JOHANNES KEPLER
UNIVERSITY LINZ

Author
Florian Mader, BSc
k01609425

Submission
Institute of
Networks and Security

Thesis Supervisor
Univ.-Prof. Dr.
René Mayrhofer

Assistant Thesis
Supervisor
Dr. **Michael Roland**

February 2024

Evaluation of technologies to build a trustworthy directory for sensors for an identity management system

Master's Thesis

to confer the academic degree of

Diplom-Ingenieur

in the Master's Program

Computer Science

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

Abstract

Identifying people digitally and securely is already important today and will become increasingly important in the future. For this reason, the JKU launched the Digidow project. Digidow tries to use a distributed system to link digital identities and the people associated with them. Therefore, everyone is expected to manage their own identity on their own devices and interact with sensors which are also distributed. However, such a highly distributed system requires participants to know or discover each other. For this reason, the idea of a sensor directory which is used to find and identify sensors was born. In this thesis, some core requirements are established which are then extended with additional requirements after the threats of the system are analyzed. After the requirements are clear several technologies and their components are analyzed that could solve parts of the sensor directory. It is also shown how those technologies might be used to implement the sensor directory. Finally, those technologies are compared to each other and a group of technologies is shown which could be used to implement the sensor directory.

Kurzfassung

Personen digital und sicher zu identifizieren ist heutzutage bereits wichtig und wird in Zukunft immer bedeutender werden. Aus diesem Grund hat die JKU das Projekt Digidow ins Leben gerufen. Digidow versucht dabei mit Hilfe eines verteilten Systems digitale Identitäten und die dazugehörigen Personen zu verknüpfen. Die Personen sollten dabei, ihre Identitäten auf ihren eigenen Geräten verwalten. Um Personen zu identifizieren, müssen deren Geräte mit Sensoren interagieren, welche ihrerseits ebenfalls verteilt sind. Damit so ein verteiltes System funktioniert, müssen die Teilnehmer allerdings von einander wissen oder eine Möglichkeit haben, sich gegenseitig aufzuspüren. Aus diesem Grund entstand die Idee eines Sensorverzeichnisses, welches benutzt wird, um Sensoren zu finden. In dieser Arbeit werden Anforderungen an ein solches System aufgestellt, diese Anforderungen werden nachdem die Gefahren für ein solches System analysiert wurden erweitert. Nachdem die Anforderungen klar definiert wurden, werden einige Technologien und deren Komponenten inspiziert, welche unter Umständen Potenzial zur Erfüllung der Anforderungen haben. Es wird auch gezeigt, wie diese Technologien in der Lage wären als Sensorverzeichnis eingesetzt zu werden. Zuletzt werden die Technologien miteinander verglichen und eine Gruppe an Technologien vorgestellt, die eventuell eingesetzt werden könnte, um ein Sensorverzeichnis ins Leben zu rufen.

Acknowledgements

This work has been carried out within the scope of Digidow, the Christian Doppler Laboratory for Private Digital Authentication in the Physical World. We gratefully acknowledge financial support by the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development, the Christian Doppler Research Association, 3 Banken IT GmbH, ekey biometric systems GmbH, Kepler Universitätsklinikum GmbH, NXP Semiconductors Austria GmbH & Co KG, and Österreichische Staatsdruckerei GmbH.

Contents

Chapter 1

Introduction

In modern society, there is a constant need to identify people and check for their permissions. This is mainly done using a sensor measuring personal information. However, a sensor alone is not able to verify a person or check if they have any permissions or are denied from any actions. This means each sensor is connected to a database storing this personal information and permissions. This information is stored on many servers controlled by many parties. However, not every party is trustworthy. Therefore the idea emerges that everyone should manage their own personal identities instead. This is the motivation for JKU to start the project Digidow, which is trying to accomplish exactly this goal. But if everyone maintains their personal data, this means the data of all users is distributed via many distinct devices. The devices storing the personal identities of an individual are called personal identity agents, or PIA. But PIAs are not the only highly distributed resources in the system, there are also lots of sensors. Those sensors are required to communicate with PIAs and exchange information with each other. This means there is a huge-scale distributed network consisting of PIAs and sensors that have to discover each other in some way. Therefore in Digidow, there has to exist a system allowing for the discovery of participants in the network. This element is called the sensor directory. The goal of this thesis is to analyze the sensor directory, its requirements, and threats and to find potential technologies able to build it. The sensor directory has to make it possible to add and find sensors securely and privately while everybody knows exactly whom they have to trust. This is done by creating a requirement definition, creating a threat model, deriving more requirements from this threat model, doing literature research on potential technologies, analyzing the technologies if those are a potential solution to the problem of the sensor directory, and comparing those technologies to find those that are best suited to establish the sensor directory with it.

1.1 Digidow

Nowadays sensors measuring fingerprints and other body parts to allow for identification or authentication are indispensable, however, the number of such systems in the future will only rise [85]. The information coming from such systems could further be used to lift restrictions depending on the individual's permissions. To allow for restrictions to be lifted, certain certificates allowing so have to exist. To identify a certain individual a sensor is required, this sensor has to measure parts of the body and send those to a well-known global authority [85]. This authority would then identify the individual and send the data to an entity trying to identify the individual, in Digidow this entity is called verifier [49, 85]. This system makes every physical ID obsolete and could be used on a wide range of applications on a daily basis. Examples of this application are opening doors, as a substitution of a passport for traveling or to verify an individual is old enough to get a drink in a bar.

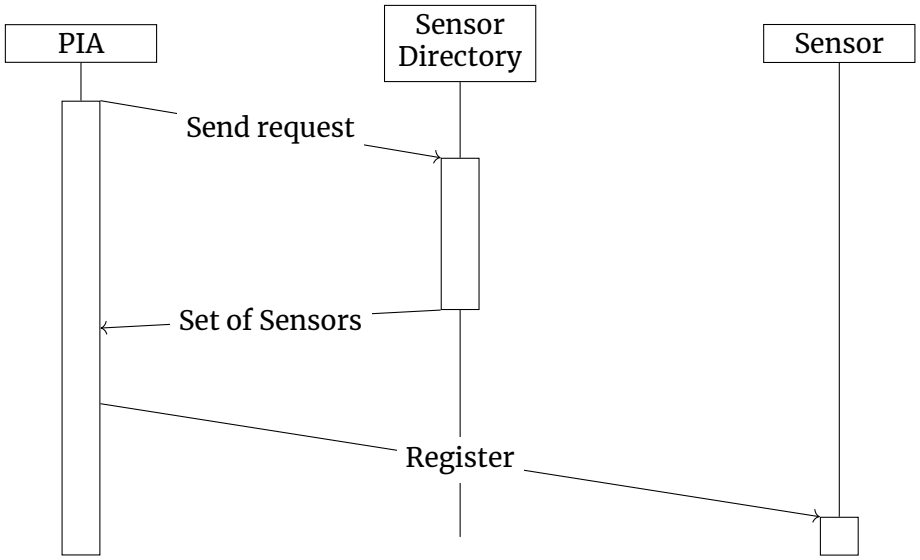

Figure 1.1: Example of the message flow

The process above has one major flaw, there is only one big known global authority that receives every piece of information and therefore knows everything. Digidow tries to prevent exactly this scenario where one party has too much information or power [77, 85]. Such a party would be able to restrict people or track them. Instead of those big global authorities in Digidow, each individual is expected to set up their own personal identity agent [49, 77, 85]. This PIA is the only entity storing the biometric data of the individual as well as their certificates. However, this also means sensors scanning an individual have to send data to this specific PIA. To efficiently solve this problem PIAs have to know the rough location of the individual, they belong to and register at all sensors in that area. This means if a sensor discovers an individual tries to use it, responsible PIAs should be contacted. However this creates the next problem, while the PIA may track the individual to know their location, PIAs do not know all sensors for each possible location. For this reason, a directory holding a set of sensors, which could be queried by PIAs is needed. In Digidow said directory is called the sensor directory. This means a PIA uses the location of the individual to search for sensors in the area. PIAs then register at those sensors and wait. If the individual interacts with the sensor they are notified and are required to identify the user. A flowchart of how sensors could be derived is shown in Figure 1.1.

1.2 Sensor Directory

The sensor directory is an essential part of Digidow because it is the only way to resolve the distributed system and allow PIAs to discover sensors in the system. To allow this, there are some initial requirements the sensor directory has to fulfill:

Each user using the sensor directory should be anonymous. This is the case because one goal of Digidow is to hide personal information and help users to get some of their privacy back. This is only possible if PIAs are anonymous and cannot be identified. If this is not the case it might be possible to match personas

to PIAs and consequently act differently. This is also important to mitigate discrimination and ensure freedom.

Obviously, the directory is expected to store a list of sensors. It is important to know for each sensor which data has to be stored. Each sensor has to include some way to contact the sensor, for this reason, an address has to be stored. Because Digidow is expected to work in the Tor network this address might be but does not have to be an onion address. An onion address can be up to 56 characters long. Also, a description of each sensor needs to be stored. This is important because PIAs might have to act differently on different sensor types. It can be assumed such a description would require some kB in storage. Because the PIAs should be able to query the sensors in a specific area, it is also necessary to store the location of each sensor. To store the location latitude and longitude for each sensor has to be stored. It is important in which precision those values are stored because it might be required to know the exact location for querying later. This information should also fit into a few bytes of storage. Also, a flag is needed if the sensor is currently active and connected to the internet or if this sensor is deprecated or not connected anymore. This flag might be used to show old sensors or sensors that are temporarily unavailable by setting this field to not connected. If this flag is not set this could also represent sensors the creator does not expect to be used anymore. Most likely PIAs are only interested in connected sensors. However, which sensors are not connected might still be valuable information. Such a flag could be represented by a single bit and would not take lots of storage. It is also important to verify the creator of a sensor, therefore the creator or sensor provider is expected to sign the sensor. The signature would also require some storage space depending on the chosen algorithm. Last but not least additional data might be necessary. This data could be trust information or any other useful information to the sensor. However, it is not yet determined which information should be stored additionally. It is assumed the storage required for this additional data would be up to a few MB. All in all, a sensor can be assumed to require storage space somewhat between a few kB up to a few MB.

The sensor directory has to be queriable. This means it should be possible to send a request to the sensor directory and the sensor directory should answer according to the query sent to it. This is important because a system only storing data without a way of retrieving data is useless.

Everyone should be able to query the directory. This means everyone is expected to be able to set up their own PIA and participate in the network. Also, each of those PIAs has to be able to query the sensor directory and gain a list of sensors in the area of the person they are responsible for. So if someone wants to query sensors there might be no authentication information included. This is also important because everyone should be anonymous.

Everyone is expected to add data to the directory. Consequently, the directory should not differ between different sensor providers. This means every company and private person should be allowed to add their sensor to the directory.

Two equal requests should always deliver the same data. In other words, the response should only depend on the data stored and the request received. So if different users send the same request, they should also receive the same response. This may only differ if, in the meantime, a sensor provider adds additional sensors.

The data of the directory should be immutable. This means it should not be possible to change data once included in the directory. This is the case for the provider of the sensor directory, as well as the sensor provider or anyone else. For sure there might be the need to update sensors from time to time, this

might be the case if the sensor is replaced by a new one or because the sensor broke down and is not connected anymore. This still should be possible while maintaining the old data of the sensor. Meaning for each update a new entry has to be created. Therefore, a history of all existing and deprecated sensors is stored.

The data of the directory should be verifiable. It means if a PIA receives a set of sensors from the sensor directory, there should be a way of verifying this entry is included in the directory and also there is no entry missing which is included in the directory.

Everyone should know whom to trust. Meaning for those using the sensor directory, everyone should know who is allowed to see their data, under which condition it is possible for someone to change their data, or which entities are required to verify data included in the directory. This also means it should be clear which combination of users could tamper with which information.

1.3 Trust System

A trust system might be required to know which sensors can be trusted. This system should allow everyone to know which sensor is trustworthy and which sensor is not. There are many ideas on how to implement such a system. One option would be for each sensor to store an additional value that changes over time and shows how trustworthy this sensor is. Someone needs to vote on the trust value for it to change. Those votes could be summarized into a value between 0 and 1 which would be a good estimation for trust. The impact of each vote is important and might change. Due to the need of this data to change, this data cannot be stored within the immutable sensor entry, however it might be an option to store the votes in an immutable list to show who voted for which change. Another option would be to create a static trust value each time a sensor is added to the sensor directory. This means when a sensor is added, a trust figure has to be generated and stored with the sensor. Such a trust value would be generated by the system and would show how trustful sensors were at insertion. An additional example of a trust system would be a system based on a web of trust. This would allow everyone to sign sensors they trust and other users can choose based on this information if they also trust a sensor.

1.4 Objectives

The goal of this thesis is to find a set of technologies that might be able to solve the requirements of the sensor directory. To do this the requirements from this chapter are extended with additional requirements which are concluded from the analysis of the threat model. But already in the basic requirements of this chapter, some problems arise, one of those problems which will be tackled by future work is the immutability combined with the storage capacity. Because the storage should be immutable it would increase indefinitely, this means it might be needed for the system to forget data after some time while still maintaining the trust and all other requirements. Different technologies are then analyzed using the requirements and the threat model. To allow for such a comparison the different requirements get assigned an importance factor. By doing this in the end a set of technologies should be found which are best suited out of the technologies analyzed to implement the sensor directory. Also, parts of the technologies should be found that could be beneficial to the system and

could therefore be migrated into other technologies to increase those to better fit the needs of the sensor directory.

1.5 Outline

In the next chapter, the threat model of the sensor directory is analyzed. This raises additional requirements which are later used to find a suitable technology. After the threat model is analyzed some background information is given, which is used later by several technologies. Afterward, the potential technologies are explained and it is shown how they could be employed to implement the sensor directory. It is also shown for each technology which requirements can be solved and which are not solved. After explaining every technology they are compared with each other by selected figures. Last but not least, an outlook is given for the sensor directory.

Chapter 2

Threat Model

2.1 System

As already explained, the sensor directory is a system that is used to store a list of sensors. Those sensors are provided by anyone by sending them to the sensor directory. The sensor directory then adds the sensor to the storage in an immutable way, this implies the integrity of the data is essential. Everybody should be able to query a location from the sensor directory and the sensor directory is expected to answer with a full set of sensors fulfilling the query. This means confidentiality is not important because everyone can get data. A full set of sensors means each sensor satisfying the request and stored in the directory is part of the response set. This should be verifiable. It should also be confirmable, all sensors in the response are actually stored in the directory. Depending on the technology the sensors are stored on one or multiple nodes. It also depends on the technology in which data is included in communication and how often this is the case. This means if there is only one node the sensors are only communicated with these directories and are only sent from the directory as part of a response set. If there are many nodes these nodes might communicate with each other and exchange sensor information. This means the location of the data can be as follows:

- on the node,
- in transmission to a node,
- in a set in transmission from the directory to the requester,
- at the requester (most likely PIA).

