the underlying DAG (c.f. Fig. 2). This means that the model performs access control on two operations: appending new transactions to the data store in a subject-defined causal relationship and querying previously published transactions. Subjects might be dishonest and can try to circumvent access control or elevate their privileges.

All transactions have a *type*, which differentiates between transactions with different kinds of application-specific semantics. All types of transactions fall in one of two[5] categories, but which type belongs to which category is up to the application. All transactions

[5]For clarity, we omit that Matrix includes a third type for redacting other transactions, which makes no fundamental difference to the access control model.

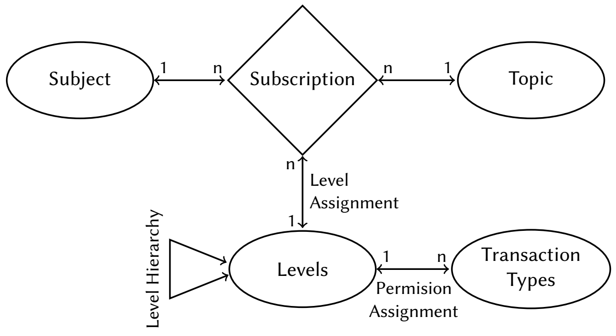

Figure 3: Conceptual Model of Le(A)BAC

```
Subscription      Topic
level: Level      creator: Subscription
...               permissions: Type → Level
                  ...
```

Figure 4: Attributes in Level-based Access Control (LeBAC)

have a `content` field as main payload. The basic form of transactions are *message transactions* which represent one-off messages published by a subscribed sender to a specific topic. The second transaction category are *attribute transactions* which set or update attributes of *Subscription*s and *Topic*s. The attribute name is the transaction's *type*, the attribute value is the transaction's *content* and can be arbitrary, e.g. an integer, string, reference to another transaction, list or map. Usage of the "Query Attributes" interface of the Reference Monitor is treated equal to a query to the corresponding latest attribute transaction of the specified type.

3.1.2 Level-based Access Control. In its basic form, we describe LeBAC, a variant of Biba's LBAC without information flow control. As shown in Fig. 3, each subscription is assigned to a *level*. *Levels* is a totally ordered set representing the permissions a subscriber has, e.g. a set of clearance levels or a range of integers. Requiring a total order on levels is a key difference to LBAC, and the total comparison between subjects by level is used in the Matrix Conflict Resolution layer implementation to deal with partial order and non-finality (see Subsection 3.1). *TransactionTypes* is the set of all existing transaction `types` of all categories, representing the permission to publish a transaction of the specific type. Each level inherits all permission assignments from lower levels, creating a linear hierarchy of permissions. As shown in Fig. 4, subscriptions are assigned to a level through their `level` attribute, and permissions are mapped to levels through the topic's `permissions` attribute. Subjects create new *Topic*s by sending an attribute transaction of type `creator` with themselves as attribute value, targeting a non-existent *Topic*. The transaction is the only transaction which is valid with an empty *happened_before* relation, and grants the creator universal permissions until they send their first `permissions` attribute transaction. Other subjects are subscribed to a topic by permissioned subjects sending an attribute transaction of type `level` targeting the non-existent subscription of unsubscribed subjects. This process results in discretionary access control from the point of view of the topic creator, and mandatory access control for other subscribers, as they can not subscribe to the topic or publish any kind of transaction to the topic at their discretion. However, the creator can transfer permissions, including the permission to change levels and permission assignments. While this effectively creates new subscribers with discretionary access rights, every transfer of permissions can be traced back to the initial creator having had all permissions. Due to administrative permissions being handled with the same primitives as regular permissions, LeBAC is its own administrative model.

For a query operation, the policy evaluation by the Reference Monitor is simple: First, it checks whether the queried topic and transaction exist. If they exist, it checks whether the querying subject has a valid Subscription to the topic at the time of the queried transaction or is the topic creator. To check this, the Reference Monitor uses the Conflict Resolution layer, asking for the value of the `level` attribute of the Subscription and the `creator` attribute of the topic. The point in time is given by passing the transaction to query, for which the Conflict Resolution layer computes the attribute state, i.e. the Subscription's `level` and the topic's `creator`, and returns it to the Reference Monitor. If the subject has been assigned a level, the subject is considered to be subscribed to the topic and the access is seen as valid.

