Bachelor's Thesis Computing Science

From Multi Server Authentication to Multi Server Authorisation

Exploring architectures for distributed access management in the PEP repository

Mitchell Boesveld
s1045762

May 8, 2024

First supervisor/assessor:
Dr. Bernard van Gastel

Second assessor:
Dr. Ir. Erik Poll

Second supervisor:
Job Doesburg

Radboud University

Abstract

In this thesis, we explore approaches to distributed access management. We do this based on a case study of the PEP Responsible Data Sharing Repository (PEP Repo). PEP Repo is a secure storage system that provides pseudonymisation, encryption and therefore privacy of personal data, for instance for research purposes of medical professionals. A large part of the PEP Repo components are implemented in twofold, to prevent a Single Point Of Failure from compromising that component. For instance, user authentication is done by two servers. However, a Single Point Of Failure still remains in the implementation of authorisation in PEP Repo. This problem prevails also in systems similar to PEP Repo. In this thesis, we explore two scenarios in which this problem can be mitigated. One is with a set of access rules that is managed by a single entity (SSoT), the other is a scenario in which access rules are managed by multiple entities (MVoT). For both, we describe how verifying of access rules works. For the Single Source of Truth (SSoT) scenario, we explore existing distributed systems algorithms to maintain a synchronised state between multiple entities, and describe how they can be applied to reach consensus between access lists. For Multiple Versions of Truth (MVoT), we describe how Multi Party Authorisation can be implemented in PEP Repo, and what the consequences of using this model are. We evaluate all options from both the SSoT and MVoT scenarios based on reliability, implementability, maintainability, security, network efficiency and cost efficiency. We conclude with a review of the found systems, based on this evaluation. Applying the MVoT model and using the Multi Party Authorisation system is found to be the most fitting solution.

Contents

Chapter 1

Introduction

Since the General Data Protection Regulation (GDPR) was introduced in the European Union, there has been an ever-increasing need for solutions to the privacy requirements this regulation stated. One of these requirements is that data may only be processed for a specific and valid purpose. This principle is called "Data minimisation" in the GDPR [1]. The PEP Responsible Data Sharing Repository (PEP Repo) is a secure data storage solution that aims to facilitate in solving this problem, by providing access to only relevant subsets of data instead of all data. PEP Repo also provides pseudonymisation of the data subject's identity, minimising data even more by removing linkability between the data and the data subject [2].

Even though the fundamental cryptography behind PEP is rigid and well-proven [3], weaknesses arise at the implementation in PEP Repo. For instance, if the main private key leaks, the whole system is compromised, no matter the rigidity of the cryptography. That is why most parts of the PEP Repo system are implemented redundantly: more than one part can fail before the system becomes insecure. For instance in authentication, two servers have to independently authenticate a user for the system to accept their established identity. We call this Multi Server Authentication. This means that if one server is compromised, authentication as a whole is not.

However, because there is currently no redundancy in the authorisation of PEP Repo, security concerns prevail at the point of granting certain people (or groups) access rights to parts of user data. This problem also exists in similar systems to PEP Repo. If a malicious actor would be able to modify the table of access rights, confidentiality, integrity and availability of the user data can no longer be guaranteed, no matter the cryptographic rigidity of the system. This means that the way authorisation is often implemented poses a Single Point Of Failure (SPOF) when updating or modifying access rules.

The goal of this thesis is to explore the solutions to this problem and analyse them based on reliability, implementability, maintainability, security, network efficiency and cost efficiency. We utilise PEP Repo as a specific use case to represent systems these solution apply to. Ultimately, the result of this thesis is a reflection on the solutions that are available, with regards to these aspects. We distinguish two cases (see also chapter 4 and chapter 5): one with multiple servers performing authorisation based on a Single Source of Truth (SSoT) leading to Multi Server Authorisation, and one with multiple servers administered by different parties and thus with Multiple Versions of Truth (MVoT), leading to Multi Party Authorisation.

The scope of this thesis is limited to the theoretical exploration of possible ways the SPOF in authorisation can be reduced to two or more points of failure, and how we can thus achieve Multi Server Authorisation (MSA) or, as an extension of this, Multi Party Authorisation (MPA). The cryptographic details of PEP and implementation details of the solution are out of scope, as we aim to find a general approach to solve this problem.

The result of this thesis is an overview of the options there are to split the authorisation responsibilities of PEP Repo over multiple entities. These results are then analysed and compared. In the end, the thesis concludes in a recommendation as to which system is most fitting in respect to the requirements that were posed.

Our approach is as follows. We first review existing literature on distributed systems, and select applicable consensus algorithms. We do this based on, among other sources, the book “Designing Data Intensive Applications” [4], as this is a very extensive source of these algorithms. We also consider a separate SPOF in the current authorisation system, namely the problem of only one person having full authority over the access rules that are in effect.

Then, we use PEP Repo as a case study and explore the ways these findings can be applied to it as it is currently implemented, or the ways the PEP Repo system has to change in order to allow for more secure authorisation. All options are compared and scored based on reliability, implementability, maintainability, security, network efficiency and cost efficiency.

Chapter 2

Current state of access control in PEP Repo

To understand how authorisation works in the PEP Responsible Data Sharing Repository (PEP Repo), we first explore the current way authentication is implemented. This is mostly done separately from the authorisation process, making it almost a completely separate system. However, since both authentication and authorisation are needed for access management, they are both related to the solutions this thesis will provide.

2.1 Authentication

Authentication currently works via Security Assertion Markup Language Protocol (SAML) [5]. SAML uses the concept of a Service Provider (SP) and an Identity Provider (IdP). In the case of PEP Repo, SURFconext is the IdP, and the PEP Repo authentication server is the SP. To mitigate a Single Point Of Failure (SPOF), the current implementation uses two PEP Repo authentication servers.

The authentication flow is as follows[1] (see Figure 2.1). A PEP Repo user logs in to SURFconext, and a request is sent to the first authentication server. This server signs the first authentication step, and sends it back to the client. The client then verifies the signature, and sends it and a second authentication request to the second authentication server. This part can happen without end-user interaction, as the SURFconext session will still be active once the response from the first authentication server arrives at the client. After the client has sent the second request to the second authentication server along with the signature from the first authentication

[1]PEP Repo currently has two Authentication Servers, but the system is set-up to allow n Authentication Servers.

server, the second server will sign the authentication attempt along with the signature from the first server and send it back to the client.

In the end, the PEP Repo user has a certificate that proves his identity. This certificate is signed by two servers, taking away SPOF a system with one server would have. With the certificate that was obtained, the user can now go to the Access Manager (AM) and Transcryptor (T) layer, and perform the actions they have permissions for. This is where authorisation begins.