The sensors can be provided by:

- anyone.

The sensors can be queried by:

- anyone.

2.2 Attack Vectors

Obviously, as for each system, also the sensor directory has several attack vectors, which are discussed in the next subsections.

2.2.1 Denial of Service

Denial of Service (DoS) is not one single threat, it is more a combination of many different threats which make a service unavailable. Most of the time this

attack is performed by exhausting all resources a system has available and therefore paralyzing the system, so it is no longer able to respond to legitimate requests. In many cases, this is done by draining the network capabilities or destroying the network in another way [73, 101]. Similarly to this, it is also possible to exhaust the storage capabilities or the computational resources. But the term is also met if an attacker can generate a hardware failure, use a software bug, exhaust any other recourse than the network capability, or use any environmental condition [63, 73, 101]. Such an attack can lead to legitimate packets or requests being lost because the service is occupied by the DoS [73]. If a DoS hits the system the service is not accessible anymore, which means the functions of the system might not work for the duration of the attack. For the sensor directory, this means it might not be possible to request sensor information or to add new sensors to the directory.

By Adding Sensors

In addition to common DoS attacks the sensor directory suffers from the threat of DoS by someone adding a lot of sensors. It is hard to prevent such attempts because adding sensors is a legitimate use of the system. Because everyone should be able to set up sensors and add them to the directory it is not possible to distinguish between someone buying some sensors and adding them at once or pretending to do so and therefore performing an attack. If adding a new sensor requires a lot of resources on the side of the sensor directory, this means adding many sensors even increases the load on the sensor directory. Such a situation might also lead to a lot of network traffic if the sensor directory consists of lots of nodes or a lot of CPU power consumed if there is a lot of calculation required to add sensors. If this attack hits the system, the consequence would be the same as if a normal DoS attack hits the system. The system might not be able to add new sensors or answer queries for the duration of the attack. This attack might be even more problematic because afterward lots of potentially useless sensors are stored in the directory occupying storage resources.

By Requests

Another possibility to attack the sensor directory is to request sensors until the system cannot handle the requests anymore. This attack is similar to the generic DoS attack on the network layer. The impact highly depends on the work required to respond to one request. If lots of calculation is required to respond to a request this might exhaust the whole computational power and therefore make the sensor directory unable to respond. If this attack succeeds the impact on the sensor directory would be the same as in the standard DoS attack, the sensor directory would not be able to add additional sensors or answer queries while the attack is running.

2.2.2 Network Sniffing

For attackers who are able to read network communication, it is possible to receive additional information depending on which part of the communication the attacker is able to sniff. The attacker can position the sniffing tool at the sensor directory or the PIA. This sniffing tool might be an actual hardware tool or it might be a software application installed on a device.

At the Sensor Directory

If the attacker's sniffing tool is located in the network of the sensor directory, the attacker is able to read how often a sensor is requested or how often a specific sensor is requested. The attacker might also receive each sensor added to the sensor directory or be able to sniff parts of the consensus protocol if such a protocol is required. Finally, the attacker might also be able to generate a fingerprint for PIAs and identify them which is explained in section 2.2.3. An attacker could gain some insight into the system and also gain knowledge about the importance of several sensors this way.

At the PIA

If attackers can position their sniffing tool in a way they receive the same packets the PIA receives, the attacker might be able to know which requests the PIA is sending, and also what the set of sensors in the response looks like. This might allow the attacker to locate the user the PIA is responsible for. The attacker might also be able to sniff on the verification step and gain knowledge about the permissions of the user.

At Both Ends

If an attacker can read both ends of the communication, in addition to the information from every single location, the attacker may be able to link the outgoing and incoming messages. This means the attacker can identify which packets are from this specific PIA and also which sensors the PIA requests. This means the attacker is able to detect where the individual the PIA is responsible for moves roughly and it is also possible to identify the PIAs packets. It might also be possible for the attacker to further identify the PIA by analyzing a pattern.

2.2.3 Fingerprinting

Fingerprinting is an approach that allows the identification of a client. To do this the attacker collects as much information as possible. This information might range from hardware information to details about the operating system as well as it might include information about the used software to perform the request [62]. The information collected may vary depending on the application [45]. This information can be seen as the fingerprint of a device. If enough information can be collected these fingerprints are unique for most devices and can be used to identify the device [62]. This might allow for the sensor directory to discover different PIAs. In this case, the PIAs might not be anonymous anymore, and it might be possible to predict actions in a deterministic way.

2.2.4 Target Discovery

It would be possible to use the sensor directory to search for targets and attack them. To do this and receive potential victims an attacker only has to follow the normal use case of the directory and request sensors. If no vulnerable target is found within the response set, the attacker can repeat this attack by retrying their request using new data until a potential target is found. If the attacker

is able to receive some sensors that are vulnerable to some known attacks, the attacker has an easy shot. It is also possible for an attacker to receive further information about the vulnerabilities of a sensor from the description field of the sensor. This attack is quite easy to perform if a user is able to request special attributes from the directory. An example would be if attackers are able to search for specific models of sensors or software versions with known vulnerabilities. When these conditions are met it is easy for attackers to search and find their targets. If this attack is not mitigated many sensors might be attacked if they show any form of vulnerability. This means if this attack vector is not mitigated the threat for sensors increases. However, the security of sensors is out of scope for this thesis. If lots of sensors are attacked the number of malicious sensors in the system rises. Because each sensor could be attacked by new attacks, sensors that are originally trustworthy might get malicious. As a consequence, the trust system would lose its purpose. This leads to a situation where no sensor can be trusted and the whole system is not usable anymore.

2.2.5 Data Collection

The sensor directory can be used by an attacker to collect information and use this information in another attack. This is the case because everyone can request and analyze the system and can try to gain information. An example could be, if the location of a company is known, it is possible to query the sensor directory for this location and get some of their used addresses by analyzing the sensors in the response set. This also means it might be possible to gain information on which companies are using Digidow and also how many sensors are used by which company. If additionally the structural building is known it may also be possible to identify exactly which sensor is used for what. For further information also the type of sensor can be taken into consideration, this means for example a fingerprint sensor next to a door might be used to unlock this specific door. It is also possible to gain information about the usage in a specific area by analyzing the density of the sensors in an area. There might be much more information that can be obtained from the sensor directory, but this may depend on the use case. The threat here is to gain information about the sensor providers other than only their sensors and their addresses. This might be useful to plan further actions against someone or to analyze some behavior. This could be perfectly expanded with social engineering to follow up with some devastating attacks.

2.2.6 Malicious Sensor

Because it should be an option for everyone to set up a sensor and add it to the sensor directory. This also allows an attacker to set up malicious sensors and add them to the sensor directory. Such a sensor can be malicious in different ways, possible examples are a sensor trying to identify the PIA of an individual (section 2.2.16) or tracking people using sensors (section 2.2.15). But even if the sensor is not malicious in such a destructive way, already the existence of such a sensor is a threat because network capacity, computational power, and storage are used.

2.2.7 Compromised Sensors

It is possible for sensors to get compromised by an attacker or for an owner to get malicious and use sensors to attack other members of the system. This leads

to a situation, where sensors that might previously be trusted by many users get malicious and attack other users. As already explained in other attack vectors the existence of malicious sensors is a threat on its own, but those sensors are also able to perform devastating attacks. It can also lead to trust issues where no sensor is trusted and therefore, for the sensor directory to lose its purpose.

2.2.8 Compromised Sensor Directory

It might be possible for parts of the sensor directory or the whole system to get compromised and be controlled by attackers. The impact might depend on the technology and which part of the system is compromised. It might allow an attacker to sniff communication (section 2.2.2), read, or even change data in the sensor directory. If an attacker gains control over parts of the sensor directory this allows them to perform several other attacks quite easily, examples of such attacks are Fingerprinting (section 2.2.3), Target Discovery (section 2.2.4), or Data Collection (section 2.2.5). If the system consists of multiple nodes it is important to know which nodes are able to perform which actions and the impact it has, if they are compromised. The impact highly depends on the technology used to implement the directory and which parts are compromised.

Compromised Majority

For some technologies, it might be possible for everyone to set up a part of the network. In many of those technologies, nodes do have equal power levels which means a node already running for some time has the same power as a newly created one. For those technologies in most cases, there is a crucial number of nodes which has to be honest for the system to work properly, thus it is not possible to manipulate data as long this requirement is met. If attackers are able to control a big part of the system, attackers can control the system and add, modify, or delete data. If everyone is able to set up a part of the network, there might be the possibility of someone setting up as many nodes as needed, until the crucial number is reached and the attackers have control over the system. If not every node has the same permissions in the system this number may change or it is never possible to reach such a situation. It may also be possible to compromise nodes until the crucial number is reached and attackers control the system. If nodes do have different permissions in the system, then the threat for nodes with more permissions to be compromised is higher because the attackers might need them to compromise the sensor directory.

2.2.9 Data Integrity

If an attacker is able to change data in the sensor directory this would lead to inconsistencies and would make the system untrustworthy. This would destroy the purpose of the directory. Also, if this is not discovered, attackers are able to manipulate response sets. For example, attackers could send sensors with a specific address to everyone and create a DDoS attack because all of those PIAs would connect to this address and register. Attackers would also have the possibility to send sensors they control to PIAs and gain information about those PIAs this way. The providers of the sensor directory may also be malicious and try to perform such actions. If this is possible, the sensor directory is not trustworthy, and the whole system would break down under this problem.

2.2.10 Manipulation of Data on the Wire

If attackers can change packets sent or received by the sensor directory this allows them to manipulate sensor information or the corresponding metadata in the packet. This allows them for example to send sensors with a specific address to PIAs and create situations where many PIAs are connecting to a specific address and creating a DDoS. Also, it would be a possibility for attackers to get specific PIAs to connect to sensors controlled by them. This is not only an option for attackers but also for the owners of the directory themselves. This means the system could send data not included in the system or data which was not requested. If this is possible the system is not trustworthy.

2.2.11 Data Validity

Because everyone should be able to add sensors into the directory, there is also the potential for attackers to add wrong data into the sensor directory. Also, every sensor should be allowed in the directory and therefore it is not possible to identify wrong data. This is also the case since every sensor should be allowed and therefore unknown information can potentially be valid. If it is possible for attackers to add data without verification, each included piece of data uses up some storage. Also, if attackers are able to do this without any cost they can use this until there is no storage left and the sensor directory is not able to accept valid requests anymore. This would lead to a DoS attack. The problem is, that adding data to the directory is a normal use case, and everybody is expected to do this. Therefore, it is very hard to identify wrong data. Some data might be verifiable, an example is if the sensor is connected to the internet or if the address is valid. Even if the address of the sensor is validated by connecting to the sensor, the attackers are able to wait until this has happened and disconnect the sensor afterward. This compromises the validation in the first place and the wrong data is included in the directory anyway. An attacker only needs one sensor or any device that pretends to be one to generate multiple wrong data into the sensor directory.

2.2.12 Address Problems

Many threats arise from the address field of the sensors. Sensors should be planned to work in the Tor network, but if the address is a Tor onion address, an IP address, or any other address, should not make any difference for the threats. Because everyone can add sensors, it is possible somebody who is not the owner of an address, could add a sensor with this address. Therefore it is important if the address field in the sensor directory is unique or not.

Override Sensor

When a user adds a new sensor to the directory, there is the possibility of the address being taken by an already existing sensor. If this happens, there are multiple ways the directory could react, on one hand, it is possible to drop the already included sensor and replace it with the new one. This approach comes with a lot of problems because everyone could try to drop legit sensors from the directory, even if the user is not the owner of the sensor. This would be catastrophic if everyone is able to drop sensors from anybody else. It would also be possible to decline the newly added sensor, but this could lead to the next threat

where someone anticipates addresses. Last but not least, it is possible to allow multiple sensors with the same address, this would solve this threat, but would allow for different attacks.

Anticipate Address

If the address field is unique, it is possible to deny other users from adding data to the directory. This is the case if a new sensor is not allowed if an already known, active sensor with the same address already exists. If this is the case, attackers who know the address space of a victim can try to anticipate sensors that will be added by the victim in the future. The attacker can add data to those addresses before the victim can do so. Because an active sensor is included in the directory the legit user would not be able to add sensors in their own address space. If this attack is not mitigated and attackers are able to find out the addresses of targets, the directory would be useless. This is due to the fact an attacker could prevent legitimate users from using the sensor directory.

Same Address

If it is possible to add multiple sensors with the same address, this allows attackers to add a huge amount of sensors with the same address into the system. These newly added sensors can be used to attack any address and would have distinct locations assigned in the sensor directory. This leads to many PIAs discovering them and trying to connect and register at this address. Because those sensors all have the same address, all PIAs connect to the same address and generate network traffic. The address used does not have to be of an actual sensor in the system, it rather has to be in the same address space used for sensors in the sensor directory. PIAs receive the address in the sensor directory and connect there regardless if it is an actual sensor or any other system. If an attacker adds enough sensors, this leads to enough PIAs discovering them and therefore connecting to the address, which leads to a distributed DoS (DDoS) attack. Because those sensors are discovered by lots of PIAs over an extended period such an attack might persist for a long time. So, if this threat is not mitigated it is possible to deny the service of any sensor included in the directory. This would mean attackers can prevent sensor providers they do not like from participating in the system, which would destroy the purpose of the directory and therefore make it useless. Even more so, if this threat is not mitigated it could be used to attack any service using the same address space used by the sensors, this means the sensor directory would be a threat to all other systems in the same namespace.

2.2.13 Sensor Trust Problems

To prevent certain threats and unnecessary traffic a PIA might choose to connect only to trusted sensors. Therefore information providing trust has to be stored alongside sensors. This information could be a value between 0 and 1 and each PIA is expected to choose its own threshold for sensors the PIA is willing to communicate with. There are different approaches for such a trust value to be generated, the trust could change over time depending on votes from the sensors users, it also be possible for the sensor directory to calculate this value. Another approach would be users signing sensors and therefore showing they trust a specific sensor. All options do have threats shown in the next sections.