For append operations, the policy to evaluate is more elaborate: Like for the query operation, it begins with checking whether the queried topic exists, and whether the subject has subscribed to the given topic. All transactions require a "happened before" relation to at least one existing previous transaction in an existing topic, transactions of type `creator` are required to be sent to non-existing topics and have an empty "happened before" relation. Using the `type` of the transaction to append, the permissions of the subject are checked. For this, the Conflict Resolution layer is queried for the `permission` attribute of the topic. If there is none, i.e. there has not yet been a permission assignment, the Reference Monitor checks the `creator` attribute of the topic to see whether the subject is the topic creator and therefore is allowed all permissions until first permission assignment. Permission assignments are a map from transaction types to required levels. The reference monitor checks whether the required level is less than or equal to the subject's level. To get the topic's permission assignment and the subject's level, the Conflict Resolution layer is queried for the required attributes, using the list of all "happened before" relations from the transaction, i.e. the causally independent chains to append the transaction to. The response contains the attribute values after combining the state of all causally independent chains. For this, conflicts between chains are resolved to end up with a single attribute value considered to be the current state for the new transaction to append. If the transaction to append is an attribute transaction which targets another subject than the publisher, the Reference Monitor enforces that the publisher has a greater level than the target. Lastly, there is a set of policies that control and restrict the flow of permissions, i.e. attribute transactions relevant for access control. First, a subject that is allowed to set the `level` attribute cannot assign subjects to a greater level than its own level. This means that a subject cannot elevate its own rights, and cannot grant other subjects more rights than it possesses itself. Second, the permission assignment for a given transaction type cannot be elevated to a higher level than the subject's own level. This means that a subject cannot remove permissions from other subjects on the same level. However, permissions can be made unreachable if a single subject which has the greatest level sets the required level for a given `type` to its current level, and then demotes itself to a lower level.

LeBAC's policy decision mainly applies permissions to publishing, not to receiving transactions, i.e. it only checks for the subscription existence. The idea is that in publish-subscribe systems, different receiving permissions are modeled through the allowance to different publish-subscribe topics, but not restricted inside the same topic.

3.1.3 Comparison of LeBAC to other Models. The main difference between LeBAC and LBAC is that LeBAC uses a total order instead of a partial order on levels and has no concept of information flow control. However, the LeBAC system policies include rules which can be considered *permission flow control*, which LeBAC can support due to being its own administrative model. Missing information flow control, however, means that confidentiality cannot be provided if any subject that was made part of the topic is malicious and decides to copy the information and make it available by external means.

Comparison of LeBAC to other access control models yields an impression on its capabilities and limits. Sandhu already showed that RBAC can be used to realise a LBAC model and thus also LeBAC. Yet, the single hierarchy of LeBAC prevents the realization of permission assignments that are not in a total order, i.e. the permission of two subjects are either identical or a strict subset / superset of one another, as subjects higher in the permission hierarchy always have all permissions of any subject lower in the hierarchy. This means that LeBAC is less expressive than LBAC and RBAC. However, the requirement of totality of the "more powerful than" relation is a key point for the Matrix idea of implementing Conflict Resolution, which we will cover in Subsection 3.2, and could not be implemented easily using the partial order permission hierarchies employed by LBAC and RBAC.

3.1.4 Level- and Attribute-based Access Control. To overcome the limited flexibility of LeBAC, the LeBAC policy decision can be augmented with principles from Attribute-based Access Control (ABAC) [12]. Policies are defined by the implementation, using the policy decision of LeBAC, level and permissions attributes, and other attributes of topics and subscriptions.

Level- and Attribute-based Access Control (LeABAC) allows extending the set of attributes used for policy decision as well as to restrict attribute values and value transitions: For example, Matrix extends topic subscriptions to explicitly model the subscription state as a `status` enumeration restricted to values "subscribed", "unsubscribed", "invited" and "banned". This subscription state can be used for policies, e.g. only allow subjects in the "subscribed" state to receive transactions instead of using the existence of a level assignment like in LeBAC. In addition, the transition between states can be restricted, e.g. to disallow the subject of a subscription in the "banned" state from entering the "subscribed" state on their discretion, even if the attribute is on their own subscription.

Also, Matrix extends topics with an attribute for a default level assignment to subjects and categories of transactions. This e.g. allows specifying that publishing attribute transactions requires a level greater than the default subject level, except if the attribute is on a subject's own subscription. In addition, there are attributes to specify whether a topic can be subscribed to by anyone or needs an invitation, whether newly subscribed users can access topic history from before their subscription existed, and many more.[6]

However, due to the Conflict Resolution layer being dependent on the total order on subjects provided by LeBAC's level assignment, the additional attributes from LeABAC are for increased policy expressiveness and do not fundamentally change the dealing with partial order and non-finality.

3.2 Conflict Resolution

The core issue of access control on partially ordered transactions is conflict resolution between concurrent, conflicting transactions. Some form of conflict resolution has to be executed when a new transaction has more than one other transaction in its "happened before" relation, as it then merges two causally independent chains which might contain conflicting transactions. The general idea of conflict resolution is to use a partial order on transactions, and extend that to a total order. The key idea is to use the total order on levels from the access control system as specified in Subsection 3.1 to do this extension. The Conflict Resolution layer is comparable to concurrency control algorithms in database systems [1], for which approaches based on the significance of subjects exist as well [22]. However, a key difference is that, in database systems, concurrency control is employed to schedule the application of transactions on the database, i.e. at write time, while Conflict Resolution is used to get a resolved state from transactions already committed to the database, i.e. at read time.