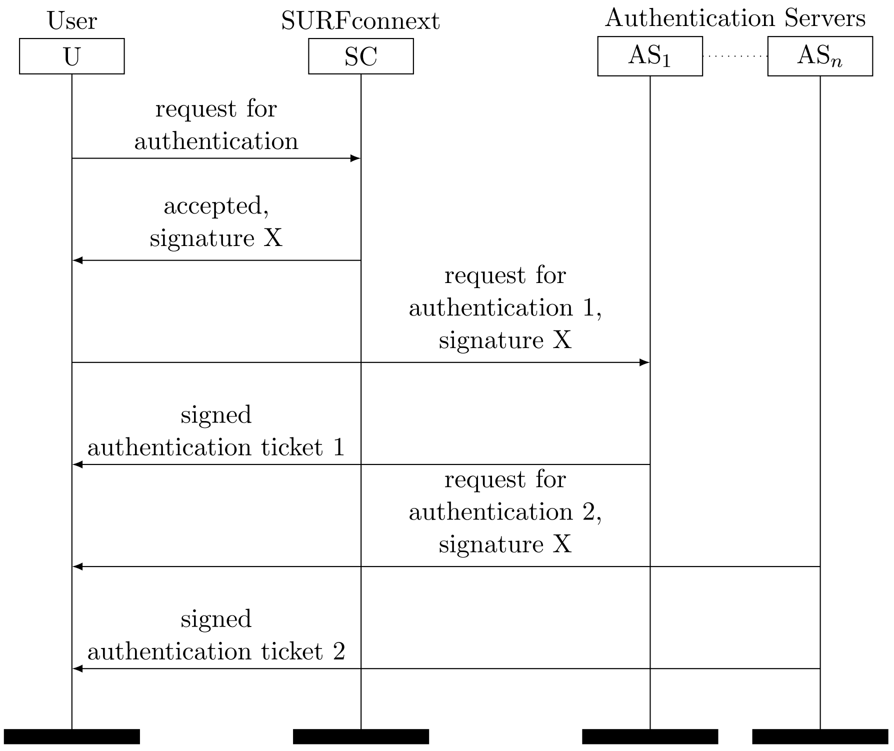
User
U
SURFconnext
SC
Authentication Servers
AS$_1$
AS$_n$
request for authentication
accepted, signature X
request for authentication 1, signature X
signed authentication ticket 1
request for authentication 2, signature X
signed authentication ticket 2

Figure 2.1: graphical representation of authentication process

2.2 Authorisation

2.2.1 Changing access rules

Currently, PEP Repo stores access rules in a server called the Access Manager (AM). Only an Access Administrator (AA) can change the rules that the AM enforces. The AM stores a table of usergroup-datagroup relations, with the options for read access and write access. To update these rules, the AA can obtain an authentication certificate. With that certificate, a request to change an access rule can be sent to the AM server. This server then verifies that the AA is allowed to change the rule and changes it accordingly.

2.2.2 Verification of access rules

Verification of the access rules works as follows (see Figure 2.2). A user that wants to retrieve data from the system sends a request to the AM with the certificate that was obtained during authentication, which verifies if the user has the permissions to do so. If correct permissions are present in the AM, it will pass the request to a server called the Transcryptor (T). This server logs the action, does the proper pseudonymisation and cryptographic operations, creates a database request ticket and sends the request to the AM, which sends it back to the client. The client can then send a request to the database with the ticket obtained in the previous steps, and retrieve the pseudonymised data[2].

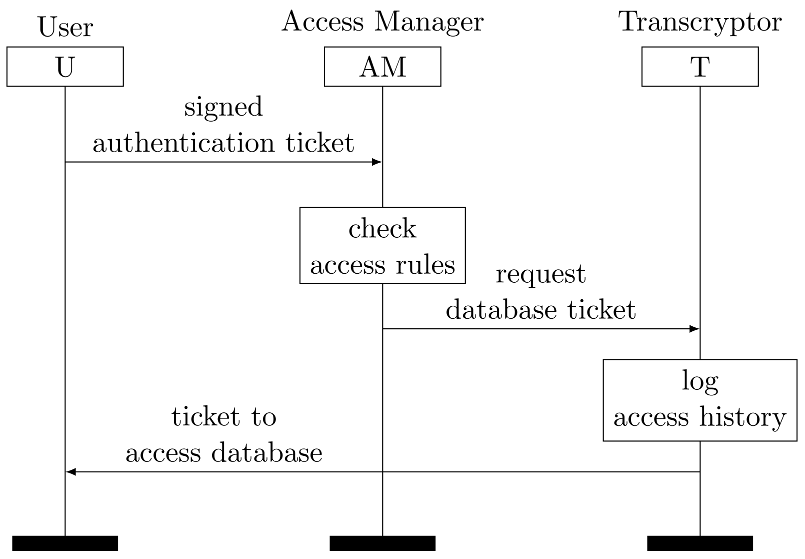

Figure 2.2: graphical representation of authorisation changes

[2]The retrieved data also needs to go through the PEP Repo system again for transcription

Chapter 3

Problem description

As described in the previous chapters, the main goal that this thesis tries to accomplish is to eliminate the Single Point Of Failure (SPOF) in authorisation. We do this by using PEP Responsible Data Sharing Repository (PEP Repo) as a case study. In PEP Repo, there are two main parts of authorisation, namely the updating of access rights, and the verifying of access rights. In this chapter, we highlight the importance of removing the SPOF in authorisation, and explore why it is a problem. Specifically, we discuss an attack model on the authorisation part of PEP Repo that the current implementation is vulnerable to.

In the current PEP Repo implementation, only the Access Manager (AM) stores and verifies the access rules to the system. If the AM affirms a users' access rights, it will relay the request to the Transcryptor (T). T then logs that certain data was accessed and does the required cryptographic operations, but does not prevent illegal retrievals. Verifying at T cannot be easily implemented in the current system, as T does not have the access rules.

3.1 An attack on the Access Manager

To update access rules of the PEP Repo system, there are two administrators: the Access Administrator (AA) and the Data Administrator (DA). The DA defines an access context, consisting of a relation between a datagroup and a usergroup. The AA can give rights to this relation, specifically read- and write capabilities. The DA is out of scope in the rest of this thesis, because they are not directly involved with access control.

Assume a compromised AM. This means that access rules to usergroup-datagroup relations can be changed to the liking of the attacker. At first glance, it seems that this allows only existing users (for instance, researchers) to gain more privileges than allowed, as it can impersonate the capabilities of the AA. However, an attacker can also create a new user and access context, as this is also only handled by the AM. The combination of the ability to add an access context and the ability to assign all rights to it give the attacker complete access to the data. Note that this data will still be pseudonymised, as colluding with T is required to decrypt the original identifiers of the data. Having access to all data does however make it more likely that identities leak, because some stored data has identifying attributes.

3.2 Towards a solution

As seen in the section 3.1, the current way that authorisation is handled poses a SPOF. The solution that we propose should thus aim to split authorisation responsibilities over multiple servers. This however poses a consensus problem in the Single Source of Truth (SSoT) model (depicted in Figure 3.2). Namely, since entries in the list of access rights should be checked by multiple entities, they must be present on all these entities. This means that, to prevent unexpected effects when checking which user has which access rights, the list must always stay synchronised. However, they can not just verify if the access managing entities have the same lists and update them if they differ, because this would indirectly still pose the same SPOF. Next to that, the synchronisation between the two or more access lists can go wrong in many malicious and non-malicious ways, with for instance network instability or attacks on access rule updates, making it a non-trivial problem solve. In the Multiple Versions of Truth (MVoT) model (depicted in Figure 3.3), this consensus problem does not occur.

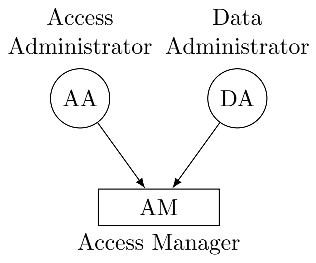

Figure 3.1: Current authorisation

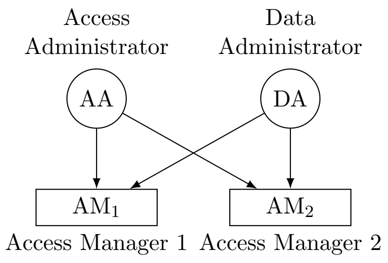

Figure 3.2: SSoT authorisation

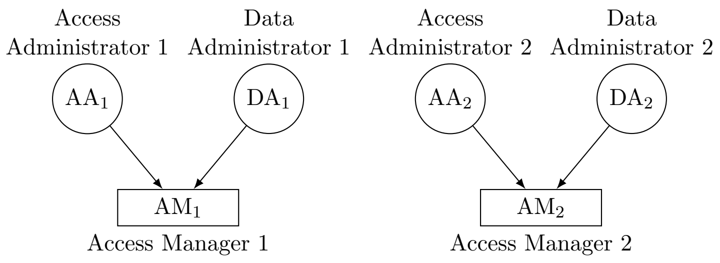

Figure 3.3: MVoT authorisation

Chapter 4

Updating access rules with SSoT

In this chapter, we assume a model with a Single Source of Truth (SSoT). In the case of PEP Responsible Data Sharing Repository (PEP Repo), this would be the Access Administrator (AA). However, this role is generalisable to any system, and is not specific to PEP Repo alone.