Destroy Trust

If there is a trust value that can change over time, it is important to know who can interact with this value and change it in any way. This is important because it might be the case that if the trust of a sensor is below a certain threshold PIAs are no longer communicating with this sensor, this means, that if someone can change the trust value to any value, this person can exclude sensors from getting used. This is a problem because valid participants of the system are excluded from participating. So if there is a trust value that can be changed it is important to know who is able to do so. Because PIAs are the only entity communicating with sensors, only those PIAs can observe if a sensor is trustworthy or if it is not. This means only those PIAs could potentially change the trust in a sensor. Due to the fact everyone is able to create a PIA and participate in the system and those PIAs are not interchangeable with any real-life person, it is also possible to create multiple PIAs if necessary. Additionally, because all PIAs are anonymous, no one is responsible for the actions of a PIA. This means if a PIA acts malicious, and gets punished for it, the owner can just drop this PIA and create a new, fresh, unencumbered one. Due to the fact PIAs are most likely those changing the trust value of sensors but no one is responsible for their actions, anyone can change trust in any way. Because everyone can create unlimited PIAs it follows it is possible to change trust for any amount chosen. This is the case because the weight of one vote no longer matters if someone is able to vote indefinitely often. Even if one PIA can only influence the value slightly. This means malicious users can exclude any sensor from being selected by PIAs. But this also means if a malicious user has a malicious sensor, and this sensor gets a bad reputation from other PIAs, the user can positively change the reputation and make the whole trust system worthless. On the other hand, there is the option of allowing changes of trust only if the voting device is identifiable or kept responsible. If this is the case additionally to who can change the trust value also the weight of each vote is important. This means if the device is identifiable and therefore not anonymous everyone can only create one instance and use it to influence the trust once. The amount of change might be influenced by the number of votes available for this sensor. However, because those devices have to be known those devices cannot be PIAs because they are anonymous.

Outdated Trust

It would be an option to create trust values when a sensor is included in the sensor directory, this might take into account how many sensors a user has or if this user is known to perform any malicious actions. Maybe also the type of sensor or the location might be important for such a trust system. But this also means it is hard to set a trust value for new users who are not known yet. It also means sensors that are once included with a positive reputation cannot be flagged as malicious afterward. This is also the case if the reputation is built like a web of trust where users sign the sensors they trust. If a sensor gets malicious afterwards the trust cannot be revoked unless specifically possible by the system. It might also be possible for such signatures to be only valid for some time. This situation where trust data is not up to date might also appear in several other trust approaches if trust is not checked regularly. If this is the case this means if anyone can set up or take over a sensor with a good reputation, this sensor can take whatever malicious action they like because there is no punishment for it. If this is the case the whole reputation system itself would be useless. This is because once a specific trust is reached there is no punishment, and the sensors can perform whatever action they want to.

2.2.14 People Tracking (Sensor Directory)

If the sensor directory is able to link several requests to each other or a PIA, the sensor directory is able to recreate the path a person took. It might also be possible for the sensor directory to locate people if necessary. If the sensor directory gets a request for location x and after some minutes for location y which is in the same street or connected in any other way and it is possible to identify these requests belong together, then it is possible to infer that a person is going from point x to point y. Furthermore, if additional data is available it might be possible to identify a PIA.

2.2.15 People Tracking (Sensors)

An attacker can use the sensor directory to track the movement of a crowd or single people. This could be done by adding lots of sensors that are evenly distributed throughout the map. If there is a crowd of people the attacker might be able to estimate how many people are in a specific area, and where the majority is located. If this analysis is done over time the attacker might also be able to identify a direction in which the crowd is moving if there is one. If attackers are able to connect multiple requests, they are also able to analyze the movement of single persons with this technique. If it is possible to follow one person it is possible to identify the location, direction, and speed of this individual. This could be done by reconstructing the registrations of the PIA at sensors. This attack might at first not look as impactful as others, but attackers might be able to gain a lot of knowledge from this. Maybe attackers want to gain knowledge of a location, and when it is less crowded because they want to rob a bank. Or they want to know when there are big crowds or how those crowds are moving for a terrorist attack. But also movement profiles of single people might be interesting and can be recreated if the registrations can be linked. If it is possible to identify the person additionally, the way a specific person took can be reconstructed. But this could also be used for educational purposes to identify peak times at shopping centers or identify over time at which location people are most crowded.

2.2.16 PIA Identification

Let's say attackers want to attack a person they know, they know where they are working or where they are living. The attackers could set up sensors in that area and add them to the sensor directory. After some time, the PIA of the victims might discover the sensors in the sensor directory and connect to it. If the attackers have additional data, like the location of the victim by using other attacks like social engineering, it is easy to identify the PIA of the victim. This is the case because the time the PIA registers and the time the person is around the sensor can be compared. Now attackers know which PIA is owned by the target and can attack this PIA. An example of an easy attack would be to DoS the PIA and therefore make Digidow useless for the target. This means the victim might not be identifiable by any sensor because their PIA might not register there or cannot identify them. While this attack is only possible with additional information like the location of the target or their routine, the impact of this might be catastrophic because highly valuable targets could be selected and attacked easily. But also every other user who has enemies could get an easy target if the attacker has enough information. It might be even easier if those people are people of public interest, such as politicians or celebrities. This attack gets

even worse with the evolution of social media where everyone constantly posts information and therefore the PIA gets identified even easier.

2.2.17 Split Views

The sensor directory could show different sets of sensors to different users even if those send the same query. This also means the sensor directory could not send all data that would be valid by the request. Because one of the requirements for the data is to be immutable, this means if a PIA once receives data this PIA has to obtain a modified version of this data each time. If the PIA receives another version, it can recognize these changes. This means it is only possible to perform this attack if it can identify a PIA or group of PIAs. It also means the sensor directory has to store multiple versions of the data, the sensor directory has to store the data once for each split view. If the sensor directory is able to perform such an attack it is possible for different users to receive different response sets for the same request. This might prevent users from finding some sensors even if those are available. It might also be possible for the sensor directory to show additional sensors for certain users.

Accountability

It might be a possibility for sensor providers to add sensors to the sensor directory and the sensor directory only pretends to add the sensor. If the sensor providers themselves request the data the correct sensors are shown, if it is any other user this is not the case. This means the sensor directory might be able not to add sensors to the directory.

2.3 Selected Attack Vectors

Because there are many attack vectors and not each of them impacts the sensor directory in the same way, only some of them are selected for further discussion, and mitigations for those attack vectors are selected. The selected attack vectors are the following:

- DoS by adding sensors to the directory (2.2.1),
- Fingerprinting (2.2.3),
- Target discovery (2.2.4),
- Data gathering (2.2.5),
- Malicious sensors are set up (2.2.6),
- Compromised sensors (2.2.7),
- Data integrity (2.2.9),
- Data validity (2.2.11),
- Override sensor (2.2.12),
- Preregister expected sensors (2.2.12),
- Insert many sensors with the same address (2.2.12),
- Destroy the reputation of sensors (2.2.13),

- Reputation being outdated (2.2.13),
- Data gathering (2.2.5),
- People tracking (sensor directory) (2.2.14),
- People tracking (sensors) (2.2.15),
- Sensor directory shows different sets (2.2.17).

First of all, only those attack vectors directly concerning the sensor directory are selected. Because a generic DoS attack might hit each system, it is not specific to the sensor directory. On the other side, someone adding lots of sensors is specific for the sensor directory and might add lots of data to the sensor directory, therefore this attack is part of the investigation. Because sending lots of requests is not much different from the generic attack vector this attack is not investigated further.

Someone who can bypass encryption, as well as install a network sniffer (software or hardware) in any network, might be an insider and therefore has lots of information. The information such a sniffing attack gains is the same information, the PIA and sensor directory have, which means such an attacker could also use their privileges and attack those entities. Because there is no additional information leaked and the commitment of such an attacker would be huge this attack vector is ignored. The same holds if someone can manipulate data on the wire. This user is again already in a position where each piece of information is available to this attacker. Also, an attacker in such a position is able to mimic any device in a network. This attacker can manipulate requests or responses and therefore already possesses all privileges. Such attackers have to be repelled before they are in such a position and the sensor directory is not able to do this. However, manipulating data on the wire should not be possible if all messages from and to the sensor directory are signed.

Because the possibility of fingerprinting does enable lots of additional attack vectors, this attack vector is selected to be mitigated. Also, target discovery and data collection are selected. This is the case since for those attacks the sensor directory is used. Also, those attacks can only be mitigated by the sensor directory and if this is not done other attacks might be possible. Also, data integrity is one of the core requirements of the system, this means if someone tries to change data this has to be mitigated.

If the sensor directory includes lots of malicious sensors, it is useless. This means malicious sensors that are created and sensors being compromised are attack vectors that have to be mitigated. This is especially important because such malicious or wrong data requires lots of disk space. This means also wrong and useless data should be prevented as successfully as possible. A field that can be verified and also creates lots of problems is the address field. Also, the impact of those attack vectors is enormous and therefore all of those attack vectors are selected to be mitigated.

However everyone should be able to add an arbitrary number of sensors to the directory, however those sensors could be used to analyze the location of people. While this is generally an attack regarding Digidow and not the sensor directory it is still selected to be mitigated. A quite different attack is if the sensor directory is able to track a person, this is an attack done by the sensor directory which is the reason why it has to be mitigated. Also identifying a PIA and creating a link to a person is a threat regarding Digidow and not the sensor directory, but because the sensor directory is not involved in this attack, it is not selected.

Taking over parts of the system and compromising it is an option for every system because this is highly dependent on the chosen technology. Those tech-

nologies have to mitigate this attack somehow due to the fact every system using this technology suffers from such an attack vector, therefore this attack vector is not chosen. Also if an attacker is powerful enough to perform such an attack it is not possible to mitigate it.

Because there might be a trust system needed and tampering with the trust in sensors might have a huge impact, attack vectors regarding trust are selected. Last but not least, if the sensor directory can show different views to different users this would mean no response could be trusted and the sensor directory is useless. For this reason, this attack has to be mitigated.

2.4 Attack Vector Mitigation

In the next sections mitigations for the selected attack vectors from section 2.3 are explained and rated in different categories. Each attack vector mitigation is rated on the following categories:

1. Effectiveness
2. Impact on functionality
3. Impact on usability
4. Impact on performance
5. Implementation complexity

Effectiveness will be rated as low, moderate, or high and shows how effective this mitigation is. Impact on functionality, usability, and performance will also range from low to high. But if the impact is high this is bad while it is good for the effectiveness to get a high value. Impact on functionality describes if the functionality of the sensor directory would be decreased by the mitigation. Impact on usability describes how much the user experience is impacted by the mitigation. Impact on performance describes how much the performance decreases, most of the time this also describes how much spending increases. Implementation complexity describes how hard it is to implement the mitigation and is also rated as low, moderate, or high.

2.4.1 Transaction Cost

This attack vector mitigation is able to mitigate:

- DoS by adding sensors (2.2.1),
- Malicious sensor (2.2.6),
- Same address (2.2.12),
- People tracking (sensors) (2.2.15)

Attackers are only able to send a lot of new sensors because they do not have any costs to do so. If there is some cost to adding data into the system this does increase security because it makes it harder for attackers to add data, however, it is always an option for attackers to solve such a problem by throwing money on the problem. In several distributed ledger technologies this threat is tackled by adding real costs in the form of crypto coins to transactions. However such costs are only viable for cryptocurrencies.

Another option used by distributed ledgers is by adding a nonce field to each transaction. This field is used to solve a proof of work (PoW) condition. Proof of work means the transaction is hashed with a hash algorithm and a specific number of leading characters have to be zeros, because the hash cannot be guessed it is a computationally difficult task and can only be solved by trial and error by changing the nonce of the transaction [65]. This PoW can be made quite easy just to add costs to transactions. While a blockchain using PoW as a consent algorithm must have a hard cryptographic puzzle, for this use case it is sufficient if only a few of the leading characters have to be 0 [65]. See section 3.2.3 for more information on PoW.

This means when a sensor is added, the sensor directory has to check the PoW first, which should be easy because only one hash function has to be calculated. This should be done first so it cannot be used by attackers to DoS the system because there is no computational power left in the system. This mitigation method is only moderately effective because if an attacker has enough computational power attacks can still be performed [65]. The impact on the functionality is not very high, this is the case because the PoW is quite easy to solve and therefore fast [65]. The usability is not restricted by this mitigation because valid users only have to calculate an easy PoW value. From the point of view of one device the performance is only impacted moderately, this is the case because such a PoW is quite easy and can be solved by an IoT device without any issue [65]. The sensor directory only has to solve one hash so the performance is not limited on this side as well [65]. The actual effectiveness of such an approach for this application has to be tested due to the expected asymmetry in calculation power between an actual user and potential attackers. However such tests are part of future work because the mitigation techniques are not yet finalized.

- Effectiveness: moderate
- Impact on functionality: low
- Impact on usability: low
- Impact on performance: moderate
- Implementation complexity: moderate

2.4.2 Sensor Verification

This attack vector mitigation is able to mitigate:

- Malicious sensor (2.2.6),
- Override sensor (2.2.12),
- Anticipate address (2.2.12),
- Same address (2.2.12)

It might be needed to verify entries before they are included in the sensor directory. The data that is verifiable is the address of a sensor, the connection status, and the signature of each sensor. The location of a sensor and the sensor descriptor cannot be validated. Depending on the address used it can be validated by different approaches or challenges. First of all, it is possible for the users to receive a token that they are expected to provide on the address, by doing so they verify, they do have control over the address [35]. Because Tor addresses are public keys it is possible to verify the user is the owner of the address by verifying the owner holds the private key. Doing so is possible by providing a self-signed TLS certificate containing the received challenge on the

address [86, 99]. This proves the user holds the private key and therefore is the owner of the address. The connection status can be checked by connecting to the sensor and checking if the response is valid. It is important which device performs this check, if many devices are performing this simultaneously this may lead to a DoS. Verifying these values uses some computational power, this means if many sensors have to be validated in a short period of time this adds up and takes much computational power and therefore potentially leads to a DoS. This means the impact on the performance might be high. The complexity may depend on the verification methods used, the signature may be verified quite easily. The connection status can also be checked easily by connecting to the device. How easy it is to check the address highly depends on the address space used. The implementation complexity is low. This is the case because connecting to addresses and validating signatures is easy. It might be possible for an attacker to create a sensor or a device that pretends to be one and add it to the directory. After the device is accepted the attacker may choose to disable this device. This may add wrong data into the sensor directory while requiring a quite high cost for the attacker. This is the case because the attacker is required to own the address for each sensor added. If the data is verified the attacks above are not possible or are very hard, which means the mitigation is quite effective and is assigned high as a value.