As message transactions do not modify shared state, concurrent messages can not conflict. However, concurrent attribute transactions can, and conflicts caused by administrative attribute transactions, which modify attributes relevant to the access control decision, are especially sensitive. The challenge is to design a conflict resolution algorithm that combines several properties:

- deterministic: every execution on equal transaction sets should give identical output
- secure: subjects should not be able to manipulate resolution outcome to gain new or regain past permissions
- expectable: subjects expect that appending a new transactions leads to a resolved state close to the recent state, even if the transaction references old transactions
- efficient: every new transaction can trigger the execution of the conflict resolution mechanism, and combine several causally independent transaction chains

In Subsection 3.1, we used the Conflict Resolution layer to get attributes of topics and subscriptions after the combined application of all referenced "happened before" transactions by a new transaction. In general, it is desirable for transactions to include more than one "happened before" transaction, in order to reduce the width of the DAG for efficiency reasons. To achieve determinism, the function has to be a pure function that only depends on the currently known transactions and their causal relation to each other. For security, a major point is that malicious subjects cannot evade the withdrawal of rights by other subjects with a higher access level. For example, if an administrator publishes a transaction that reduces the level of a subject, this subject could fork the DAG before the reduction by publishing an arbitrary concurrent transaction that keeps their level intact, and then merge both causally independent chains by a third transaction referencing both. State resolution has to ensure that the withdrawal always ends up in the resolved attribute set. In a more elaborate version of this attack, the attacker can try to use their past access permissions by publishing an attribute transaction for which their level is not sufficient anymore in parallel to the withdrawal transaction, which then could end up in the resolved attributes when both chains are merged. A state resolution algorithm therefore has to treat causally independent chains with an attribute and chains without an attribute as possibly conflicting, and it has to ensure that the withdrawal is executed before the concurrent transaction. The general idea of how to achieve this is to execute a deterministic topological sorting on the partial order given by the causal relation to yield a linear order in which parallel transactions are applied to the state from before forking. This linear order is what then has to guarantee the above properties. A description of the concurrency in a distributed system with a causal "happened before" relation on events and generating a linear order from it dates back to Lamport's "Time, Clocks, and the Ordering of Events in a Distributed System" [16]. Policy-related events always have to be preferred in order to avoid withdrawal evasion, but without violating the causal relation. An efficient state resolution algorithm minimizes the number of transactions that have to be accessed, and optimally works incrementally, i.e. the result for new transactions can be computed from the result for old transactions without having to take the full DAG into account.

3.2.1 The Matrix State Resolution Algorithm. The idea of the Matrix state resolution algorithm [6, 13] is to split transactions into non-conflicting and possibly conflicting sets, and execute the possibly conflicting transactions in two passes: In a first pass, sort and apply transactions that could lead to a permission reduction, and break ties by preferring transactions of subjects with a higher level. In a second pass, sort and apply the remaining transactions. This is based on the assumption that the subject with higher level is not the attacker, as those subjects actually have the ability to insert concurrent transactions to past transactions that shift state resolution in their favor. We now explain the core ideas of how the Matrix state resolution algorithm works[7], simplified to focus on the access control fundamentals. The algorithm combines the possibly conflicting attribute sets of n causally independent chains into a single attribute set.

(1) For all n causally independent chains: Reduce the chain to a set of the most recent attribute transactions of each `type`.
(2) If a given type of attribute transaction has equal *content* in all n sets, that attribute is considered non-conflicting. The transactions are therefore removed and the attribute is added to the resolved attribute state result set. All remaining attribute transactions are considered to be potentially conflicting.
(3) From the remaining attribute transactions, separate all transactions that can potentially withdraw access rights, as well as all policy transactions relevant for the separated transaction's authorization which "happened before" them. Such

[6]The full system policy definition of Matrix, which Matrix calls its "authorization rules", can be found at https://matrix.org/docs/spec/rooms/v1#id3.

[7]In accordance with https://matrix.org/docs/spec/rooms/v2

authorization transactions are the `level` and `permissions` transaction from LeBAC, but also include transactions relevant for LeABAC extended policy decision making. The `creator` transaction is relevant for authorization as well, but as there can neither be a second `creator` transaction nor a concurrent transaction, it cannot end up in the conflicting transactions.

(4) Sort the extracted transactions topologically using the partial order given by their transitive happened before relation in the DAG. To resolve ties in a deterministic and secure manner, for two concurrent transactions, the smaller one is:
   (a) The one whose `sender` has a higher level at the point of those transactions
   (b) The one with the earlier sender timestamp.
   (c) The one with the lower hash value.

(5) Apply the linearized transactions to the attribute state result set from the non-conflicting transactions. For each transaction, check before application whether the access control system would accept the transaction based on the current state, as specified in Subsection 3.1. If this is not the case, the transaction is ignored. This is why the relevant authorization transactions are required: The authorizing transactions for the withdrawal transaction has to be applied to the attribute state before the withdrawal transaction.

(6) Take the remaining conflicting transactions and sort them topologically by the partial order given by their transitive happened before relation in the DAG. To resolve ties in a deterministic and secure manner, for two concurrent transactions, the smaller one is considered to be:
   (a) The one with the earlier sender timestamp.
   (b) The one with the lower hash value.