The SSoT model is explored first, because the current way PEP Repo (and most other similar systems) works only involves one person that is responsible for updating the access rules that allow certain users to access certain data. The model that assumes Multiple Versions of Truth (MVoT) will be explored in chapter 5.

The solutions proposed in this chapter fit the most common workflow best, and do not require significant changes in terms of human interaction with the system. However, this does mean that the human will still be a Single Point Of Failure (SPOF) in the authorisation process; If the human gets compromised, arbitrary access to the system can be granted. Since our goal is to mitigate a SPOF in all parts of authorisation, we will discuss this version of the problem in more detail in the next chapter. However, since the SPOF reduction is not the only part of the solution that has to be taken into account, we will also explore the solutions that have one SSoT.

4.1 Consensus algorithms like PAXOS and RAFT

One option for reaching the goal of multiple synchronised access lists is the use of existing consensus algorithms. At first glance it seems like the problem has already been solved with systems like PAXOS or RAFT. These are systems that guarantee consensus between multiple nodes under certain conditions [4]. Nodes in this case are the Access Manager (AM)'s, but could theoretically be any processing unit, even virtual ones.

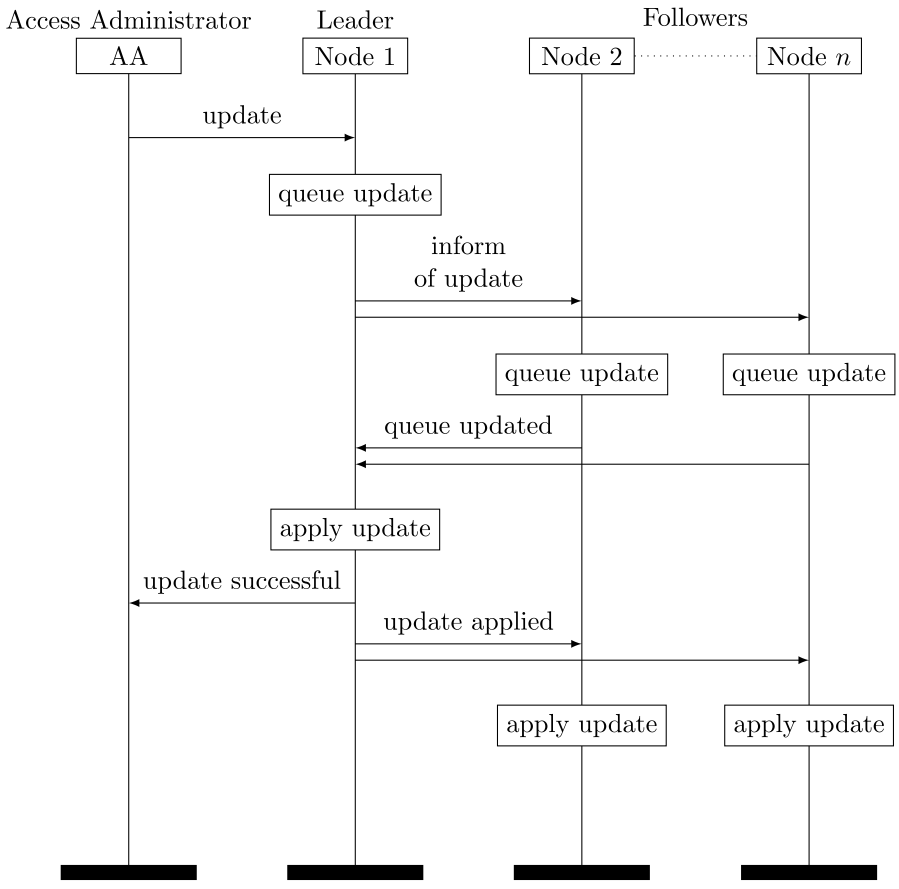

Figure 4.1: graphical representation of a general successful RAFT update

Although many versions exist (see Appendix A), they generally work as shown in Figure 4.1. One node acts as a leader and all the other nodes act as followers[1]. All state changes are sent to the leader. In this case, the state changes are changes to the access rules, sent by the AA. The leader then queues the state change, and informs all other nodes of the state change. Once more than half of the follower nodes has also queued the state change, the leader applies the state change on it's internal state machine. Once a follower learns the leader applied the new state, the follower will also apply the same state change to it's internal state machine. This mechanism ensures consensus over the access rules, by at least half of the nodes in the system[2]. The system applied to the AM's of PEP Repo is shown in Figure 4.2.

[1]Electing this leader is a non-trivial problem, see [4] for more details

[2]It can take some time before the followers also *apply* the change to their state machine, so only eventual consistency is guaranteed

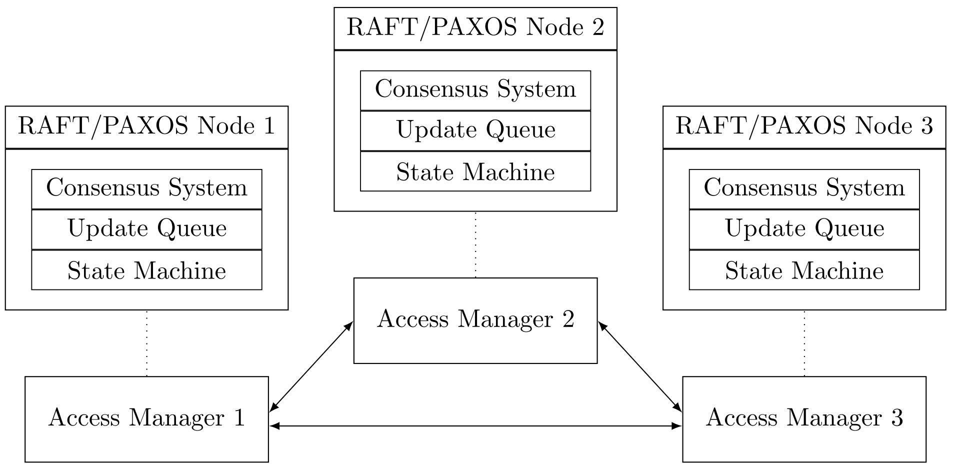

Figure 4.2: Graphical representation PAXOS or RAFT applied to PEP Repo

For security it is important to note that all changes are sent to the current leader of the PAXOS or RAFT system. Without extra measures, a compromised leader could thus change the state on all machines. Therefore, it is required that the AA also signs each state change by using public key cryptography. This ensures that a compromised leader can't simply insert their own state changes, which would be applied to the whole network. The signature has to be checked on each node in the network, so on all AM's in the case of PEP Repo.

Lastly, another important factor to take into account is the usability of the system itself. Even though the RAFT system is easier to understand than PAXOS, they are both still hard to implement, maintain and debug. This should be taken into account in relatively small projects that do not have programmers specialised in distributed systems algorithms, like PEP Repo.

4.2 Two or Three phase commit

One other option would be to directly commit changes of access rules to two servers, in this case the AM and the Transcryptor (T), directly from the AM. This would put more trust on the AA, but as that is a SPOF that is impossible to circumvent in the case of SSoT, this is not a problem.