- Effectiveness: high
- Impact on functionality: low
- Impact on usability: low
- Impact on performance: high
- Implementation complexity: low

2.4.3 Notaries

This attack vector mitigation is able to mitigate:

- Malicious sensor (2.2.6),
- Override sensor (2.2.12),
- Anticipate address (2.2.12),
- Same address (2.2.12)

There is the option to require one or multiple signatures of known entities, called notaries before a sensor is added to the sensor directory. These notaries could be expected to verify the data included in the data set before signing it [12, 64, 95]. This means the Notaries could verify the connection and address field of the sensor before signing. The sensor directory would know this data is already checked by checking if a signature is available. In general, notaries are checking the data before it is included in the sensor directory. Users have to trust those notaries because they are known and responsible for their actions [64, 95]. Notaries validating data is very efficient while mitigating the attack vectors above [64, 95]. This is the case because only verified sensors are able to end up in the sensor directory. Even if a sensor ends up in the sensor directory by using a malicious notary, PIAs would be able to verify the signature of trusted notaries before they connect to those sensors and therefore know if the data is trustworthy [64, 95]. The impact on usability is moderate, this is the case because everyone who wants to add data to the directory needs to reach out to a notary for their signature first [64, 95]. This could be made easier if notaries are

part of the system and after they sign the data directly forward the data to the sensor directory. The impact on the performance is low, the sensor directory only has to check if a signature is available. Also, PIAs might choose which notaries they trust. The problem is, in addition to the sensor directory notaries are required who are willing to participate and sign sensors. Those notaries have to be trustworthy because they are known or because they are checked regularly. The functionality is not impacted by this. This mitigation is quite hard to implement because there are several new devices needed and those have their own threat model which has to be considered. Also, communication and verification have to be considered. Because this is the case the implementation complexity is high.

- Effectiveness: high
- Impact on functionality: low
- Impact on usability: moderate
- Impact on performance: low
- Implementation complexity: high

2.4.4 Redundant Sensor Directories

This attack vector mitigation is able to mitigate:

- DoS by adding sensors (2.2.1),
- Fingerprinting (2.2.3),
- People tracking (sensor directory) (2.2.14)

Multiple sensor directories can exist next to each other. This would decrease the performance if someone wants to add sensors to all directories because this has to be done multiple times. If this is not done, PIAs have to search in multiple sensor directories to get an absolute impression of the area around the user, this means the performance is decreased for those adding sensors or for those searching for sensors. Also, the usability decreases because no one is able to know if all sensor directories are synchronized or if there are sensors only available in some directories. Nevertheless, the effectiveness of this mitigation might be very high because there is a redundant system. This allows for better fail-safe and also makes it harder for the directory to gain knowledge about users [60]. This is the case because PIAs might choose to distribute their requests and therefore one directory only gets a subsection of information. For example, if a directory is only used once every few hours it is hard to track a person efficiently because there is just too much data missing. The complexity of implementing multiple sensor directories should be low because the software can just be started a second time. Of course, the hardware has to be set up a second time to allow for redundant systems. Also, the redundant directories should be run by different parties known not to collude, otherwise creating a redundant system is worthless.

- Effectiveness: high
- Impact on functionality: low
- Impact on usability: moderate
- Impact on performance: moderate
- Implementation complexity: low

2.4.5 Cache Sensors

This attack vector mitigation is able to mitigate:

- DoS by adding sensors (2.2.1),
- Fingerprinting (2.2.3),
- People tracking (sensor directory) (2.2.14),
- People tracking (sensors) (2.2.15)

PIAs should be able to cache sensors they are using regularly or were using shortly. This not only relieves the sensor directory from stress but also makes PIAs more independent from the sensor directory [8, 69]. This increased independence may lead to a situation where an affected sensor directory does not impact PIAs in the same way it would if the PIAs fully rely on the sensor directory [8, 69]. This situation allows for a lower impact of a DoS attack on the sensor directory [8, 69]. If PIAs do not have to request each sensor every time it also makes it much harder to create a fingerprint for those PIAs. The same is applicable if the sensor directory tries to track a specific user. If the PIA does not have to request as often this means the sensor directory has to work with incomplete data. Of course, fingerprinting and locating persons is still an option but it gets much harder, which means the effectiveness is only moderate and highly depends on the update times chosen. If the sensor directory goes down for any reason, this also allows the PIA only to use the cached sensors. The performance of the system will not decrease, it is more likely to increase because of caching, due to the fact some communication is no longer needed. The impact on usability is low. The impact on functionality highly depends on the update times chosen for the cache. If the refresh times are chosen low the effectiveness of this countermeasure is much lower. If the time to live is chosen high it is possible for PIAs to rely on deprecated data. However, it can be assumed that sensors not to be updated each day and therefore the impact on functionality is moderate. Due to the fact caching only means data is stored on the device the complexity of implementation is low. PIAs only have to store either the last response so they do not have to send a new query until the person leaves the area, or they store the most used location so they are able to skip requesting those sensors. It might even be possible to cache entire parts of cities and therefore go incognito for the sensor directory while inside this location. PIAs might only cache sensors they trust to decrease storage requirements because they would not use them anyway.

- Effectiveness: moderate
- Impact on functionality: moderate
- Impact on usability: low
- Impact on performance: low
- Implementation complexity: low

2.4.6 Distributed Sensor Directory

This attack vector mitigation is able to mitigate:

- Fingerprinting (2.2.3),
- People tracking (sensor directory) (2.2.14),
- Split views (2.2.17)

If the sensor directory consists of multiple nodes, it is possible for users to add sensors at different nodes and also to request data at different nodes. The nodes of the system have to be synchronized securely and should be controlled by different parties. This is the case so one party does not accumulate too much power. Those parties have to be most likely not to collude. Otherwise, there is no gain from using a distributed approach. If those parties do not collude they can only gain limited knowledge from the sensor directory because PIAs only have to send data to one server which distributes the data further. An example of such a structure would be a distributed ledger (section 3.2). Because this requires parties known not to collude the effectiveness is only moderate. The functionality is not impacted by this change and also the usability stays the same. The performance might be impacted because there must be multiple endpoints that have to be synchronized, the impact depends on the number of nodes and the chosen technology and may reach from low up to very high [15, 51, 91]. If there are multiple nodes that have to communicate with each other, this may increase the impact and probability of a DoS by adding sensors (section 2.2.1) [28]. Implementing such a system comes with rather high costs, a consent algorithm is needed and also, all nodes need to know what they have to do. Luckily most DLTs are frameworks and are already ready to use. If such a system can be used it is easy to implement [59, 67].

- Effectiveness: moderate
- Impact on functionality: low
- Impact on usability: low
- Impact on performance: moderate
- Implementation complexity: moderate

2.4.7 Onion Routing

This attack vector mitigation is able to mitigate:

- Fingerprinting (2.2.3),
- People tracking (sensor directory) (2.2.14),
- Split views (2.2.17)

It is possible to use Tor to relay requests and responses. Because it is possible to use different exit relays when using Tor and also other PIAs would use the same relays, the data is diluted. Because this is the case and lots of data may use the same exit relay fingerprinting is much harder [97, 98]. While it becomes harder it is still possible to identify a client by observing different patterns [97, 98]. However, doing so is quite hard when not in a laboratory environment [98]. Unless there is a perfect fingerprint and the device is therefore identified it is much harder to link requests to one another and therefore track a specific person. Also because it is required to identify a device to split the view for this device, because the same device needs to receive the same split view each time, splitting views is much harder. This is moderately effective because it increases anonymity but fingerprinting is still possible [97, 98]. The functionality is not impacted a lot, this is the case because the application does work the same. The only difference is the packets have to be routed and encrypted differently. The usability is also not impacted. The impact on the performance is also moderate because each packet has to be sent through multiple relays. The complexity is also moderate because the system already exists.

- Effectiveness: moderate

- Impact on functionality: low
- Impact on usability: low
- Impact on performance: moderate
- Implementation complexity: moderate

2.4.8 Pruned Sets and Queries

This attack vector mitigation is able to mitigate:

- Target discovery (2.2.4),
- Data collection (2.2.5)

It should not be allowed to query an arbitrary amount of data, there should rather be several limitations to stop attackers from accumulating lots of information [44, 70]. This means for example the radius should not be selected by the query, it should rather be static or depend on the density of sensors in that area. This should not limit legitimate PIAs but it would automatically require attackers to spend more resources because they would have to query multiple times. The query should only allow searching for connected sensors in an area and not for anything else like a description. In addition, it relieves the server because automatically fewer data is included in the response set [37, 56]. The response set should also have a limit so it is harder to find information in the sensor directory or use the response to attack it [44, 100]. If the response set is huge this could be used by an attacker to create a DoS attack [100]. An example of such a limitation could be for such a set to include a maximum of 100 sensors. This leads to an effectiveness of low against the attack vectors. This is the case because attackers easily can send multiple requests and avoid this mitigation [44]. It also impacts the functionality and usability of the system, this is the case because it may force users to request multiple times. The complexity of implementing this should not be too hard because the response set has only to be filtered not to be too long.

- Effectiveness: low
- Impact on functionality: moderate
- Impact on usability: moderate
- Impact on performance: low
- Implementation complexity: low

2.4.9 Validators

This attack vector mitigation is able to mitigate:

- Compromised sensors (2.2.7),
- Data validity (2.2.11),
- Destroyed trust (2.2.13),
- Outdated trust (2.2.13),
- Split views (2.2.17)

It would be a possibility to add validators to the system whose purpose would be to vote on the reputation of sensors. Those validators would be extra devices that are not anonymous so they can be held accountable if they act maliciously. Everyone should be able to set up exactly one validator, but there must be at least a minimum number of them, so some known big players would be expected to run such validators. Each vote should be documented in a chain of votes so it can be traced who voted for which changes. This also means if a validator is marked as malicious it would be possible to drop their votes or stop them from further voting. One problem is, if a new sensor is added those sensors do not have a trust assessment yet. A solution to this problem is, that PIAs could not use such sensors until validators voted on their trust. Validators should also be used to share their point of view so PIAs are able to compare theirs with those of the validators [51, 71]. In addition, validators should compare their views with each other to prevent the sensor directory from splitting views [51, 71]. This would be easy to implement if the data set used is a merkle tree, this is due to the fact it is sufficient to compare the roots of those trees to see if they match (see section 3.5).

The effectiveness of this approach is moderate because the sensors do not have any trust at the start. It may also be hard to estimate how long it takes until sensors are validated by those validators. An additional problem is, that those validators are extra devices with a new threat model. Also, it could be possible for such a validator to be compromised by an attacker therefore rendering the system useless. This is the case because the attacker may use the reputation of the validator to establish their own sensors [46]. This means the effectiveness is moderate. The functionality is not impacted a lot. The performance is also not decreased unless it is very hard to validate those sensors. The usability may suffer because sensors are not usable from the start. Also, the implementation complexity may be high because there are extra devices needed that have to be developed and their own threats have to be considered.

- Effectiveness: moderate
- Impact on functionality: low
- Impact on usability: moderate
- Impact on performance: low
- Implementation complexity: high

2.4.10 Immutable Data

This attack vector mitigation is able to mitigate:

- Data integrity (2.2.9)

The data structure the sensor directory is using should be one that does not allow any data changes. This may decrease the performance but it should mitigate the threat of someone changing data [65]. Data structures that could be used are merkle trees, blockchains, or similar technologies. This is very effective and has a low impact on functionality and usability [65].

- Effectiveness: high
- Impact on functionality: low
- Impact on usability: low
- Impact on performance: moderate
- Implementation complexity: low

2.4.11 Owner Changeable Data

This attack vector mitigation is able to mitigate:

- Data integrity (2.2.9),
- Anticipate address (2.2.12),
- Override sensor (2.2.12),
- Data validity (2.2.11),
- Same address (2.2.12)

If a sensor is already included in the directory, it should only be possible for the owner of the sensor to change the data. This means it should only be possible to change data if users provide proof they own the sensor. PIAs are expected to check the signature before using any sensors. This is the case so they can be sure the sensor is valid. However, verifying a signature is only possible for known sensor providers due to the fact the public key is required. If the key is available PIAs may verify if the data was changed and if so if always the same key was used. This means the creator of the sensor is always the owner and only the key used in the initial creation is allowed to change it. Because only the owner can change data, the integrity is ensured. Also once an address is included and the ownership of this address is verified, no attacks using the address field are possible. Checking the ownership and signatures may take some time, this is the reason for the performance to be only moderate. Users are not impacted when they act honestly and therefore functionality as well as usability is not impacted. The effectiveness is very high if this is enforced.

- Effectiveness: high
- Impact on functionality: low
- Impact on usability: low
- Impact on performance: moderate
- Implementation complexity: low

2.5 Threat Model Analysis

In this section the mitigations are compared with each other, therefore the effectiveness and the impact on each part are represented by a number. For each attack mitigation in section 2.4, the numbers for the impact are summed up to get a final total impact value. If this number is high the functionality, usability, and performance are impacted only a little while the implementation should be easy. If the number is low the impact on functionality, usability, and performance is high, and implementing this mitigation should be hard. It is important to understand that those numbers are only approximations and cannot be verified yet. To verify those numbers further testing is needed which will be part of future work. For this reason, the impact values are substituted with the following numbers:

- Negative implication of high = 1
- Negative implication of moderate = 2
- Negative implication of low = 3

Table 2.1: Attack vector mitigation classification

Attack Mitigation	Imp. on Func.	Imp. on Usa.	Imp. on Perf.	Impl. comp.	Total
Transaction Cost	3	3	2	2	10
Sensor Verification	3	3	1	3	10
Notaries	1	2	3	1	7
Redundant Sensor Directories	3	2	2	3	10
PIA Cache Sensors	2	3	3	3	11
Distributed Sensor Directory	3	3	2	2	10
Onion Routing	3	2	3	2	10
Pruned Sets and Queries	2	2	3	3	10
Validators	3	2	3	1	10
Immutable Data	3	3	2	3	11
Owner Changeable Data	3	3	2	3	11

Also how easily a mitigation can be implemented has to be represented by a number. Therefore an implementation complexity of low is represented by 3, moderate is represented by 2, and high is represented by 1. For each attack vector mitigated by a strategy, the total impact value is multiplied by the effectiveness to reach a final score. Different mitigation strategies are comparable by this score for each attack vector afterward. To allow for such a calculation each effectiveness has to be assigned a number, those numbers are as follows:

- No security control effectiveness = 0
- Security control effectiveness of low = 1
- Security control effectiveness of moderate = 2
- Security control effectiveness of high = 3

Also here different mitigation strategies would effect different attack vectors in different ways. Therefore this number is only a simplified representation and has to be tested further in hind side by future work. See Table 2.1 for an overview of all attack mitigation strategies and their impact values as well as their total impact value. See Table 2.2 for an overview of which mitigation technique is how effective for which attack vector. Finally in Table 2.3 the effectiveness and impact are multiplied for each attack vector for a final score. This score represents how effective a mitigation is, its impact, and how hard it is to implement. If the number is high, the costs are low and the mitigation is very effective against those attack vectors. It also shows which mitigations are required to counteract the selected attack vectors.