   The idea is that because all permission withdrawals are already applied, the tie-breaking step of checking for the higher `level` can safely be omitted and instead applied in an order which is closer to the actual temporal order.

(7) Apply the linearized transactions to the resolved attribute state result set from the non-conflicting and withdrawal transactions. For each transaction, check before application whether the access control system would accept the transaction based on the current state, as specified in Subsection 3.1. If this is not the case, the transaction is ignored.

The algorithm extends transactions by a `sender_timestamp` attribute which is set to the transaction publication time by honest subjects. However, it can be set to arbitrary values by faulty subjects, especially they can set their `sender_timestamp` to an arbitrarily high value to always appear as the most recent transaction after the resolution. In contrast to arbitrary, i.e. hash-based orderings, timestamps have a meaning for users and can be easily recognized as forged or improbable. Honest subjects with a higher power level can then still enforce permission removal on the faulty subject, as their transactions administrative actions are favored in step (4). As soon as a honest subject then sends a new transaction that references the faulty one as "happened before", the advantage is gone as causal ordering is used before `sender_timestamp`.

3.2.2 Soft Failure. A general problem for all conflict resolution algorithms is that, due to the non-finality of the "happened before" DAG, attackers can always append transactions to arbitrary existing transactions, including "old" ones, and exercise the permissions they had at that point in time. As transactions that purposefully reference old transactions cannot be distinguished from actually old transactions that just have not been successfully synchronized, those transactions have to be treated as valid, due to the lack of some form of consensus on a point in the DAG after which all transactions are seen as final. For attribute transactions, the state resolution algorithm as described above will handle this, because, as soon as a new transaction unifies both chains, state resolution will prioritize the possible withdrawal of rights and authorize that parallel transaction afterwards. However, message transactions are always accepted and end up in the topic's message history if they were valid at that point in time. To prevent this attacker behavior, Matrix uses the concept of "soft failure" as a third state between accepted and rejected transactions: If a newly received transaction does pass the authorization checks as described in Subsection 3.1, but would not pass the authorization checks based on the hypothetical current attribute state of the topic, i.e. assuming that the transaction would attach to *all* current leaf transactions, it is assumed to be a malicious event and soft-failed. A soft-failed transaction will be appended at its place, but will neither be relayed to clients nor be used in the "happened before" relation of new transactions. Nevertheless, the transaction takes part in state resolution as normal. The idea is that accepting but not delivering the malicious messages disincentivizes sending such transactions.

3.2.3 Optimization: Proof of Permission. The actual Matrix state resolution algorithm includes an important optimization [13]: Each and every transaction has a reference to all policy and permission transactions that are required to authorize the transaction, i.e. a "proof of permission". State resolution is executed using the partial ordering given by the "Authorization DAG" that emerges from recursively taking in all relevant policy and permission transactions, instead of the DAG that emerges from the causal relation of all transactions. This reduces the need to keep every transaction as long as a topic exists to those transactions contained only in the "Authorization DAG" as well as the most recent attribute transactions. This also enables a transaction retention time not only dictated by disk space or other resources, but actually as a subject-defined topic policy. However, this optimization has the potential for new security issues that are not apparent in the base idea. For example, there are circumstances where non-conflicting transactions can be overwritten by mistake due to not using the full "happened before" relation. The optimization therefore needs an additional last step where all non-conflicting transactions are applied to the result again.

4 DECENTRALIZATION AND ASSESSMENT

The explained access control model and conflict resolution is independent of whether it is implemented as a centralized, distributed, or decentralized system. Both, a distributed and a decentralized implementation, have to distribute the Reference Monitor and the Transaction Store layer to multiple nodes. We now focus on a decentralized implementation where nodes (servers) are operated by (politically) independent entities.

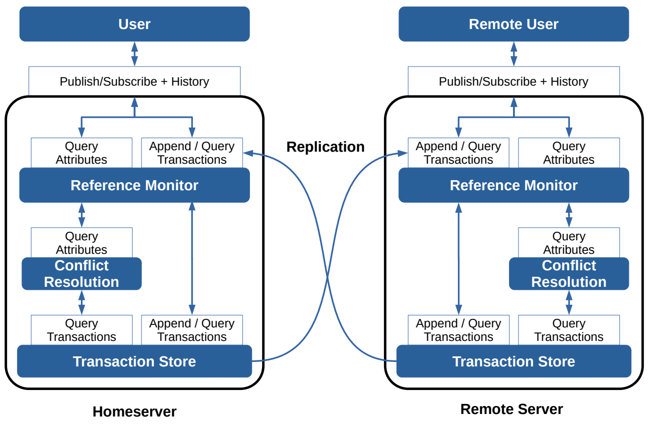

Figure 5: Layer model for a decentralized TDAG and Reference Monitor. Rounded, blue-filled boxes are layers, rectangles are interfaces provided by the lower layer and used by the upper layer. In contrast to Fig. 1, this shows entities and not only functionality.