The steps of this Two Phase Commit (2PC) would be as follows (Also see Figure 4.3). Firstly, the AA sends an update request to both AM and T. These servers then verify if the change is allowed according to their rules, and prepare to commit. Now they both send a message to the AA signifying that they can both commit. The AA now approves the final commit once both servers are prepared to commit, and send this signal to both servers.

They both commit and send a successful reply with the final message of the interaction.

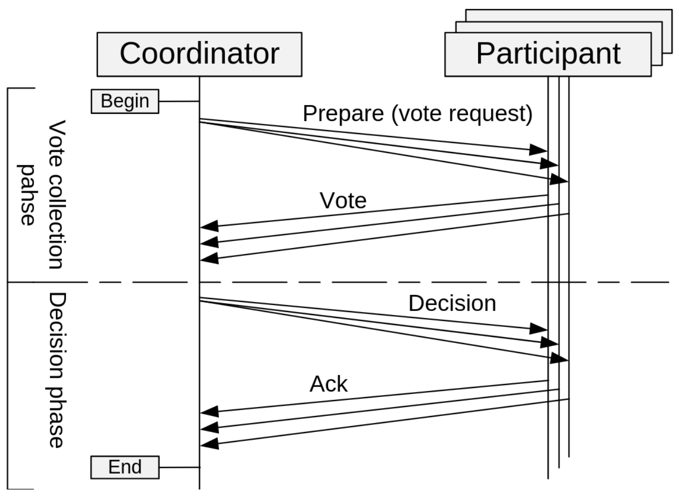

Figure 4.3: Two-phase commit protocol [6]

Adding another phase to this protocol would add certain failure recovery features, such as non-blocking on client/coordinator failure and the possibility for another node to take over the transaction [4]. The same holds for a failure of a participant; Three Phase Commit (3PC) would allow the protocol to continue on the other nodes. This is useful in the case of PEP Repo, because without these failure recovery features it would require manual intervention of the AA if a temporary failure like a temporary network outage occurred.

The problem of nodes not responding due to network failure or otherwise can be solved by using timeouts, recorded on the client/coordinator. Also a number of retries can be done if a network error causes a non-repeating failure. This only works if the protocol is executed linearly or uses a counter system [4].

This system does not automatically recover from all failures, but does notify the AA that something went wrong, and leaves it to the AA to solve the occurred problems manually.

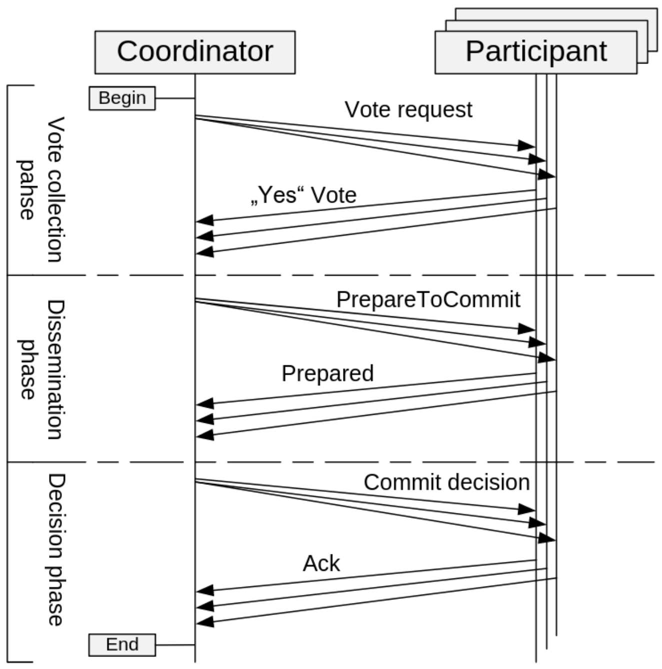

Figure 4.4: Three-phase commit protocol [6]

4.3 Time stamped commit (TSC)

Another option to synchronise the access rules on both AM and T is Time Stamped Commit (TSC), for example by using Lamport timestamps [7]. This only requires communication from the client, the AA in this case, to servers AM and T and back.

The protocol would work as follows: AA sends the request to update an access rule to both the AM and T servers. Next to that, AA also sends a time at which the rules have to go into effect. Under normal operation, both AM and T send an acknowledgement of the change and the protocol is done. If either AM or T does not send this acknowledgement within a certain timeout period, the AA is responsible for rolling back the change on both servers if needed, as it can be the case that a change has followed through correctly and only the acknowledgement message gets lost due to a temporary network error.

One point that has to be taken into account is that synchronising time between servers has been proven to be hard. But maybe solving this problem automatically is not required, and merely detecting it is enough. The goal would be to have reasonably similar times on AA, AM and T. This can be achieved by making AA also send their current time with each transaction, and having AM and T verify whether it is within x seconds of its own time. If due to network error or time difference it is not, the update is not executed and a fail message is sent back to AA. This leaves AA with the responsibility of solve the time difference manually.

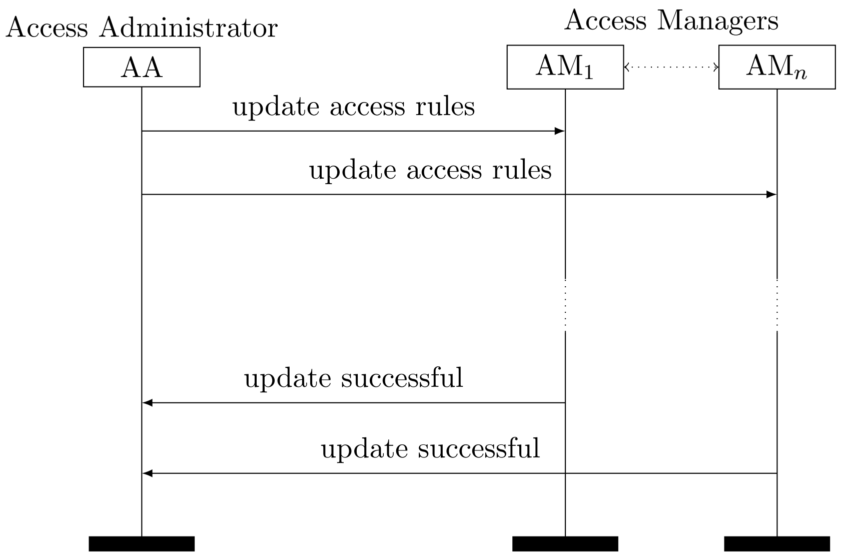

Figure 4.5: Successful flow of Time Stamped Commit (TSC)

Chapter 5

Updating access rules with MVoT

In this chapter, we will discuss the way access rules to the PEP Responsible Data Sharing Repository (PEP Repo) system can be updated in the case of Multiple Versions of Truth (MVoT), also called Multi Party Authorisation (MPA). In the previous chapter, systems with only one Access Administrator (AA) capable of changing the access rules were described. However, while these system are more fitting to the current PEP Repo workflow, they do not realise a true reduction of the number of Single Point Of Failure (SPOF)'s. Namely, the AA is still only one human, and therefore a SPOF. In this chapter, we will discuss solutions to the SPOF problem that also take into account this problem. We do this by changing the model of updating access rules in PEP Repo to a system with multiple AA's, all having to agree on the same access rules before the system will allow access to certain data.

5.1 Split responsibilities

This problem looks a lot simpler than most of the systems described in the previous chapter. It looks like this, because in the MVoT model synchronisation between access lists becomes unnecessary. Where differing access lists in a Single Source of Truth (SSoT) model would mean a breach of the system, in the MVoT model differing access lists could be considered a feature. This is inherently the case, because different AA's might allow different accesses, and a user will only be allowed access to certain data if all of the AA's agree. A simplified successful data retrieval flow is shown in the Figure 5.1, as well as an erroneous flow in Figure 5.2, where one of the AA's has not given access to the data[1].