2.6 Requirements

In section 1.2, some requirements for the sensor directory are already mentioned. Those are still valid and should be met by the final implementation of the sensor directory. The requirements mentioned earlier were:

- Anonymity of all users,
- Queriability of the sensor directory,
- Equality for everyone,
- Immutability,
- Verifiability,

Table 2.2: Effectiveness of the attack vector mitigations

Attack Mitigation	Mitigation Effectiveness per Attack Vector															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Transaction Costs	2			2						2					2	
Sensor Verification				3				3	3	3						
Notaries				3				3	3	3						
Redundant Sensor Directory	3	3												3		3
PIA Cache Sensors	2	2												2	2	
Distributed Sensor Directory		2												2		2
Onion Routing		2												2		3
Pruned Sets and Queries			1										1			
Validators					2		2				2	2				2
Immutable Data						3										
Owner Changeable Data						3	3	3	3	3						

Table 2.3: Effectiveness * impact per attack vector

Attack Mitigation	Total Impact	Mitigation Effectiveness per Attack Vector															
		2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Transaction Costs	10	20			20						20					20	
Sensor Verification	10				30				30	30	30						
Notaries	7				21				21	21	21						
Redundant Sensor Directory	10	30	30												30		30
PIA Cache Sensors	11	22	22												22	22	
Distributed Sensor Directory	11		20												20		20
Onion Routing	10		20												20		30
Pruned Sets and Queries	10			10										10			
Validators	10					20		20				20	20				20
Immutable Data	11						33										
Owner Changeable Data	11						33	33	33	33	33						

- Storage requirements,
- Know whom to trust,
- Trust system.

Additional requirements can be derived from Table 2.3. This is the case because it shows for each attack vector the most cost-effective mitigation. Because for each attack vector, a mitigation is required the following requirements have to be fulfilled by the sensor directory as well:

- Redundancy/distribution,
- Queries as well as sets should be pruned,
- Input validation,
- Repeated validation,
- Only the owner can change data,
- PIAs cache sensor data.

The sensor directory should be created in a redundant or distributed way. If a redundant approach is chosen, this allows the PIAs to be much more independent of a single sensor directory. However this also simultaneously increases the work required either to search all sensors in an area or to insert sensors.

This is the case because those steps have to be done at multiple sensor directories. It might also be possible for sensors to be only available on certain sensor directories or to search only specific directories. Another option, which is almost as efficient, is to choose a distributed technology holding the information stored via several nodes. The reason it is a bit less effective is, that it is harder to implement and more vulnerable to denial-of-service attacks, because such attacks may multiply within the network due to the fact those nodes have to communicate with each other. This requirement further improves the claim for equality, because it is easy for users to check the data on another directory or another node.

Queries and sets should follow certain rules. This means it should not be possible to search for arbitrary data. Queries should be limited to defined attributes of a sensor, like its location. That is important to increase the work for attackers if they are searching for specific things. Also, the impact on legit users is not very high. This means it should not be possible to search for specific sensors. It should also not be possible to search for an arbitrarily huge radius. Similar restrictions should apply to the response set coming from the sensor directory. The response set should not be arbitrarily big, due to the fact this would allow attackers to gain a lot of knowledge. Huge response sets also increase network traffic for legit users because sensors that are not required have to be sent. Therefore, there should be an upper boundary of bytes or sensors returned with each query.

Each sensor should be verified when inserted into the directory. This means the data should be checked if it is valid and does not conflict with any other sensor. This may result in a decrease in wrong sensor data, as well as prevent malicious sensors. This job may also be part of an additional device called a notary. Notaries might be used to investigate sensors and sign them if they are valid. If this is deployed in such a way, it could be possible to exclusively allow signed sensors to the directory. It might also be possible to require multiple signatures. Such an approach would also allow users to choose which notaries they trust. If notaries are used they may also be part of the trust system required in the base requirements.

Sensors should also be validated in a repeated fashion. Such a validation could be done similarly to a notary by additional devices called validators. Those could request the sensor directory and check if the sensors found are honest. Those should be able to flag untrustworthy sensors or influence a trust value, which can be checked by the PIAs afterward. Those validators should be known, resulting in responsibility for their actions. This might also be part of a trust system, as required in the base requirements.

Each piece of data should only be updated when initiated by the owner of the data. No one else should be able to change the data. These requirements may collide with some base requirements, like anonymity and immutability. Anonymity may be guaranteed when the only thing needed to update a sensor is the private key of a user. That way, the user is unknown, but the only user able to update a sensor is the owner. It might also be an option to hold numerous different keys, so it is impossible to trace which sensors belong to the same user. Immutability may also be violated. However, if data is immutable, it might still be possible to add another data block to update a sensor. Also if only the owner is able to change data, it might be an option to use a system that is not immutable, as long it can be guaranteed only the owner can change data. Therefore, data cannot be dropped by someone else.

While not a requirement for the sensor directory, PIAs caching sensors would decrease the impact of some attack vectors. It would also decrease the reliability of PIAs on the sensor directory and decreases the information available

for the sensor directory. It also increases anonymity, due to the fact that less communication is required which is also a base requirement.

2.7 Comparison

For objectively comparing different technologies, it is important to rank attack vectors as well as requirements. Also, a importance factor is assigned to each attack vector and requirement to further allow for better comparison.

2.7.1 Attack Vectors

At first, it is important to figure out which attack vectors are most important and therefore have to be mitigated. To better separate the attack vectors, they will be clustered into three categories: critical, high, and low. These categories represent the importance of mitigating certain attacks. For a *critical* attack vector, it is important that a selected technology can mitigate this. If a technology is not able to do so it will not be a top contender to build the sensor directory. For the category *high* it should be expected for a technology to mitigate this attack, but if a technology is not able to do so, it is not removed from the technologies that could be used to build the sensor directory. For an attack vector of the category *low*, it is advantageous, but not mandatory for the technology to mitigate the attack vector. Additionally to clustering all attack vectors into categories they also get assigned a relevance factor. This is the case because, for those technologies that could be used to implement the sensor directory, a score is required to compare them. For this reason, each technology is assigned a relevance factor for each mitigated attack vector.

Category: Low

The first attack vector which is in the category *low* is someone adding wrong data into the sensor directory. While wrong sensors may occupy resources, the impact of the attack itself is only limited. This is also the case as to why this attack vector is only assigned a relevance factor of 1. Also because the sensor directory is expected to be a public log it is expected for everyone to find data. This means everyone is not only expected, but required to find data, this also means data gathering or target discovery is expected and therefore in the category *low*. Because leaking data is part of the system, the relevance factor assigned for those attack vectors is also 1.

One of the requirements of the sensor directory is anonymity of the users, this means it should not be able to identify a single user. If it is not possible to identify a user it is also not possible for the sensor directory to track a person. This means the attack vector is categorized *low* while the requirement anonymity is very important. While this attack vector is not very important to mitigate it is still more important than leaking data, therefore the relevance factor assigned is 2. Similarly, sensors could track people. This however is an attack vector regarding the whole system and can and should not be mitigated by the sensor directory. If users are anonymous within the whole system, this might not be a problem as well. For this reason also for this attack vector, the category is chosen as *low*. Because the impact is similar to the sensor directory tracking someone also here the relevance factor is 2.

Because DoS by adding sensors is only a temporary attack, it is nice if a technology can mitigate this attack but it should not be mandatory. After some time the costs of such an attack may pile up and an attacker might not be able to sustain such an attack. Also if PIAs cache sensors, the impact decreases even more. Therefore DoS is added to the class *low*. Also, a relevance factor of 2 is assigned due to the fact the attack is only temporary.

Category: High

The sensor directory requires a trust system to establish which sensors are trustworthy and which are not. However, the trust system in itself also has some attack vectors which could lead to sensors not being used at all. Because this could lead to legitimate sensors not being used and therefore the trust system being useless these attack vectors are assigned the category of *high*. This is due to the fact that PIAs would ignore the trust value entirely to find all legitimate sensors. Also, those attacks might not be only temporary and severely damage a sensor provider or its users and therefore the relevance factor assigned is 3.

If malicious sensors are in the system, PIAs are required to perform lots of unnecessary work. This is the case because they discover those sensors and try to register at them. By doing so PIAs also leak information to those sensors which could be avoided if those sensors are not in the directory in the first place. All attack vectors regarding malicious sensors are placed in the category *high* because of that. This concerns sensors being compromised as well as malicious sensors being set up. Because this would impact PIAs as well as the information of their users these attack vectors are also assigned a slightly higher number of 4.

Category: Critical

Because it would break the purpose of the sensor directory if someone is able to preregister sensors or add already existing sensors to the system, those attack vectors are placed in the category *critical*. Due to its impact on the system, the assigned relevance factor is 6 for those attack vectors. Also, anonymity is a core requirement and would be broken by fingerprinting which means fingerprinting has to be placed in the category *critical*. This is also the reason for the relevance factor of 7. The same holds true for data integrity. Because the sensor directory would be able to perform any action and to tamper with users, showing different response sets to different users would break the system and is also placed in the category *critical*. The relevance factor assigned to this attack vector is 7 as well. This is because, for all of those attacks, the sensor directory would be useless. Last but not least inserting many sensors with the same address may be the attack vector with the most impact. This is the case because it may not only shut down the sensor directory, any PIA or sensor of the system but also every other system using the same namespace. Therefore this attack vector is also assigned the highest relevance factor of 8.
The following gives an overview over all assigned relevance factors:

- Critical:
  - 8 - Insert many sensors with the same address (2.2.12)
  - 7 - Splitting views (2.2.17)
  - 7 - Data integrity (2.2.9)

- 7 - Fingerprinting (2.2.3)
- 6 - Override sensor (2.2.12)
- 6 - Preregister expected sensors (2.2.12)

- High:
  - 4 - Malicious sensors (2.2.6)
  - 4 - Compromised sensors (2.2.7)
  - 3 - Destroy trust of a sensor (2.2.13)
  - 3 - Outdated trust information (2.2.13)
- Low:
  - 2 - DoS by adding sensors to the directory(2.2.1)
  - 2 - People tracking (sensor directory) (2.2.14)
  - 2 - People tracking (sensors) (2.2.15)
  - 1 - Data validity (2.2.11)
  - 1 - Target discovery (2.2.4)
  - 1 - Data collection (2.2.5)

2.7.2 Requirements

The requirements are split into three groups similar to the clustering of the attack vectors, which correspondingly are: critical, high, and low. *Critical* requirements are the core requirements of the system, which need to be fulfilled by the technologies. Requirements with *high* importance should be fulfilled, but technologies may solve problems in other ways. This means technologies are not expected to fulfill those requirements. Requirements that are of mediocre importance are assigned a *low* value. They are nice to have but not mandatory. Additionally, similar to the attack vectors, each requirement gets a designated relevance factor for a better comparison of the technologies. The outcome is visible in the following list.

- Critical:
  - 10 - Queriability of the sensor directory
  - 9 - Equality for everyone
  - 8 - Verifiability
  - 8 - Anonymity of all users
  - 8 - Only the owner can change data
  - 7 - Redundancy/distribution
- Low:
  - 6 - Storage requirements
- High:
  - 4 - Repeated validation
  - 4 - Input validation

- 4 - Trust system
- 4 - Know whom to trust

- Low:
  - 2 - Immutability
  - 1 - Prune sets
  - 1 - Prune queries
  - 0 - PIAs cache sensor data

Category: Low

PIAs should cache sensors, however, this is not really a requirement for the sensor directory itself, it rather is a requirement for PIAs. Therefore the category is *low* and the relevance factor is 0. The storage requirement is left for future work, because the storage may increase indefinitely for such a system and therefore, cannot be solved in the first investigation. However, it will be tracked for the technologies.

Requirements that get assigned the lowest value are prune sets and queries. This is the case because the sensor directory is developed to be a public log. Whenever queries and sets are pruned, an attacker might circumvent this by just sending lots of requests. However, this still means higher costs for the attacker, which contributes to security. For this reason, the relevance factor assigned to those requirements is 1.

While immutability is a requirement first defined in the definition of the system and therefore was part of the core requirements, the immutability may not be as strict as explained in the definition. The reason for immutability in the core requirements was to ensure no one could mess with the data, even if it is stored separately from the creator. However, if only the creator is allowed to create updates and it is ensured no one else has this authorization, true immutability is not required. True immutability only makes it hard to store the data because it grows indefinitely. While it may still be a form of immutability if only the owner can change data, this does allow for changes to happen. True immutability could still be a requirement, while it is not mandatory. If all data is kept, a history of the data is always available which is a nice feature and therefore true immutability is assigned to the group *low*. However, it still is more substantial than pruning sets and queries and is, therefore assessed with a relevance factor of 2.

Category: High

If a system should be trustworthy, it is essential to know whom to trust. This is the case, because, with enough resources and determination, every system may get malicious. Therefore, it is crucial to know who has to turn against the system, so the system gets malicious. For this reason, this requirement has already been assigned a *high* importance value. Knowing who has to be trusted so the system stays honest is more relevant than the requirements before and therefore is given a relevance factor of 4. Knowing whom to trust is useful to keep the system working. However, if the content of the system is malicious, nothing is gained from that. This means some kind of trust system should be in place to check if the sensors are honest. Trusting the sensors has the same level of importance as trusting the system and its providers. Therefore, it is placed

in the category *high* and is assigned a factor of 4. A trust system is useful, but as already described, for its functionality, known entities are required to scan the data. Those entities have to scan the data when sensors are inserted and it is also required to scan the sensors in a repeated fashion. Because those steps are part of the trust system, the same value is assigned.

Future Work

Storage is generally part of future work, therefore only little effort is put into solving any storage-related problems. This means the storage requirement is assigned an importance of *low*. However, if some technology can solve storage issues, it is designated a relevance factor of 6, because if this problem is unsolved, the system will break after some time.

Category : Critical

The system should be runnable in a redundant way or a distributed fashion. For this reason, the requirement is also in the *critical* importance group and assigned a relevance factor of 7.

True immutability is assigned a rather low relevance factor because only the owner of a sensor should be able to change the data and therefore create a light version of immutability. This means the requirement of the owner exclusively changing data is crucial. It is also given a high relevance factor of 8 because it is slightly more important than running the system redundantly or distributed.