4.1 Decentralized Access Control with Eventual Consistency

Figure 5 shows the decentralized implementation of Matrix. The Transaction Store can be implemented by a Transaction-based Directed Acyclic Graph (TDAG) in a decentralized manner where servers maintain a local replica of the TDAG for their clients. Each topic has its own, independent TDAG representing its full history. This means that only servers with users subscribed to a topic take part in a given TDAG, and therefore the concepts inherently provides sharding. We say a server is subscribed to a topic when at least one user of the server subscribed to the topic. Each server subscribed to a topic maintains a local copy of all transactions related to the topic, i.e. a replica of the topic's TDAG.

A new transaction to be published and given to a server is checked for validity and authorization by the local Reference Monitor and then appended to the server's local replica. The "happened before" transactions of this new transaction are determined by the server, always using transactions without descendants known to the server. In terms of the causal order given by the DAG as shown in Fig. 2, transactions without descendants are the set of causally latest transactions. Matrix limits the number of referenced "happened before" transactions to reduce the effort required for conflict resolution. Concretely, from the set of all transactions without known descendants, servers choose the five known transactions with the longest distance to the unique, causally earliest transaction in the DAG, i.e. the `creator` transaction, and five random transactions with a shorter distance to the earliest transaction. This strategy is intended to ensure that all causally independent chains eventually converge during a phase without concurrent transactions. Afterwards, the transaction is broadcasted to all other subscribed servers. As shown in Fig. 5, subscribed servers will enforce that new transactions from other servers pass the local Reference Monitor based on the current policy information in their TDAG replica. If the authorization checks are passed, the servers append the new transaction to their local replica. If a transaction is 'dropped' on a link between two servers, e.g. due to network partitions, the DAGs will get out of sync, i.e. get in an inconsistent state. All servers will still always accept local transactions if they pass their authorization checks done by their Reference Monitor. In the underlying DAG, as shown in Fig. 2, appending concurrent transactions leads to a fork in the DAG and two causally independent chains. As soon as the partition is recovered and the servers receive a new transaction whose "happened before" transactions are missing in their replica, the server will query other servers for the missing transactions round-robin. The receiving server can verify that the transaction is correct by computing its hash. This process is called *backfilling* in Matrix. Backfilling is executed recursively, walking down the "happened before" relation until the server encounters a known transaction. Through backfilling and the append-only nature of the TDAG, eventual consistency is achieved. After eventual consistency is achieved, new transactions will reference both causal chains in their "happened before" relation, which will merge the chains again, while potential conflicts in attributes between the two chains are handled by Conflict Resolution.

As shown in Fig. 5, each server independently operates a Conflict Resolution layer on its TDAG replica to compute current attributes values, as well as a Reference Monitor layer to enforce authorization policies. As stated in Subsection 3.2, the conflict resolution algorithm is required to be deterministic and to only have the TDAG as input. To achieve equivalence of the distributed Reference Monitor and Conflict Resolution with the trusted third party model from Section 3, the implementations executed by the replicas have to be equivalent with each other. We say that two implementations are equivalent if they derive identical attributes and access decisions based on equivalent sets of transactions. This means that employed algorithms can e.g. differ in complexity, as long as their output remains the same. While each server executes Conflict Resolution and Reference Monitor independently, with equivalent, deterministic algorithms and an eventually consistent Transaction Store, all honest, non-faulty servers will eventually reach agreement with each other on whether a given transaction should be accepted. Thus, the Reference Monitor that was provided by the trusted third party in the conceptual model is distributed and placed on all subscribed servers, which interrupt incoming as well as outgoing transactions as seen in Fig. 5, make an independent access control decision based on its current state, and enforce the decision regardless of the decisions of other servers. Servers do not vote or announce some form of consensus, there is no quorum, and majority does not win.

In addition, decentralized implementations can introduce specific attributes that are relevant for access control, e.g. Matrix allows for a "server access control list" attribute which bans certain servers and subsequently all of their users from participation in a topic.

4.2 Assessment of TDAG-based Decentralized Access Control

Let us now come back to the guiding question given in the Introduction and analyze whether the requirements stated in Subsection 2.3 are fulfilled. In summary, the access control of Matrix or of a similar TDAG-based system are based on the following assumptions:

(1) Out of two subjects, the one with the higher level is the 'honest' one.

(2) Authorization policies and conflict resolution are deterministic and equivalently implemented by all 'honest' subjects.
(3) Whatever an attacker is doing: authorization policies and conflict resolution
    - do not allow unauthorized transactions
    - do not allow unauthorized privilege escalation
    - always prefer the 'honest' subject.

While the first assumption is simply an axiomatic one ("by design"), the second and third assumptions are not easily guaranteed. We proceed as follows: first, the attacker model is clarified and some existing countermeasures against some attacks are listed. Based on the first step, we argue in an 'evidence-based' way that guaranteeing assumptions (2) and (3) requires more than the existing countermeasures by showing a class of attacks on each assumption. In particular, we derive the need for a formal verification of the conflict resolution mechanism and authorization policies.