[1]In reality, the response from the Data Storage also has to go through the Access Manager (AM) servers to the Data Requester

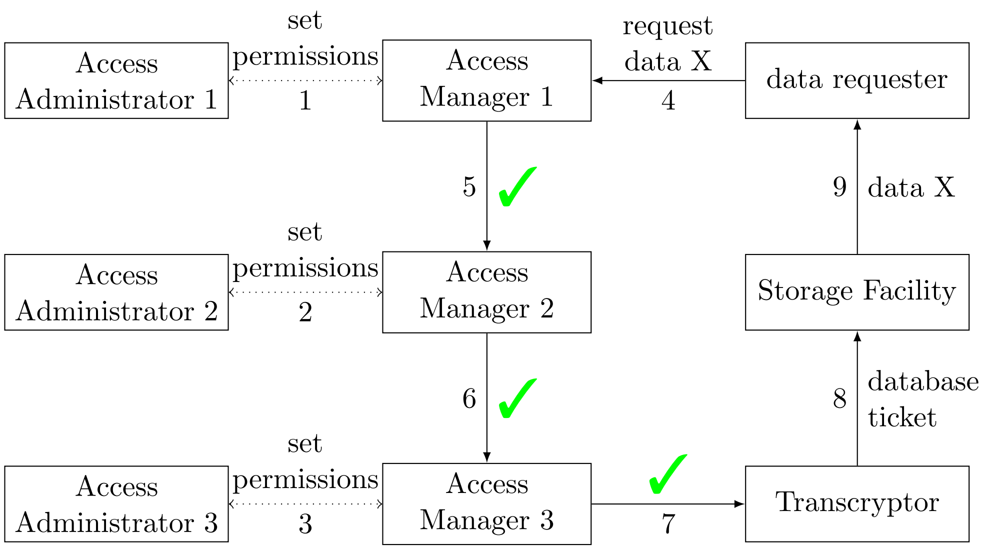

Figure 5.1: graphical representation MVoT successful flow[1]

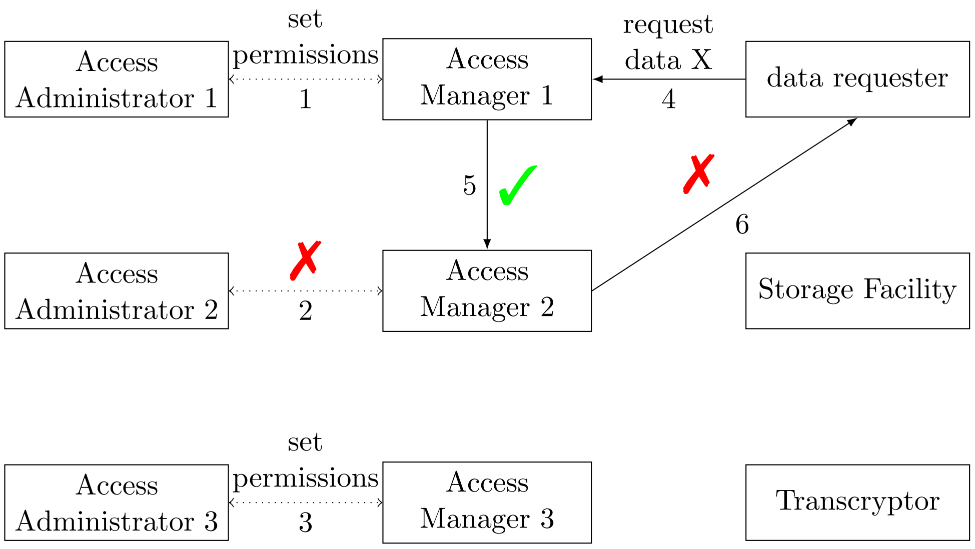

Figure 5.2: graphical representation MVoT erroneous flow[1]

[1]The Transcryptor can also be removed, because the AM's each transcript as well.

Because the AA's are now different entities, they will get differing pseudonyms for the users of which data access is controlled. Without common identifiers, access control is not possible, because there would be no way to know the AA's are giving access to the same user's data.

This problem can be solved as follows. The AM's can both function as an entity applying access control, as well as a Transcryptor (T) that can create local pseudonyms. By using this, the AMs can function as each others T, making the same local pseudonyms available on all AMs in the pipeline. This process is shown in Figure 5.3.

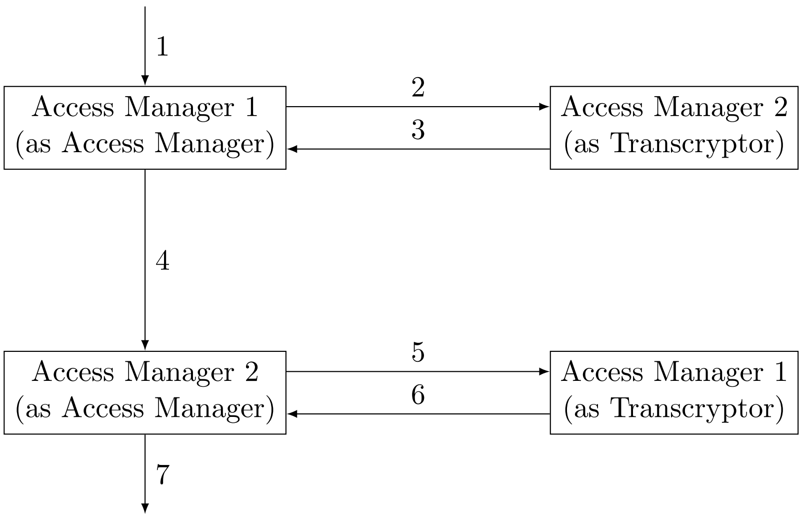
1
Access Manager 1 (as Access Manager)
2
3
Access Manager 2 (as Transcryptor)
4
Access Manager 2 (as Access Manager)
5
6
Access Manager 1 (as Transcryptor)
7

Figure 5.3: Flow of access rules with multiple Access Managers

Chapter 6

Verifying access rules

In the previous chapters we have discussed the process of updating a list of access rules, both in a Single Source of Truth (SSoT) model and in a Multiple Versions of Truth (MVoT) model. However, these systems deviate from the standard singular list, by using multiple entities to store the access control rules, to prevent a Single Point Of Failure (SPOF). This means that we also have to to adapt the way these rules are verified to make use of these multiple access lists.

6.1 Verification in the case of PAXOS or RAFT

PAXOS and RAFT are, granted their difficulty and relative inefficiency, solid systems to reach consensus over the state of a system, in this case the access control list. However, these systems do not come with a mechanism to retrieve data. They only come with varying guarantees about the state of the system, which we will have to take into account while choosing a way to verify the access rules.

6.1.1 Partial verification

As described in a previous chapter about PAXOS and RAFT, the most basic implementations allow f nodes to fail in a system of $2f+1$ nodes. This means the majority must succeed, and contain the most up-to-date access rules. So, given a system of $n \in \{3, 5, 7, \dots\}$ nodes, at least $\frac{n+1}{2}$ nodes contain correct and up-to-date data. This means that when verifying access control, we can at most require $\frac{n+1}{2}$ nodes to have granted access. This number will change if there are other requirements, like byzantine fault tolerance.

Note that the nodes that contain certain access rules do not per se have to be the same subset of nodes for each access rule. A simple example of this is given in Figure 6.1. However, because PAXOS and RAFT provide "eventual consistency", this only has a minor effect on the system in the long term.

A way a system like this can be imagined is similar to an idea used in [8].