It is an important requirement of the sensor directory for all users to be anonymous. Anonymity is also *critical*, therefore a relevance factor of 8 is assigned. Besides the data owner, the data itself should be verifiable too. This means it should be verifiable that the data is indeed stored on the server, and fulfills the query. Also, it should be verifiable that the whole set of sensors fulfilling the query is contained in the response set. Therefore, the verifiability requirement is also crucial and assigned a relevance of 8.

While the owner of the data might change the data, this should not violate the requirement of equality. This means all users of the system should have the same rights. Everyone should be able to create their own sensors and request sensors from the sensor directory. Considering this is more or less the core idea of the system, it is *critical* and assigned a value of 9.

Because a system without any possibility of interaction is useless, it is also required for the system to support queries. Whereas it should allow adding sensors and queries to search within the data set. Since the system is worthless without this functionality, this requirement is *critical* and assigned a relevance factor of 10.

Chapter 3

Background

In the next few sections, the basic background is explained. These concepts are later required to understand various technologies that are compared.

3.1 Byzantine Fault-Tolerance

To be byzantine fault-tolerant means to be able to solve the Byzantine general's problem. This problem, among other things, is explained next and is taken from [104]. The byzantine general's problem is named after a famous problem of several generals sieging a city. No general is strong enough to conquer the city on his own. Their only chance to succeed is if a majority of the generals attack the city at the same time. To achieve this goal, the generals communicate by sending couriers. The problem is, they do not know if they can trust the other generals. Additionally, generals can send wrong information or deliver variable pieces of information to different generals, which means a general can be a traitor. Another difficulty is, the couriers could get caught or killed by the besieged city, resulting in a loss of their messages. A distributed system has to face a similar problem as those generals. Each node can be a potential traitor sending wrong information, no information or packets can be lost on the wire. Subsequently, it cannot be assumed for all messages to be valid.

One possible solution is, to create a central part that is trustworthy and can overrule other nodes. Referring to the general's problem it would be a general ruling over the other generals and organizing their attacks. If the sensor directory is a distributed system, it needs to tackle this problem and a solution might be needed.

3.2 Distributed Ledger Technologies

Distributed Ledger Technologies, or DLTs, are trusted ledgers that operate in a distributed fashion. To do this they are global, append-only, immutable data structures hosted by mutually untrusted participants [18]. There are two main approaches to this task, the first approach is a blockchain and the second one is a directed acyclic graph (DAG) [18]. Before adding data to the ledger all participants have to agree to the data [25]. This is done so one global truth is maintained across the whole system [25].

3.2.1 Smart Contracts

DLTs often allow smart contracts within the system. Smart contracts are code pieces that get executed if a predefined condition is met and thereby modify the

ledger in a predefined manner [7, 17]. Smart contracts can be written in specific programming languages, depending on the DLT used [6]. These smart contracts can support legal contracts or similar structures. It could be an option to pay taxes automatically or split earnings between partners automatically using a smart contract.

3.2.2 Permission

DLTs can be built with different privacy goals in mind. Some DLTs are public, whereas others are private. Interchangeable to public, one could use the terms permissionless or open-access as well. Most DLTs get constructed to work either in public or private states specifically, but some support both options. Public DLTs distinguish from private ones by allowing anyone to join and participate [104]. Whereas private ones have restrictions for certain parts of their network, resulting in limitations on who is allowed to participate or read data [17, 104]. The consent algorithm used highly depends on the permission type supported. Because public ledgers allow everyone to join, it also allows everyone to participate in the consent algorithm, because everyone can join and exit at any time [104]. Furthermore, the number of users is always unknown and there is no trust for the users. Therefore, consent algorithms for public blockchains are neither very efficient nor fast [104]. Examples of such public consent algorithms are PoW and PoS. For private DLTs, in contrast, users are known and might have different roles. Besides, one could use by far much more efficient consent algorithms [104].

3.2.3 Proof of Work

Proof of Work (PoW) is one of the first and best-developed consent algorithms used by DLTs. The cryptocurrency Bitcoin together with 90% of all other cryptocurrencies is using PoW at the moment [42]. In PoW, users create and vote for blocks by using their computational power by solving a problem. Those users are called miners.

In Bitcoin, the problem that has to be solved is hash-based, which means the miners have to find a nonce inside a new block of the blockchain so that the hash of this block is below a specific threshold [42, 76]. If this nonce is found, the node is allowed to create the new block [42]. Miners are motivated to do so because they are compensated with cryptocurrencies if they are allowed to create a block. When a miner finds the nonce and commits it to the blockchain, all other nodes can check the nonce by validating the hash value [76]. The most important feature of PoW is, that it is hard to create, but very easy to verify [42]. For example in this scenario validating a hash is easy while finding a correct nonce is pretty hard [42]. One problem of PoW is its speed. This is the case because it is so challenging to create a new block and all nodes have to wait until any node has achieved this to add the next block. The difficulty of the problem may vary and so does the time required. A solution to the speed of the system is, to change the block sizes of the blocks [42]. Doing so allows to include of more information in one block and therefore the speed of the blockchain increases.

Another very important problem PoW systems have is their power consumption. This is the case because the problem which has to be solved is very hard and there are so many nodes in parallel trying to solve the problem on their own. However, when one node solves the puzzle and adds a new block to the blockchain, the effort of all other nodes is worthless. The power consumption

of all nodes from certain bigger known PoW systems accumulated is comparable to those of some countries [13, 92].

There is also the possibility of several miners finishing their work nearly simultaneously, which leads to them all creating the next block. These blocks fork the blockchain because both are for the moment valid [42]. This means a second branch in the blockchain is created, this fork can include stale blocks which do not enhance the blockchain further [42]. This is a problem because the miners had to put work in them while not being rewarded for it, they also create unnecessary network traffic and slow down the system [42]. Because both blocks are valid, a solution for this problem is required. The next blocks created are added to one of those chains until one chain is significantly longer than the others and is the new valid chain [104]. The other forked blocks are dropped under this condition [104].

3.2.4 Proof of Stake

Proof of Stake (PoS) is also a consent algorithm used in cryptocurrencies. PoS was developed to solve problems PoW has. Instead of using a hard problem to ensure security, PoS relies on the amount of cryptocurrency (coins) a user has and their age. This means there is no need to do hard calculations and use a lot of power [2, 41, 58, 104]. Because coins are needed for the consent algorithm to work, this can only be used in a cryptocurrency. The security assumption of PoS systems is, that users who have a lot of a cryptocurrency are more invested in keeping the system running than other users [104]. Coinage works as follows, if Alice gives coins to Bob, say she gives him 25 coins. And Bob holds these coins for 10 days, Bob has a total of 250 coin-days which may be used to influence the blockchain [58]. If Bob spends the 25 coins the coin-days get destroyed [58]. Each miner uses their coin days to build up a stake. In each round, a semi-random function is used to find the miner creating the next block [2]. This semi-random function has to come up with the same miner for all honest nodes and depends on the stake every miner has [2, 104].

An important advantage of PoS over PoW, is its reduced risk of centralization [20]. In PoW systems it is possible for some users to have so much computational power, that they nearly can guarantee to generate the next block [20]. But this does not mean PoS is free of such a problem. In the case of PoS, a user can gather many coins, and therefore the user may influence the blockchain on his behalf [20]. This opens an opportunity for those who have much money to potentially get even more money by influencing the chain [104]. It is expected for PoS to be less secure than PoW [104]. As a possible solution Proof of Activity, which combines both attempts, got constructed [104].

3.2.5 Delegated Proof of Stake

Delegated proof of Stake or DPoS is another possible solution to create the next block using fewer miners. Similar to PoS DPoS allows to build a stake, the main difference is in DPoS the nodes vote for a representative [104]. These representatives are selected according to the combined stake size of their voters and all selected nodes work on the consent [104]. This decreases network capacity because only those nodes have to communicate, which also increases the speed of the system. An advantage of this algorithm is, that if there is a dishonest user it is possible to replace this user because the voters just have to vote for another user to solve the problem [104].

3.2.6 Proof of Activity

Proof of Activity uses a simple combination of PoW and PoS. At the start of each round, each miner tries to create a block header containing the previous block, the miner's address, the block number, and a nonce [20]. This header is part of the PoW part of the system, which means that each miner tries to create a nonce so they can solve the PoW. The difficulty of the PoW is lower than that of a pure PoW system [20]. When the puzzle is solved the header is distributed to all other nodes. Using the header, and the stacked coins of the miners, it is possible to determine who the N creators of this block should be with a pseudo-random function [20]. If a node receives a block, the node then checks if the PoW is fulfilled by hashing the header and validating if the value is below the threshold. If this is true, the node checks if it is one of the N stakeholders selected. The first N-1 stakeholders have a look at the block header, and it. Then they distribute the header and the signature further [20]. If the last of the N nodes recognizes the header it creates a block containing all the transactions at the moment known by the node [20]. The final block also includes the signatures of all N-1 nodes that have singed the header before. Also, the last node signs the whole block which is then added to the blockchain [20]. Afterward, the node distributes the block to all other nodes in the network. If a node receives a block it verifies all conditions from above are fulfilled, then the block is added to the blockchain [20].

The difficulty of the PoW highly depends on the number of nodes and the total hash power available at the specific point in time [20]. However, the difficulty should always stay below that of a pure PoW system. While PoA is able to counter problems of PoW as well as those of PoS, PoA does have its own problems. PoA's bandwidth need is significantly higher than that of its direct counterparts, while systems like Bitcoin create their block in one communication round, systems depending on PoA need 2 rounds. In the first round, the stakeholders are found and in the second round, the block is created.

3.2.7 Proof of Personhood

Proof of personhood (PoP) is a consensus algorithm designed to work in a permissionless environment [22, 87]. PoP requires two phases, the first phase is called the setup phase. In this phase, everyone who wants to participate in the second phase receives exactly one PoP token [22]. This PoP-tokens binds a physical and virtual identity, which means each person is only able to hold one token [87]. This means while everyone is anonymous they are still accountable [22]. The next phase is called the minting phase. It starts with a minting pool being formed [22]. Inside this minting pool, everyone with a PoP token is included. Then a random selection is performed to select one token from the pool [22, 87]. The related member is allowed to create the next block [22].

3.2.8 Proof of Authority

Proof of Authority (PoAuth) is a consensus algorithm designed to work in a private environment [7]. This means in such a system the actors are known. Some of those known actors are selected to be authorities. Each authority is selected as a leader for a specific period of time [7]. Whenever a node is selected to be the next leader, the node creates a block and distributes it to all other authorities. Depending on the implementation, all authorities add the received block into the blockchain or they verify all authorities received the same block before

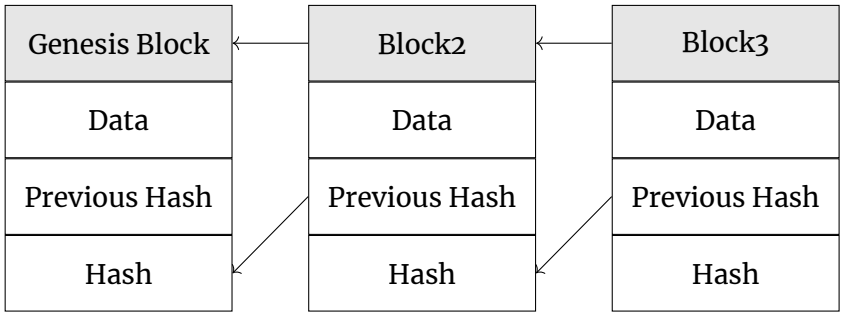

Figure 3.1: Structure of a blockchain

doing so [7]. This is done by distributing the received block to all other authorities and verifying that the blocks match. This step is called acceptance and is not implemented in all versions of PoAuth [7]. If a leader tries any malicious actions the other authorities may vote and kick him from the authorities [7].

3.2.9 Raft

Raft is another consensus algorithm working with a master and several slaves. This consensus algorithm is only an option for a private environment. The leader of the Raft network is elected, and afterward, the leader is responsible for creating entries in the logs of the members [32, 79]. If there are any new blocks, they are forwarded to the leader, which then distributes the blocks to all other nodes. The entries at the members are the consent which is found. Raft can survive the failure of multiple members of the network [32, 79].

3.3 Blockchain

Blockchain is a widespread technology used to store data safely and securely. Blockchains have a wide range of use cases, from financial services to risk management, Internet of Things (IoT) up to public and social services, but their main field of use is cryptocurrencies [38, 76]. The first blockchain was developed in 2008 and implemented in 2009 [104]. In blockchain, all stored data is represented as blocks. Depending on the data used in the system, it is possible to store several elements in one block. Those blocks are used to build a chain similar to a linked list. The structure is shown in Figure 3.1. The link between blocks is created by adding the hash of the previous block to the new block as a reference [104]. The technology prevents malicious users from manipulating old data blocks without anyone noticing and allows everyone holding a verified block, to verify all blocks which existed earlier in the timeline [104]. Each blockchain needs a start, which is called the genesis block [104].

Because the chain is distributed via lots of nodes and a consent algorithm is used to keep the chain persistent over all nodes, the blockchain is decentralized, persistent, and auditable [104]. Since blockchains are mainly used for distributed peer-to-peer networks, they are very hard to tamper with and change [104]. One major disadvantage most blockchains share is their poor scalability. This is the case because nodes have to find consent before adding a new block to the blockchain. Therefore, the addition of new blocks is somewhat time-consuming [104]. The exact duration depends on the technology and the con-

sent algorithm used. To gain a complete view of the data set it is possible to follow the chain from the genesis block to the latest block and update the view accordingly [23]. That allows the blockchain to support updates of the data set because a newer block can update old data, but aged data will always be included in the blockchain. Therefore, a blockchain is immutable. That means, if the complete data of the blockchain should be obtained, n steps are required, where n equals the length of the chain. It is also possible in blockchains to search for specific blocks if the IDs or hashes are known [23].

To validate a block of the blockchain, it is sufficient to store one block which resides later in the chain. That indicates holding the last block of the chain is able to verify all blocks before this block [10]. Referring to Figure 3.1, if anyone holds Block3, this user is able to verify all previous blocks (more precisely the genesis block and Block2) in the chain. In the case all blocks are unchanged, all blocks back to the genesis block can be verified by verifying the hashes of all blocks back until arriving at the genesis block. Simply put, whoever knows about the last block is able to verify all blocks that are part of the blockchain until this point, given the blocks in between.