4.2.1 Attacker model and existing countermeasures. It is sufficient to look at the possibilities of an attacker subscribed to a single topic, as each topic is strictly independent from any other topic and the subscribed servers form an independent federation without any non-subscribed servers. The assessment focuses on attackers that participate in the system, assuming that means like mutual authentication and encryption prevent man in the middle attacks.

The TDAG approach works in the following attacker models: honest users are on honest servers, byzantine users are on byzantine servers they fully control, and byzantine users are on honest servers they don't control. As a homeserver acts as representative for its users and is their only source of information on the current state of topics, a user's homeserver is in the position of a Dolev-Yao attacker and is also in full control of the reference monitor for their users. This trust model is comparable to e-mail with the difference that the Matrix protocol mandates transport layer authentication, integrity checks, and encryption between servers.

To cope with message forging or tampering by byzantine servers, each server has a public/private key pair that is used to sign transactions published by its users. Subscribed servers can validate the signature and verify whether the transaction was sent by the origin server and is unmodified. To provide some end-to-end guarantees even with malicious servers, authenticity, integrity and non-repudiation could be provided by public/private key pairs for each user, signing both message and attribute transactions, but only encrypting transactions which are not needed for server-side policy enforcement. Keys can be verified by a Public Key Infrastructure or out-of-band / in-person validation. Alternatively, depending on the concrete goals, ratcheting-based end-to-end encryption protocols can provide repudiation. Matrix currently only supports the repudiable end-to-end encryption through their own cryptographic ratchet protocols, which are based on the Signal protocol, and in-person validation. If both repudiable and non-repudiable end-to-end encryption were supported by Matrix, a topic attribute could specify the concrete per-topic end-to-end security mechanism. Also, Matrix currently does not encrypt Attribute Transactions end-to-end, which would be possible for all attributes that are not relevant for policy decision. While providing end-to-end guarantees are one way of reducing the impact of a malicious server that has honest users, i.e. reducing the trust required in a user's server, the reference monitor is still controlled by a malicious party, and metadata is not protected. The Matrix project tries to solve those issues by making it possible for users to be multihomed, i.e. a user's account can be associated with multiple servers, which easily allows them to move away from a server in which they lost trust. In addition, the goal is making server implementations and transaction distribution efficient enough so that they can be run directly on the user's devices [9], making Matrix a hybrid of a federated and peer-to-peer protocol. Multihomed users will pose an interesting new challenge for the access control system, as a server then has to prove that it currently is allowed to represent a given user, which requires some kind of access rights delegation from users to servers. Peer-to-peer servers also need to remove the dependency on the Domain Name System for inter-server authentication. This also improves security: An attacker which gains control over a server's domain name cannot impersonate the server anymore and e.g. access and backfill transactions in all topics that impersonated server has access to.

In byzantine fault tolerant consensus systems or distributed ledger technologies, byzantine fault resilience is usually expressed as the share of the full network controlled by attackers that consensus finding can tolerate. For the decentralized access control system described in Subsection 4.1, this logic is not applicable, since, through the causal order on transactions and the decentralized access control decision and enforcement, the system does not have to solve a consensus problem. Regardless of the number of attackers, each and every user has their own reference monitor in form of their homeserver which enforces access control on incoming and outgoing transactions. As every reference monitor makes its access decision independently, i.e. does not care about the access decision of the majority of subscribed servers, attackers cannot influence the decision through their numbers of participating servers. As long as an attacker cannot interrupt the connection between two honest servers, those will reach a consistent DAG state, enforce the same access control, and provide availability for their users, even if all other servers subscribed to a topic are malicious.

4.2.2 Attack surface and found issues. The (evident) first class of attacks is enabled by incorrectly specified policy rules that give someone 'unintended' access rights and, thus, allow for privilege escalation or the capability of deleting a user's access rights. While this class of attacks is, of course, not specific to Matrix or TDAG-based approaches, we encountered this type of policy misspecification in the Matrix specification (see below). This class of attacks targets assumption (3). The (less evident) second class of attacks is enabled by deviations in how servers deal with conflict resolution, which targets assumption (2). Both classes of attacks result in the universal or partial acceptance of a malicious transaction. Thereby, an attacker could, for example, unsolicitedly subscribe to a topic breaking confidentiality, elevate access rights to a higher level, or even break the eventual consistency by having different servers make different policy decisions. While the first class is present in the trusted third party conceptual model as well, the second class only exists in the decentralized implementation due to differing behavior in the decentralized Reference Monitor implementation.

When analyzing the Matrix specification and implementations, we found the following vulnerabilities which mainly belong to the second class of issues, i.e. subtle differences between specification and implementations due to inconsistencies or omissions on either side. Concretely, this class describes the issue of two groups of servers making different decisions on which transactions to include in the DAG. For this, two server implementations that deviate in behavior as well as an attacker are needed to be subscribed on the same topic. Depending on the inconsistency, the attacker needs to control not only their client but also their server, and have sufficient access rights to cause the inconsistent behavior. If one implementation accepts the attacker's transaction and the other implementation rejects it, the attacker forced an inconsistency of the DAG between the implementations that cannot be resolved, breaking eventual consistency. If the attacker transaction is relevant for access control, this will lead to a deviation in the policy information and therefore to the Reference Monitors making different access control decisions. But even if not for the policy information, broken eventual consistency leads to two independent views of a topic, which cannot be combined again and can render the topic unusable for its purpose and requires to switch to a new one without the malicious transaction. This situation is reminiscent of a "hard fork" in blockchain distributed ledger technologies.