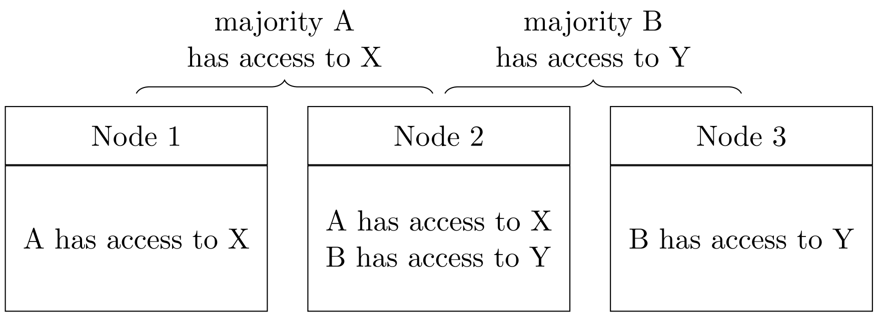

Figure 6.1: Majority nodes with different access rules

This system works as follows. Take physical nodes N, and virtual nodes K with $\|K\| = \|N\|$. On each node $n_i \in N$, distribute key material for virtual nodes $\{k_i, k_{i+1}, \ldots, k_{i+\|N\|-1 \ mod \ \|N\|}\} \in K$. Any two nodes are now always required for decryption.

In the case of a system with three physical nodes $\{AM_1, AM_2, AM_3\}$ and virtual nodes $\{v_1, v_2, v_3\}$, dividing the virtual nodes like $\{v_1, v_2\} = AM_1$, $\{v_2, v_3\} = AM_2$ and $\{v_3, v_1\} = AM_3$ always requires two out of three nodes to verify the access rules, but also allows one node to be unavailable or temporarily out of sync.

An important realisation is that this drops the security of the system down to two points of failure, instead of the potentially more nodes that do authorisation (3 points of failure to 2 points of failure in the example above).

6.1.2 Complete verification

Instead of relying only on part of the nodes in verification, it is also an option to wait until all nodes are up-to-date. This can easily be implemented, as PAXOS and RAFT nodes use strictly increasing version numbers internally to keep track of the current state they are in [4]. These can be compared at the verification stage, and if the version number of previous node AM_{i-1} is larger than current node AM_i, the current node can wait on an update (possibly with a time out).

This does however come with implications for when the state number of $AM_i < AM_{i-1}$. This can happen when an update to the access rules is done while another users' access is being verified. The initial solution might be to simply roll back to the same state of AM_i, and use the older access rules. However, doing this without extra security measures would mean that if any Access Manager (AM) is compromised, they can roll back the access rules to an arbitrary earlier state, compromising availability or potentially allowing old rules to become in effect again.

One way to solve this problem is to use time stamps on the state numbers, and requiring that the states can be no more than x minutes apart. Assuming all AM servers have reasonably synchronised clocks, the access rules can be rolled back at most x minutes in case of a compromise. Historically, a scenario of synchronised clocks among all servers was not always the case in PEP Responsible Data Sharing Repository (PEP Repo). This must be taken into account when evaluating this solution.

6.2 Verifying in other cases

Verifying in the case of manual two or three phase commit updating is relatively simple. In the current implementation of PEP Repo authorisation, the only additional requirement to achieve 2 points of failure is that not only the AM, but also the Transcryptor (T) actively checks the access rules that are synchronised between both servers.
If a user makes a request that is not allowed by either AM or T, they will not be allowed access. Even more, if AM and T do not apply the required cryptography to create the database ticket, the user is not feasibly able to get data from the system. This works, because the responsibility for creating a database ticket was already split over both AM and T, so by this system, the SPOF would be avoided.
This system could easily be adapted to distribute the authorisation responsibilities over more than 2 nodes. We could simply add servers to verify the access list and do a part of the transcription of the data, making it impossible for a single compromised server to hijack the system. This system is shown in its simple form in Figure 6.2.

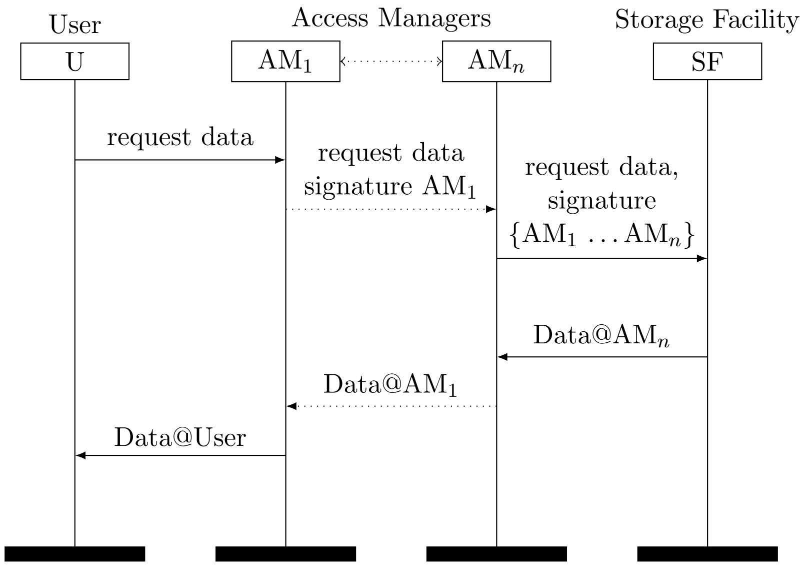

Figure 6.2: Verifying with n Access Managers

Chapter 7

Results

In this chapter, we analyse and compare the solutions that were described in the previous chapters. The metrics that were chosen are reliability, implementability, maintainability, security, network efficiency and cost efficiency. With reliability we mean the automatic recovery capabilities of the system in case a fault occurs when synchronising the access rules between Access Manager (AM)'s. With implementability, we aim to quantify how difficult it is to implement the Multi Server Authorisation (MSA) approach in a system like PEP Responsible Data Sharing Repository (PEP Repo). Similarly, the maintainability metric identifies how hard it is to maintain or debug an MSA approach relative to the other MSA approaches. The security metric shows the security benefit over the current system, in terms of how well it solves the issue of a Single Point Of Failure (SPOF) in authorisation. Network efficiency exposes how efficient the MSA approach is in terms of relative network traffic. Lastly, cost efficiency depicts the costs for the servers that are required to use the MSA approach, relative to the other MSA approaches.

	System					
	Currently	SSoT				MVoT
Metric		PAXOS & RAFT	2PC	3PC	TSC	MPA
Reliability	++	+	−	+/−	−−	++
Implementability	++	−	+	+	+	−−
Maintainability	+	−−	+/−	+/−	−	+
Security	−−	+	+	+	+	++
Network efficiency	++	−−	+/−	−	+	+
Cost efficiency	++	−	+	+	+	+

Table 7.1: System scores by metric of evaluation

7.1 PAXOS and RAFT

In the previous chapters, we have described how you could use consensus algorithms like PAXOS and RAFT to achieve automated consensus. We have shown what the consequences of using such systems are. We have found that automated consensus algorithms are very solid against network problems or certain types of attacks (byzantine faults, see Appendix A). This means that systems like this will need little manual intervention when something goes wrong, or at least don't require immediate attention. So they score high in the goal of achieving reliability.

However, we also found that the more tolerances we require, the more inefficient PAXOS and RAFT become. This has to be put into perspective of the number of faults that would occur in authorisation in PEP Repo. There is unfortunately no data about this, but we assume that the faults that do occur must be low in number, by the successfulness of the current implementation which does not use any fault tolerance.

Also important are the effects on verification using a distributed systems algorithm like these has. They require a specific solution to be found for the potential problem that at any given moment, not all servers might be up to date (as PAXOS and RAFT only provide eventual consistency).

Lastly, the implementation details of systems like PAXOS and RAFT are complex and these systems are hard to maintain. We must take into account that PEP is a relatively small project, and that debugging distributed algorithms is hard. Therefore, PAXOS and RAFT score low on implementability and maintainability.