A blockchain is somewhat resilient to many attacks, but there are still some attacks that might be successful. First of all, someone can try to change data within the chain. For example, someone could try to change the second block (Block2) in Figure 3.1. If an attacker succeeds in changing the data of this block, the hash of this block changes, which destroys the chain due to the fact the next block is storing this hash as a link [10]. This means the attacker also has to change the following block, which also changes its hash. This again breaks the link of the next block. This process proceeds until the last block in the chain is modified [10]. If PoW is used as a consensus algorithm, changing a hash also means the PoW has to be solved. Additionally because the blockchain is distributed on numerous nodes, an attacker has to tamper with the majority to modify the data [10].

3.4 Directed Acyclic Graph

Similar to blockchains, in a directed acyclic graph (DAG) data is stored in a block referring to other blocks. The links are also done using hash values. Due to the fact blocks are not able to guess the hash of the next block the edges of the graph have a direction. While in blockchain a block can only refer to one other block and can be referred to only by one block correspondingly, for DAGs a block may refer to x blocks and also get referred by x blocks, while x is higher or equal to one [18]. Therefore, a major difference between a blockchain and a DAG is, that each block can be validated by multiple new blocks, as well as it can validate numerous blocks in the DAG [18, 29]. Like in a blockchain, if someone wants to change a block, this changes its hash and therefore breaks all children referencing the block [29]. If there is no parallel block creation, and all blocks always refer to the last open block in the DAG, a standard blockchain develops [29]. Another important feature a DAG has to fulfill is to be circle-free, which means when following the directed edges, it is not possible to get to one node multiple times [4, 31]. Due to the fact this would require a block to know the hash of a future block, this is not possible when used by honest users. By using a DAG instead of blockchain, it is possible to improve performance in several ways, however, this approach does not remove the need for a consent algorithm [4]. That means a DAG also relies on PoW, PoS, or any other consensus algorithm. Those consensus algorithms might be the same used for blockchain based systems but they could also be different [4].

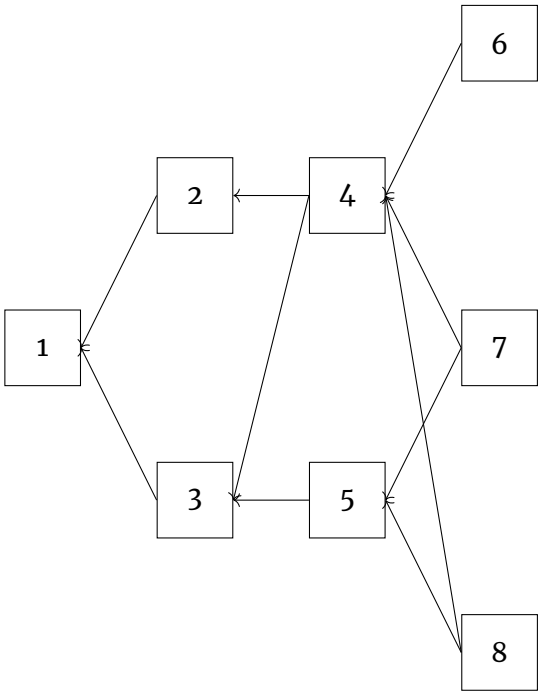

Figure 3.2: Structure of a DAG

See Figure 3.2 for an example of a directed acyclic graph. DAGs can be built in a variety of ways, but they all try to tackle the following problems of blockchain [4]:

- ledger size,
- scalability,
- fork problem.

In a blockchain, all nodes have to add the same blocks after another, resulting in a decreased performance. This is the case due to the fact that the next round of block creation can start earliest as soon as all nodes have stored the last block. DAGs can avoid this because they allow parallel block creation and addition into the data structure [31]. Correspondingly, it does not matter if every node knows every other block before creating the next block, speeding up block creation significantly. But this simultaneously creates a problem because it cannot be expected from each node to know the entire DAG at any moment in time [31]. After all, there may be a block on the wire of which the node is not aware yet.

Another problem of blockchains is their scalability in terms of block creation speed and their inclusion of transactions [31]. Because in DAGs no one has to wait for other users, it is possible to create a block at any time. Therefore this issue is solved, but as already mentioned, it cannot be assumed from each node to know all previous blocks at a given time, which might be of concern.

A more complex concern of blockchains is the orphan problem, also called the problem of forks [31]. Because in blockchains there is the possibility of simultaneous block creation, resulting in multiple chains being created while only one of them is valid. Whereas energy, time, and effort used to create one of those blocks is lost. Because DAGs have the advantage of allowing parallel block creation, this issue does not exist for them[4].

While DAGs try to solve problems blockchains are facing, they also create some other challenges that need to be handled. There are 2 types of DAG-based technologies [4]:

- BlockDAGs,
- blockless DAGs.

BlockDAG works similarly to a standard blockchain, where transactions get bundled in a block and refer to previous blocks [4]. However because DAGs allow for parallel block creation, there is no need to wait and bundle transactions together. Therefore each transaction can be posted as soon as it is available, creating a block for each transaction, leading to blockless DAGs [4]. Due to the fact this is a huge advantage over blockchains, many DAGs use a blockless approach [4].

Because DAGs consist of numerous branches, there is the possibility of those branches holding contradicting data [4]. In a blockchain, this issue is circumvented because the data is structured in one chain, and everyone knows which data was committed first. Therefore, another solution is needed to bring transactions on a timeline in DAGs [4]. This implies a solution is needed to find an order for all blocks which are part of the graph. This is easy if there is a direct connection between two blocks because this connection has to have a direction [4]. This means an order for blocks in different branches is required if they contradict each other [4]. Due to the fact the structure and functionality of DAGs vary a lot their ways of solving this issue differ a lot from each other. For this reason, the solution to this problem is presented in the respective sections for each DAG.

3.5 Merkle Tree

A merkle tree is a structure used to store data in an immutable way and allows to quick verify whether data is included in a data set. Similarly to blockchains and directed acyclic graphs, a hash function is essential for the data structure [82]. This is the case due to the fact the metadata is stored as a tree of nodes where each node is the hash of its child nodes [88]. As leaf nodes the data that should be verified is hashed [88]. Because each node always has two children, if the number of leaf nodes is odd, the last leaf node is duplicated [82]. The root node of the tree can be utilized to verify all data elements included in the data [82, 88]. An example of such a merkle tree can be seen in Figure 3.3.

A merkle tree can be used to prove to another individual, that a piece of data is included in a data set. This can be required after data is inserted into the data set as proof of insertion, or when data is requested as proof that the data is included in the data set [71]. This verification is pretty simple as shown in Figure 3.4. Similarly to the examples from [74, 82], in this example, a user wants to verify T_D is stored in the data set. To prove this data is contained in the data set, the system responds with some required hashes from the tree [82, 88]. Those hashes are H_C, H_{AB}, H_{ABCD}. The response set grows correlating with the height of the tree. However, the size of the set equals only log(n) of all hashes and proves the data is implied in the structure [88]. Each individual is able to use T_D to calculate HD by hashing it. Afterwards, they use H_C and H_D to calculate H_{CD}. After knowing H_{CD} and H_{AB}, it is possible to calculate H_{ABCD}. This gives them the root of the merkle tree. The user can compare the calculated value of the root with the transmitted value. If they match, the user can be certain the data is included in the data structure.

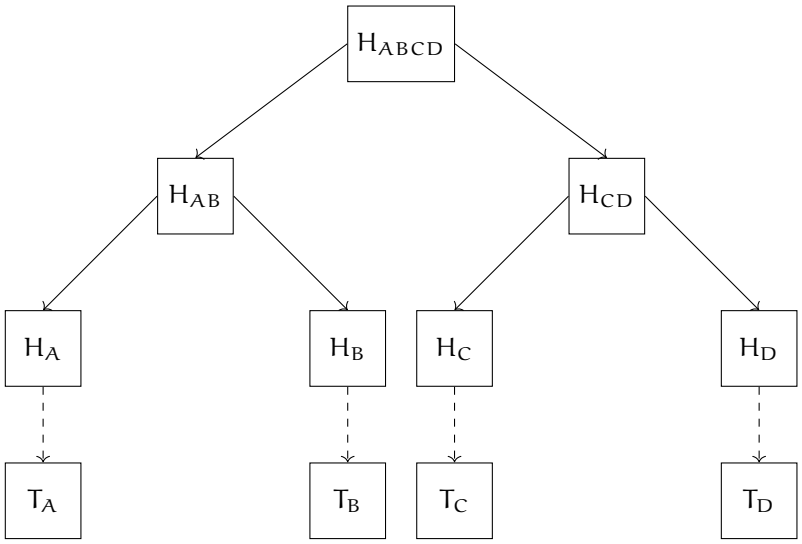

Figure 3.3: Example of the structure of a merkle tree

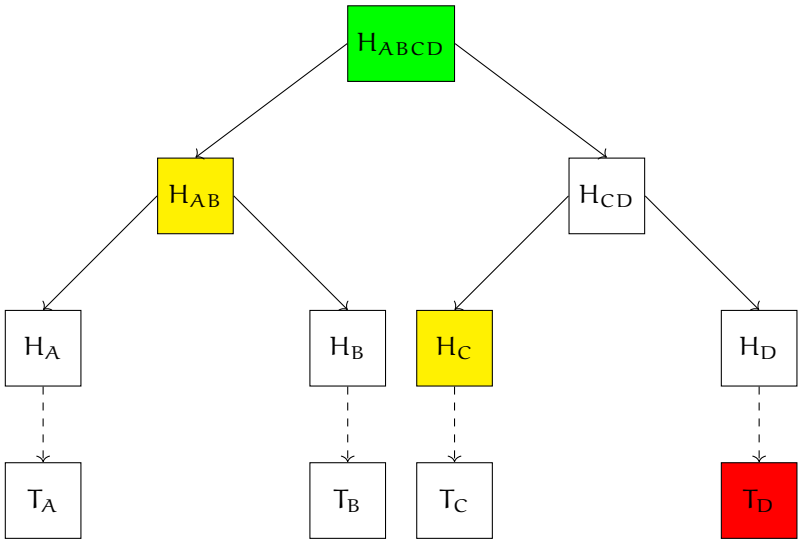

Figure 3.4: Using a merkle tree to prove TD

Chapter 4

Technologies

In this chapter, a lot of different technologies are shown. These technologies are explained and it is shown how they could be used to implement the sensor directory and solve the requirements described in section 2.6. In the next chapter, the technologies are compared and it is shown which of them might be the best choice to solve the directory. Most of those technologies could not be used out of the box to implement the sensor directory. However, only minor changes are required to allow those technologies to be used. Many times those adjustments are API functionality to interact with the sensor directory or minor changes at the storage. When searching for technologies that could be used several starting technologies and keywords were used. Those keywords and technologies were:

- Blockchain,
- Distributed Ledger,
- Distributed Databases,
- Transparency Logs,
- Data Provenance and
- Tor.

Starting from those keywords, technologies are searched and related technologies or technologies that solve similar problems are also added to the search parameters to find several technologies which could be used for the sensor directory. Also, other technologies that were found in the process that solved similar problems were analyzed.

4.1 Certificate Authorities

Today's web communication is mostly encrypted using TLS, making communication much more secure and preventing spying on the content of modern web traffic. Before HTTPS was widely spread, this was not the case, and everyone was able to wiretap traffic easily. Because TLS is an asymmetric encryption method a need for public and private key exist. To be able to match a key to an entity, HTTPS requires known Certificate Authorities (CAs), which are trusted by the clients [1]. Those CAs are used to sign certificates and therefore bind identities to their public keys [1]. The public key of the certificate is then used to communicate with the corresponding server [1]. This approach is by far not perfect, and there are still vulnerabilities that could be exploited. For example, modern browsers nowadays require not only the signature of a CA but also the certificate to be added to a transparency log, which is explained in section 4.5.

4.1.1 CAs for the Sensor Directory

It may be an option to allow certain known users who are generally trustworthy (big players) to set up CAs. Those CAs might sign keys for further CAs, so more CAs emerge. However, those CAs are related to their root CA's trust, and if the root CA loses its trust, also their sub-CAs lose theirs. If sensor providers want to create sensors, they reach out to one or more of those CAs, and those verify the sensor by signing it. There might be so-called universally trusted CAs, which gained their trust because they are run by trusted known operators. But the users should decide for themselves which CAs they trust and which they suspect to be malicious. That means PIAs do not trust each signature from all CAs equally. This also means a sensor provider would have the incentive to get multiple signatures so PIAs accept at least one of those. It might also be an option for a PIA to demand multiple signatures from a subset of all CAs, so the sensor is trusted. In this scenario, everyone can create a CA as long as a user knows about them, it would be possible to trust them and require their signature.

This approach is the basis for trust in a system and can be combined with a variety of ways of storing the data. The most basic approach possible to distribute sensor information is to setup a service providing a list of sensors and the corresponding CAs signatures. Because the storage of those servers is not append-only it might be an option for those servers to decide to drop certain sensors or decline to distribute them, depending on the sensor providers. This cannot be prevented and raises the requirement for the sensor provider to publish their sensors on several servers of different operators, to make sure their sensors are not dropped by all servers at once. However, this might be a requirement anyway because PIAs might search on different servers. The sensor providers may also operate their own servers to prevent their data from being dropped. If the sensor information is spread via lots of servers, this makes it hard for PIAs to find all sensors and gain an absolute view of the world. This means PIAs have to scan multiple servers containing lots of redundant sensors. It is also possible for sensors to be stored on servers the PIA does not know about which means those sensors would not be discovered. To prevent this, the owner of the PIA should be required to configure which servers the PIA queries for sensors. There might be a lot of standard servers that are used, but the PIA owners may also speak to sensor providers and discover additional servers. In Digidow for most sensors, an initial step might be required. In this step permissions for an individual might be exchanged and other preconditions can be met. This action involves communication between some entities of the sensor and the PIA. Because this is the case this step could also be employed to introduce new servers to the PIA. Whenever a PIA receives data, it may verify the signatures of the CAs and verify additional requirements, like a minimum number of signatures. Because the data is signed, the server is not able to tamper with the data.

A CA approach would allow mitigation of some attack vectors, as selected in section 2.3. See Table 4.1 for an overview of those mitigated attack vectors. An approach using CAs allows the redundant setup of many servers providing the data, which is part of the requirements in section 2.6. However, it is possible to attack a single server with a DoS. This would temporarily disable the server, therefore, sensor providers have to provide their sensors at multiple servers. Also CAs could be the target of DoS attacks preventing them from signing new sensors for the duration. However sensors which already were signed still have their signature and are therefore still valid. Using lots of servers might also be a solution to fingerprinting (section 2.2.3) because one server does not have the required data to perform such an attack. It also allows PIAs to mitigate the possibility of tracking people by the sensor directory (section 2.2.14) by

Table 4.1: Mitigated attack vectors by a CA approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
CA	X	X		X	X	X					X	X		X		

randomly selecting sensor directories. However, multiple servers also have a huge disadvantage, PIAs need to know all the servers they need to crawl to gain the necessary data. A CA approach is not able to protect from target discovery (section 2.2.4), which is the case because the provider of the server can perform such an attack easily by searching the data and the CA information can not do anything against this. Everyone else is also able to perform such attacks by querying those servers and searching for vulnerabilities. Similarly it is also possible for anyone to collect additional data. Server providers might choose to prune sets and queries, while this might not be the case for all, it does not stop such attempts it does only slow them down a little. This is the case because PIAs are able to perform lots of requests and therefore get lots of knowledge even if queries and sets are pruned.