The found issues were as follows:

a) The wire format of integers in the specification restricts their value to the range $[-2^{53}+1, 2^{53}-1]$. This restriction was not enforced by synapse[8], the reference Matrix server, and effectively limited to the range of 64 bit signed integers. This allows for a class two attack. While this attack can be executed on any integer present in the wire format like the sender timestamp, it can especially be used by the attacker to send a `level` attribute transaction with a value beyond the range allowed in the specification but still accepted by synapse, leading to an inconsistent policy database between synapse and a specification-compliant homeserver. This is planned to be corrected in room specification v7.

b) For the permission to publish a message transaction with a highlighting notification, the authorization policies did not follow the general rule of denying to set the required level higher than the policy sender's current level. This is a class one attack that allows to remove the permission to send notifications from other users with the same level. Synapse did implement the check anyway and rejected such transactions, but a specification-compliant implementation would have had to accept it, leading to a possible class two attack. This will be corrected in room specification[9] v6.

c) The specification does not limit the number of transaction linked as "happened before" and does not specify a transaction selection algorithm other than taking all known leaf transactions, but Synapse does reject transactions with more than 20 parent transactions. This can be abused by an attacker by sending enough concurrent transactions to always make sure there are more than 20 leaf transactions in the DAG. The specification-compliant servers will list all of them as "happened before", which will lead synapse to reject the transaction. In a variation of mechanism from the other class two attacks, this does not send an in-itself malicious transaction, but seemingly honest transactions lead to a denial of service on the channel between the specification-compliant server implementations and synapse. As an effect of the denial of service on the channel, this breaks eventual consistency and therefore is a class two attack. This is planned to be corrected in room specification v7, and will be retroactively enforced in older versions.

d) Server access control lists allow publishing subjects to ban a server including all of its users from a topic. However, no equivalent of the rule of not being able to withdraw access rights from subjects with a higher level was demanded, which allows the attacker to ban users with a higher level. This an error in the authorization policy and therefore a class one attack. While synapse accurately implemented this inconsistency in the specification, alternative implementations could deviate from the specification replace it with a secure version, e.g. enforcing a check that none of the users on the banned server have a higher level than the banning user, which would then lead to an additional class two attack. A mitigation in synapse was introduced[10] which restricts the modification of server access control lists to administrators by default.

All issues were responsibly disclosed to the Matrix core team.

4.2.3 Gained insights. The security of Matrix faces two fundamental threats: Incorrect specification, i.e. authorization errors also present in the trusted third party model, and non-equivalence, i.e. divergence of decentralized implementations that leads to divergence from the trusted third party model. All security issues from Subsection 4.2 were implications from those threats. One important way to mitigate both threats and, thereby, to prevent all found issues, is the formalization of the authorization policies and conflict resolution algorithm. Using the formalization to prove security and correctness properties helps to prevent the first threat. We identified the following security properties as crucial to be satisfied by any set of authorization policies and conflict resolution mechanism to be secure:

- The algorithms have to be deterministic and only depend on data present in the DAG, to create a decentralized reference monitor that makes the same policy decision at each consistent replica.
- Users are unable to gain permissions out of nowhere, but have to receive those permissions from someone who has the permission to provide others with them.
- If two transactions are conflicting, the one with the higher level has to be preferred[11].
- A permission withdrawal always has to be preferred to a concurrent permission usage with lower level.

While the set currently employed by Matrix is engineered to fulfill those properties in most cases, any action for which they are not satisfied is a potential security vulnerability.

To protect against the second threat, the formalization can be used to generate code in any language the implementations require, providing them with equivalent implementations of authorization policy checks and conflict resolution. In addition, in face of new and improved algorithms, the formalization can be used to prove equivalence and therefore compatibility with the current ones without relying on empirical evidence.

[8]https://github.com/matrix-org/synapse/
[9]https://github.com/matrix-org/matrix-doc/pull/2240
[10]https://github.com/matrix-org/synapse/pull/6834
[11]In case of same level, a deterministic tie break is required.

The security assessment shows that a valid decentralized access control system can potentially be built on top of the weak guarantees of causal order, eventual consistency and no finality, but requires that the distributed reference monitor instances make the same policy decisions when presented with the same data. Here, some form of logical centrality comes into play. The use of Matrix and its access control approach as an open, decentralized system in "mission-critical" environments should, thus, require formal verification of authorization policies as well as of conflict resolution.