7.2 Two phase commit (2PC)

The use of a simple system like two phase commit seems nice at first sight. It is easy to implement and it is efficient when everything goes right. When problems like unavailable servers do occur, solving a problem with Two Phase Commit (2PC) could potentially be difficult. However, as we previously concluded that such problems do not occur often or at all, 2PC can be considered very user friendly. When no faults occur 2PC is also very network efficient, as it requires few communication steps. It is however not capable of solving problems such as attacks or network issues on its own, and it is therefore less reliable. The resulting system from using 2PC is of similar security or higher than PAXOS and RAFT, with the same number of nodes, as it does not have build in network redundancy. It can therefore rely on more nodes to verify an access request from a user.

7.3 Three phase commit (3PC)

Similar to using 2PC, Three Phase Commit (3PC) scores high in implementability and maintainability because of the ease with which it can be integrated in the current system. Although slightly less network efficient, 3PC does score higher in reliability, as there is no chance of the system blocking while updating access rules. It also provides a higher chance of the multiple AM's to be synchronised, because of the extra preparation that is done before committing and activating a rule. In terms of security it achieves similar goals to 2PC.

7.4 Time stamped commit (TSC)

In terms of set up, Time Stamped Commit (TSC) is the easiest protocol to implement into the PEP Repo system. It does however require more manual intervention when network problems occur. If for some reason, one of the AM's or Transcryptor (T) become unavailable, for instance due to a connection issue, the only way this problem can be solved is by manual intervention. Since the frequency of such errors is assumed to be low, this might not have to be a big downside. Moreover, if connection from the Access Administrator (AA) to one of these servers is down, there is a high change a normal user is also unable to connect to the server, in the case of network based problems. This would mean the system is down and manual intervention would have been required anyway. The maintainability is therefore considered to be high.

TSC is however less reliable, as one erroneous transmission cannot simply be retried such as in 2PC or 3PC. The chance of the two or more AM's to get out of sync is relatively high, and therefore this system scores lower on reliability. If no problems occur, the system is very efficient as it requires little network communication. Therefore, this system scores well in network efficiency. In terms of security, TSC does perform similar to two- and three phase commit systems.

7.5 Multiple Versions of Truth

For the scenario in which we assume Multiple Versions of Truth (MVoT), there was no requirement for an algorithm to keep the access lists synchronised. This was the case because one of the features of this system is that multiple AA's can apply different access rules, explicitly not having consensus. This makes it a relatively easy option to implement technically, but it does require signifiant changes to the PEP Repo policies for managing access. The main difference would be having multiple AA's that each update the access rules individually.

This system is the most secure, as it not only removes the SPOF present in the technical part of PEP Repo, but also removes it in the policy part. In the Single Source of Truth (SSoT) scenario, there would always be the chance that the single AA get compromised (or makes a mistake), still keeping a SPOF. We found the system with multiple versions of truth would be the most secure, reliable, maintainable and network efficient. The hit it takes on user friendliness is worth the trade off, as it is a system that truly removes all single points of failure in authorisation.

Chapter 8

Reflection

As this thesis was a broad exploration, we did not go in depth, and did not do a cryptographical analysis or implementation of the solutions in terms of the PEP Responsible Data Sharing Repository (PEP Repo). This means that the security claims can be elaborated upon. The general conclusions are however still applicable to systems similar to PEP Repo, which do not per definition use cryptography.

The results are partly based on the assumption that faults do not occur often within the Multi Server Authorisation (MSA) system. Although this seems like a reasonable assumption given the current operation of PEP Repo, for a general system this has to be taken into account when selecting a solution.

Next, a PEP Repo specific requirement for the verification of access rules is that the PEP Repo allows for historical queries. This means that even though someone might not have access to the current version of some data selection, if they did have access at some point, they should keep access to the version of the data they retrieved before. This could be looked into in more depth in future research.

The methodology is also something that should be reflected upon. We decided on a purely theoretical approach. We wanted to explore the breadth of options available to solve the Single Point Of Failure (SPOF) problem in authorisation. Therefore, we first defined criteria that a solution should be scored on. Then, we did a literature review on the applicable methods, and selected those that seemed fitting. However, although we took care to review all available methods by using [4] and other peer reviewed papers as sources, it is possible that other systems were available unbeknown to us. We judged the systems that were found based on the criteria we selected, and found the most fitting systems as a result. Future research could go in on the specifics of the system we found to be most fitting, or look into different ways consensus can be reached between different entities.

Chapter 9

Conclusions

In the current PEP Responsible Data Sharing Repository (PEP Repo), most parts are implemented in twofold, to prevent a Single Point Of Failure (SPOF). Authorisation was found not to be implemented in twofold, meaning the compromise of the single authorisation server would imply the compromise of the whole authorisation system. This is a problem found in many systems with a single Access Administrator (AA), and thus a Single Source of Truth (SSoT) for the access rules.

In this thesis, we explored solutions to the problem of a SPOF in authorisation in the implementation of systems like PEP Repo, to effectively improve the security and resilience of these systems. We evaluated all solutions based on reliability, implementability, maintainability, security, network efficiency and cost efficiency. We described two scenarios, namely one with a SSoT, and one with Multiple Versions of Truth (MVoT). In the SSoT scenario, there is one person that manages the access rules, namely the AA. In the MVoT model, there are multiple AA's that manage the access rules, and they all have to agree on an access rule for it to be in effect. This system is called Multi Party Authorisation (MPA).

Currently, authorisation in PEP Repo operates in a SSoT scenario. However, this puts all the responsibility on a single AA. To fully remove the SPOF, we have considered the MVoT scenario because it does not only move, but actually eliminates the problem of a SPOF. In terms of security and reliability, the MPA solution in a MVoT scenario would be preferable, as it is the most complete solution to the SPOF in authorisation, also taking into account the SPOF in the AA.

However, having multiple AA's is a potential management issue of small projects like PEP Repo, and it is not a given that the MVoT scenario is feasible in practice. Therefore, we have also explored the solutions in the

SSoT model, to maximise the potential of this scenario. As this is mainly a consensus issue among access rules on multiple Access Manager (AM)'s, we explored existing distributed systems algorithms and evaluated them based on aforementioned criteria. Of the systems we explored, Paxos & Raft were found to be too impractical for a project like PEP Repo, as they require too much specialist expertise to implement and maintain. Two Phase Commit (2PC) and Time Stamped Commit (TSC) on the other hand, are too simplistic to provide a reliable solution. We found that Three Phase Commit (3PC) is the best fitting system in the SSoT model, as it is relatively simple to implement, provides good security, and is efficient and sufficiently reliable.

In the end, for a security improvement to be effective, not only the technical solution is important, but also the practical implications have to be taken into account. We have shown that the current way authorisation is implemented in PEP Repo is not sufficient to protect against an attack on an AM, and that mitigating the SPOF is important to improve security in similar systems. Our provided solutions prevent attacks on an AM from causing a major security risk for the system, while also considering practical limitations. With these results, the security of PEP Repo and systems like it can be improved, offering a more secure data storage solution.

Glossary

Access Administrator (AA)

In PEP Repo, the Access Administrator is the role reserved for the person that can update the access rights a user of PEP Repo has. The Access Administrator is responsible for allowing the right usergroups access to the right data stored in the PEP Repo database.

Access Manager (AM)

In PEP Repo, the Access Manager is a Transcryptor that also stores and verifies the access rules a usergroup has to certain data stored in the PEP Repo database.

Data Administrator (DA)

In PEP Repo, the Data Administrator is the role reserved for the person that can define an access context. An access context consists of a relation between a usergroup and a part of the data stored in the PEP Repo database.

General Data Protection Regulation (GDPR)

"This Regulation lays down rules relating to the protection of natural persons with regard to the processing of personal data and rules relating to the free movement of personal data." [1].