5 CONCLUSION & FUTURE WORK

We positioned and generalized the concepts used by the Matrix federated publish-subscribe middleware with respect to both access control models and distributed ledger technologies. We have shown that a causal-order TDAG-based distributed ledger without finality can indeed be sufficient for the decentralized implementation of a conceptual model for access control systems based on Lattice- and Attribute-based Access Control. We described and categorized the model, and explicitly stated its assumptions and required interface. To the best of our knowledge, Matrix is the only system that implements access control based on an eventually consistent partial order without finality and without a consensus algorithm. While this results in a valid access control system, the system behaves differently than traditional, consensus-based access control systems: Matrix allows for "pluralism of opinions" on the current state of the system and provides access control mechanisms that cope with that fact instead of following the majority or an assigned leader. Therefore, a good understanding of the resulting consequences is recommended with regard to deployments in sensitive environments.

Our security analysis found no inherent flaws in the decentralized implementation of the reference monitor and policy information data structure, but showed possible points of attack on concrete implementations. The found issues were disclosed responsibly. In our security assessment, we stressed the importance of specifying the conflict resolution algorithm as the core security mechanism in a formal calculus. First, this allows code generation from the calculus as a single source of truth, thus avoiding implementation differences. Second, this allows a formal proof of the security of the authorization policies and conflict resolution as crucial parts of the decentralized access control system.

ACKNOWLEDGMENTS

We thank the Matrix developers for their ingenious system design and their open and cooperative spirit towards the research community.

REFERENCES

[1] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. 1987. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley. ISBN: 0-201-10715-5.

[2] Matt Blaze, Joan Feigenbaum, and Jack Lacy. 1996. Decentralized Trust Management. In *Proceedings of the 1996 IEEE Conference on Security and Privacy* (SP'96). IEEE Computer Society, Oakland, California, 164–173. ISBN: 0818674172.

[3] Eric Brewer. 2012. CAP Twelve Years Later: How the "Rules" Have Changed. *Computer*, 2, 23–29.

[4] Eric A Brewer. 2000. Towards robust distributed systems. In *PODC*. Vol. 7.

[5] Vitalik Buterin. 2017. The Meaning of Decentralization. (Feb. 6, 2017). https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274.

[6] Travis Ralston et al. 2019. Room version 2. Retrieved Jan. 31, 2020 from https://matrix.org/docs/spec/rooms/v2.

[7] Matthew Hodgson. 2019. Matrix in the French State. https://fosdem.org/2019/schedule/event/matrix_french_state/.

[8] Matthew Hodgson. 2019. The 2019 Matrix Holiday Update! https://matrix.org/blog/2019/12/24/the-2019-matrix-holiday-update.

[9] Matthew Hodgson. 2020. The Path to Peer-to-Peer Matrix. https://fosdem.org/2020/schedule/event/dip_p2p_matrix/.

[10] Mike Hoye. 2019. Synchronous Messaging at Mozilla: The Decision. (Dec. 1, 2019). https://discourse.mozilla.org/t/synchronous-messaging-at-mozilla-the-decision/.

[11] Florian Jacob, Jan Grashöfer, and Hannes Hartenstein. 2019. A Glimpse of the Matrix: Scalability issues of a new message-oriented data synchronization middleware. In *Proc. ACM 20th Int. Middleware Conference Demos and Posters*, 5–6.

[12] Xin Jin, Ram Krishnan, and Ravi Sandhu. 2012. A unified attribute-based access control model covering DAC, MAC and RBAC. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 41–55.

[13] Erik Johnston. 2018. State Resolution: Reloaded. https://github.com/matrix-org/matrix-doc/blob/server_server/release-r0.1.3/proposals/1442-state-resolution.md.

[14] Arthur B Kahn. 1962. Topological sorting of large networks. *Communications of the ACM*, 5, 11, 558–562.

[15] Niclas Kannengießer, Sebastian Lins, Tobias Dehling, and Ali Sunyaev. 2019. What Does Not Fit Can be Made to Fit! Trade-offs in Distributed Ledger Technology Designs. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*.

[16] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21, 7, 558–565.

[17] Lauri IW Pesonen, David M Eyers, and Jean Bacon. 2007. Access Control in Decentralised Publish/Subscribe Systems. *JNW*, 2, 2, 57–67.

[18] Serguei Popov, Olivia Saa, and Paulo Finardi. 2019. Equilibria in the Tangle. *Computers & Industrial Engineering*, 136.

[19] Ravi S. Sandhu. 1993. Lattice-based access control models. *Computer*, 26, 11, 9–19.

[20] Ravi S. Sandhu. 1996. Role Hierarchies and Constraints for Lattice-Based Access Controls. In *ESORICS*.

[21] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. 1996. Role-based access control models. *Computer*, 29, 2, 38–47.

[22] Tomoya Enokido and Makoto Takizawa. 2005. Concurrency control based on significancy on roles. In *11th Int. Conf. on Parallel and Distributed Systems (ICPADS'05)*. Vol. 1. (July 2005), 196–202. DOI: 10.1109/ICPADS.2005.112.

[23] Kaiwen Zhang and Hans-Arno Jacobsen. 2018. Towards Dependable, Scalable, and Pervasive Distributed Ledgers with Blockchains. In *ICDCS*, 1337–1346.