Identity Provider (IdP)

in SAML, the Identity Provider is "A kind of service provider that creates, maintains, and manages identity information for principals and provides principal authentication to other service providers within a federation" [5].

Multi Party Authorisation (MPA)

Multi Party Authorisation is the process of managing authorisation within a system with more than one Access Administrator, such as in a Multiple Versions of Truth model.

Multi Server Authorisation (MSA)

Multi Server Authorisation is a system to split the responsibility of authorisation over multiple servers, as to prevent a Single Point Of Failure (SPOF).

Multiple Versions of Truth (MVoT)

In the context of authorisation, Multiple Versions of Truth means that there is more than one definition of the access rules to a system. These rules may differ among versions.

PEP Responsible Data Sharing Repository (PEP Repo)

PEP Responsible Data Sharing Repository is a secure data management system, which relies on polymorphic encryption to allow pseudonymised access to sensitive user data [2].

Security Assertion Markup Language Protocol (SAML)

SAML is a language system used to share data about authentication and authorisation between servers.

Service Provider (SP)

In SAML, a Service Provider is "A role donned by a system entity where the system entity provides services to principals or other system entities" [5].

Single Point Of Failure (SPOF)

A Single Point Of Failure refers to a part of a system that, if the part fails, the whole system will fail.

Single Source of Truth (SSoT)

In the context of authorisation, a Single Source of Truth means that at any point in time, there is a single collection of access rules in effect.

Three Phase Commit (3PC)

In distributed systems, the Three Phase Commit protocol is a system by which multiple nodes can be updated at the same time, by going through a process of three phases.

Time Stamped Commit (TSC)

Time Stamped Commit describes a distributed systems algorithm that synchronises transactions among nodes based on timestamps, for instance by using Lamport timestamps [7].

Transcryptor (T)

In PEP Repo, the Transcryptor is a server that creates a unique pseudonym per usergroup, for each identity stored in the PEP Repo databse. This is similar to the AM, but T does not do access control.

Two Phase Commit (2PC)

In distributed systems, the Two Phase Commit protocol is a system by which multiple nodes can be updated at the same time, by going through a process of two phases.

Bibliography

[1] Regulation (EU) 2016/679 of the European Parliament and of the Council (General Data Protection Regulation), April 2016.

[2] Eric Verheul, Bart Jacobs, Carlo Meijer, Mireille Hildebrandt, and Joeri de Ruiter. Polymorphic encryption and pseudonymisation for personalised healthcare. *Cryptology ePrint Archive*, 2016.

[3] ER Verheul and Bart Jacobs. Polymorphic encryption and pseudonymisation in identity management and medical research. *Institute for Computing and Information Sciences Radboud University Nijmegen*, 2017.

[4] M Kleppmann. *Designing data-intensive applications*. O'Reilly Media, 2017.

[5] Eve Maler Jeff Hodges, Rob Philpott. Glossary for the oasis security assertion markup language (saml) v2.0, 3 2005.

[6] Brahim Ayari. *Perturbation-resilient atomic commit protocols for mobile environments*. PhD thesis, Technische Universität, 2010.

[7] Leslie Lamport. *Time, clocks, and the ordering of events in a distributed system*, page 179–196. Association for Computing Machinery, New York, NY, USA, 2019.

[8] Erik Poll Job Doesburg, Bernard van Gastel. n-pep: Secure data sharing with verifiable distributed pseudonymization. Unpublished, 2024.

[9] Leslie Lamport. Byzantizing paxos by refinement. In David Peleg, editor, *Distributed Computing*, pages 211–224, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[10] Matthias Fitzi. *Generalized communication and security models in Byzantine agreement*. PhD thesis, ETH Zurich, 2002.

[11] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[12] J.-P. Martin and L. Alvisi. Fast byzantine consensus. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 402–411, 2005.

Appendix A

Byzantine fault tolerance in PAXOS & RAFT

One of the caveats is that the basic versions of PAXOS and RAFT do not prevent any form of malicious use, they only guarantee correctness under non-malicious failure (still, other conditions apply) [9]. This means the pure synchronisation of access rules is not the only caveat to be solved. They must also synchronise correctly while dealing with malicious faults. In the context of consensus algorithms these are called "Byzantine faults". These are faults that are caused by loss of connection or maliciously misbehaving servers. Formally, byzantine fault tolerance means the following. If there is a system with a set of nodes N, with honest nodes $H \subset N$, and $F \subset N$ nodes are dishonest or unavailable (with $H \cap F = \emptyset$ and $H \cup F = N$), then the system is byzantine fault tolerant if $h \in H$ can broadcast a message x, all nodes agree on x, and if $h \in F$ broadcasts a message, all nodes agree to discard x [10].

Ultimately, the goal is thus to reach consensus between possibly dishonest servers about the access rules that are in effect.

Secondly, to reach consensus with normal (non-byzantine) fault tolerance and majority voting, one trivially has to set a lower bound of at least $2f + 1$ nodes that are required for tolerance to f failing nodes. This is the case, because for a majority consensus, at least one more than halve of the nodes need to agree on a change of state. A majority is needed, because otherwise for instance a network partition in the system could cause the states of nodes in each partition to diverge from each other, dismissing the promises that PAXOS and RAFT make. This is called split-brain, and is depicted in Figure A.1.

The number of nodes required for a certain amount of fault tolerance only

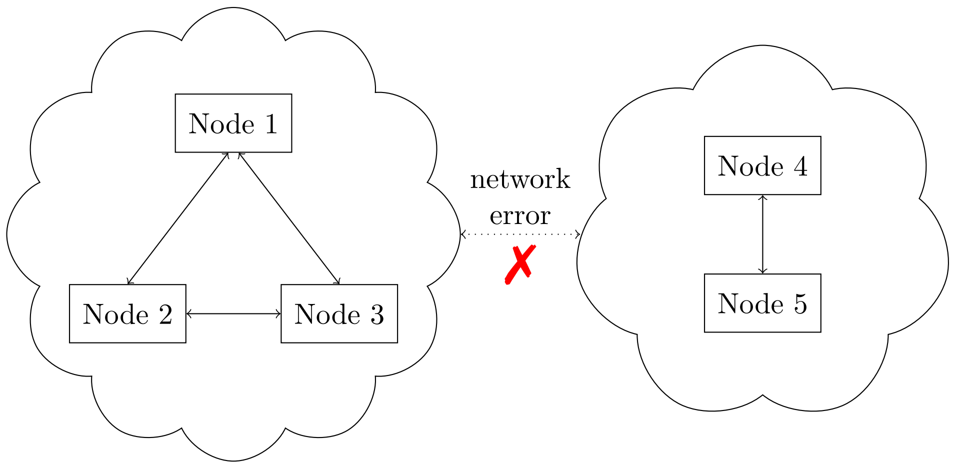

Figure A.1: graphical representation of a split-brain state

increases when we also want to mitigate malicious faults. There are versions of RAFT and PAXOS that do prevent compromised nodes from interfering with the overall system [11]. These are called Byzantine resistant. However, these versions require more overhead to reach consensus. This can be split into two categories, namely network overhead and computing overhead. To tolerate Byzantine faults, it is shown that one of those overheads take a hit [12].

If the network is the main bottleneck, a system would require $5f + 1$ nodes to allow for f compromised nodes [12]. There are versions that allow fewer nodes under normal operation with the same network limitations, but these will be significantly more network intensive upon failure, and cannot revert to the more efficient version once a fault has occurred, even if that fault was not malicious [12].

If the number of nodes is the main bottleneck, the least number of nodes that tolerates Byzantine faults is 4. In this case, only one node can fail without causing a problem in synchronisation between nodes. In the general case, the minimum number of nodes needed for f byzantine faults is $3f + 1$ [11].