If a request is received, servers have to respond with the sensor data and the corresponding metadata. By doing so the PIA is able to verify the signature and therefore knows the sensor is authentic. However, there is no data structure tying together multiple pieces of data which means it is not possible to check if a server responds with the full data set fulfilling the query.

Server providers are not required to accept each entry, this is the case because each server stands on its own and may create its own policy. This could be a problem because those servers might exclude certain sensor providers. This means those servers are able to choose who is allowed to publish their data and who is not. Because users are anonymous they are not able to perform such actions on ethnicity or religion, however, they are able to do this on the data provided like the user's key or location of the sensors. The same holds true for CAs and their signing step. Because there is no immutable data structure like blockchain used the server is able to drop data stored on the server. This violates the requirements while it also allows the server to drop malicious sensors when discovered and therefore save storage space. This might be also done after data has lost its trust or when data is updated. Servers might choose to allow sensor providers to update their sensors by checking their signature, however, they are not forced to allow this.

The validity of the data itself could also be a problem. While it is signed by a CA it might be possible for malicious CAs to exist. This means all attacks that could be done by inserting wrong data into the directory are an option for attackers. This is also the case for all attack vectors regarding the address field (section 2.2.12). PIAs however are able to check which CA signed the sensor and are able to choose not to use sensors signed by untrusted CAs. However, if only valid information should be stored on servers the servers have to check validity themselves on insert or only allow for sensors signed by certain CAs. It might be sufficient for some PIA operators to verify the signatures of trusted CAs. Those CAs can be expected to sign trusted sensors only and therefore to verify the data. This means not the server is required to check the data but the CAs are. Therefore, the trust in some CAs might rise even more and the trust in others might decrease because it is not known if they perform those actions. This validation

Table 4.2: Requirements fulfilled by a CA approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
CA	X	X	X		X	X	X	X			~	~	X	X

could already be seen as an initial trust assessment, however, servers might also support different trust assessments. After some time, there might evolve an adventitious version of trust assessment, and PIAs can implement their behavior according to this. Until then, PIAs have a hard time following different trust assessment techniques. In section 1.2, some trust assessment techniques are described, which could be applied by different servers. Each server could be expected to check the sensors when adding them to the storage and act as notaries additional to CAs. So far there is no way of checking validity of sensors repeatedly. One option would be for certificates to have a limited lifetime. This would require sensors to renew their certificate from time to time which leads to CAs verifying sensors periodically. However this means the data on the servers has to be updated to hold the new certificate. Also the trust into CAs might be heavily impacted by the duration of their certificates. If a user knows which CAs they trust, they also know which sensors can be trusted. Nevertheless if a CA is not known a PIA does not trust the sensor even if the CA is actually honest. Therefore the trust problems are solved but some honest sensors could be missed. Another option would be to add validators that are used to influence a trust value on the server. However trust values that are stored on the server are not protected by the CA signature and therefore can be changed by the server. This requires the trust to be signed by the validators. If this is the case the server is still able to drop the trust data because there is no immutable structure preventing the server from doing so. A solution could be if validators sign the whole trust data as well as the sensor data when adding their trust. This would generate a chain. However, the server still has the ability to drop the last signature.

Servers are able to drop sensors if they lose trust in them. It is also possible for a server to split views (section 2.2.17) between different users without anybody noticing it. This can only be discovered, if PIAs gossip with each other. However, if PIAs are truly anonymous, a server might not be able to do so because it is not possible to show the same wrong view to the same PIA each time. The requirements that can be fulfilled by using a CA approach can be seen in Table 4.2.

4.2 Web of Trust

Web of Trust is a base concept used to establish trust without a need for centralization. Web of trust is built on the concept users have higher trust in information if other users trust this information too. Therefore a concept is re-

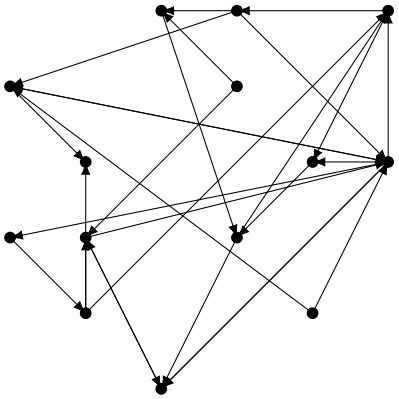

Figure 4.1: Graph of trust

quired to enable users to show which data they trust. Users can show their trust by signing data [21, 102]. If a user downloads the data, the user can verify the signatures and therefore gain trust in the data. The main benefit of such an approach is to allow users to gain trust without the need for any centralized certificate authorities (CA) [26]. This type of trust assessment can be used for every piece of information. Therefore it is also possible to use this concept to establish trust in the keys of other users to create a public key infrastructure (PKI) [26, 40]. If such a concept is employed this does not only allow for explicit trust assessment but also for implicit assessments [26, 68]. In more detail, someone having faith in a specific user may also trust users that are trusted by this user. When this concept is applied to its fullest potential, a graph like in Figure 4.1 is created. It is also possible for such trust assessments to not only show which users they trust but also to include a value for this trust [26]. This also enables implicit trust to decrease the further away the initial trusted signature is [26]. However, if the trust graph is complicated the calculation of such an implicit trust gets quite complicated [26].

One technology which is widely known and is used to raise trust into public keys is PGP. PGP uses a network of key servers to share public keys [21]. This is done to enable easy encryption of e-mails and messages. Trust may be demonstrated in four different trust levels when PGP is used [102]:

- Full (level=4),
- Marginal (level=3),
- Untrustworthy (level=2) and
- Do not know (level=1).

Because PGP does not allow removal of keys it gives access to 20 years of PGP keys [21]. This means there are also lots of trust assessments stored that have to be considered when calculating trust for a public key.

As long as a key server is available, it is only possible to add new data but not to remove any from the server. Instead, if a public key is invalid, it can be marked as such [21]. Data on the key server can only be revoked by users harboring a revocation key for this specific private key, which is especially important because the users may not be able to abolish the key themselves. It is important, in PGP no one is expected to trust the key servers, and everyone can push data into them. Therefore, users should only use the server for informational purposes [102]. In the beginning, the first servers were stand-alone servers, which exclusively stored their datasets [16]. Later, they began to send emails to each

other to update other key servers [16]. Further improvement allowed sending multicasts to other servers [16]. Key servers may also have the possibility to crawl other servers to gather more information. Because there is no standardized communication whenever data is updated on one server, it takes up to 30 hours until information is updated via all communicating servers [102]. This is the case due to some servers not being synchronized with others regularly. Consequently, if a key gets revoked, this is not the case for all servers immediately [102]. Because this is a huge drawback, there are several approaches using new techniques to improve PGP key servers. One option is to use blockchain-based technologies; another one would be to use DAT [5, 89]. Both solutions try to create a decentralized network performing automated synchronization.

PGP key servers based on blockchains, seem to solve many of the problems normal PGP key servers have. One such implementation is using Ethereum to solve the known problems of such servers [5]. This approach is distinct, which means it is not running on the standard Ethereum blockchain. It rather runs its own nodes. This is done because this approach makes the blockchain, by far, more lightweight. Allowing the usage of a considerably more lightweight consensus algorithm, like Proof of Authority. Additionally, the cryptocurrency is removed from this instance [102]. Because a blockchain is used in addition to the signature, it is possible to ensure only the creator of the initial block or an administrator (because of Proof of Authority) is able to change included data [102]. Since this approach is based on Ethereum, it can use smart contracts. The smart contracts in this implementation are used for the following functionalities [102]:

1. Checking the rights of the user trying to change data.
2. Add certificates into the blockchain and give the right to change entries.
3. Sign a certificate of another user. This signature has to be accepted by the owner of the key.
4. Revoke a certificate.
5. Revoke signatures from a certificate of another user.
6. Accept the signature of another user for one certificate.

The implementation using Ethereum also provide some additional benefits:

- The whole history is available [102].
- Due to the slightly different blockchain the entries are easy accessible (see section 4.23).
- Only the real holder of the key can update their key [102].

By using Ethereum the risk of downloading a malicious outdated certificate gets minimized, as the synchronization problem does not exist anymore [102].

4.2.1 Web of trust for the Sensor Directory

The sensor directory can be built using a web of trust approach. However similar to the CA approach web of trust is about building trust and not about storing data therefore this can be combined with each storing technology. Sensor providers would send their sensors to the server, including them in the data set. Each user of the system, meaning sensor providers but also people who want to be identified, can own a key pair which could be used to sign sensors of other users they trust. This key is also used to sign the sensors created by

a sensor provider. Because those sensors are signed, they cannot be modified later. The signatures also allow verification of the data of a sensor. However, it is not possible to verify if a server sends the whole set of data, fulfilling a query. Signatures establishing trust in sensors cannot transition any further, this is the case because sensors do not have the ability to trust any other sensor. Yet, there is the possibility of additionally including user keys in the system. This would allow users to trust not only sensors but also other users, leading to a transitive trust in the sensors and users those users trust. By signing sensors they trust, users generally act as validators because they show their point of view for several sensors. Some users might sign lots of trustful sensors. If they get known for being trustful, the reputation of those users might increase as well, and probably additional users will listen to them regarding trustful sensors. This means those users might have more power than others because their signature has more value. Everyone can add whatever data they want into the system and trust is generated over time when other users sign the data. This means there is no initial check for validity, raising a distinct problem of inclusion of wrong (section 2.2.11) or malicious sensors (section 2.2.6) into the directory. Besides, the address field does not get reviewed and attack vectors of this field (section 2.2.12) are not mitigated. It might be an option for servers to check certain fields before data is included in the data set. This means the server could act as a notary before data is included to counteract some attack vectors. However, servers are not required to do so which means some servers would do this while others would not. PIAs and sensor providers would have to choose which servers they trust and use according to known information. It might also be possible to use CAs before data is included in the system to act as a notary and validate the sensors before they are inserted. Such an approach would also be easier to verify. Signatures as trust assessment also have additional problems, this is the case because they do not show when the trust was issued, leading to potentially old signatures (section 2.2.13), misleading people. Besides, compromised sensors (section 2.2.7) still have the trust they had when they were compromised. However, a user is able to revoke a sensor in the same way they would revoke a key. While this may take some time until available on all servers it might do the trick. To mitigate this situation it might be necessary to remove trust once issued. It might also be an option to include timestamps or a period of validity to a signature, however, this means after some time signatures lose their validity and therefore the trust in a sensor decreases. This means users have to sign sensors periodically.

If a sensor should be updated this is only possible by the sensor provider creating a new entry in the system. This sensor then has to gain new trusted signatures. If the sensor is not owned by the creator the sensor would gain no signatures and therefore only the owner is able to create entries. Nevertheless, owners are only able to disable sensors in the system by revoking them. If there is an update it is a new sensor that has to gain new trust. However, if the web of trust is mainly used on the user level increasing trust in users, instead of the sensor level, the impact of such situations is significantly decreased. An overview of the requirements when using this approach is shown in Table 4.3.

PIAs may search for trusted sensors or trusted users on servers. Depending on the used data structure, it might be possible for stand-alone servers to drop sensors, including all their signatures. The only possibility to mitigate this is if sensor providers check whether their sensors are still part of the server. However, this also allows standalone servers to maintain their storage while this might not be possible for cooperative systems. Each PIA is able to query servers without demonstrating their personality and therefore being anonymous. For sensor providers, this might be true as well because they are represented by a key. However, because the keys are used quite frequently and are

Table 4.3: Requirements fulfilled by a web of trust approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
Web of Trust standalone	X	X			X	X	X	X			X	X	X	~
Web of Trust DLT	X	X		X	X	X	X	X			X	X		~

not only used to add sensors, the identity of some users might be discovered. Because servers store information and everyone is able to query this information, this also means everyone can read the information, this also means everyone is able to scan for vulnerable sensors (section 2.2.4) or other valuable data (section 2.2.5). Because servers may not follow the same rules, servers might choose if they prune sets and queries. However, because users are anonymous they can circumvent this by sending lots of requests and receiving the data anyway.

Also the chosen structure the servers use is important. While the web of trust approach can be combined with each storage technology, there are 2 different types described next. This means if servers are standalone, communicate with each other, or are a distributed system all along. If the servers are standalone ones, they may be vulnerable to DoS (section 2.2.1), but prevention is rather simple through the communication of servers within each other. However, it is essential if and how those servers communicate. It is possible to set up the sensor directory in a redundant way, regardless of this. Anyway, if an approach based on a distributed technology is applied, it might be an advantage to cooperate on one system. Because the directory is redundant or distributed, it is not possible to use fingerprinting (section 2.2.3) or tracking people (section 2.2.14) if PIAs distribute their requests. A distributed technology as a base not only improves update time on all servers but also makes the system immutable, because those systems are based on immutable data structures. Additionally, if a DLT is used it may include a validation of the sensors in the consensus algorithm. The impact of DoS could also be mitigated if servers validate sensors before propagating them. However, if such an approach is used it also means data can never be deleted, this means the storage grows indefinitely and at some time the system will break. DLT approaches require lots of work at insert, which means inserting data might take longer compared to stand-alone servers. For an overview of all attack vectors see Table 4.4.

4.3 Domain Name System

The domain name system (DNS) is widespread and deeply rooted in the modern internet. It is used to resolve human-readable names, like URLs, to their associated addresses [11, 72]. If the system can be trusted, this can also be used as a

Table 4.4: Mitigated attack vectors by a web of trust approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Web of Trust standalone	~	~		X	~	X		X	X	X	X			~	X	~
Web of Trust DLT	X	X		X	X	X		X	X	X	X			~	X	~

PKI [11]. Most of the data presented in this section is from [72]. In general, DNS consists of two distinct parts. There are name servers, which are used to resolve human-readable names to the corresponding data [9]. And resolvers which are used to find name servers that are responsible for the human-readable search terms.

DNS is a distributed database holding data as well as their human-readable counterparts. Whenever a request is sent to a name server containing the human-readable string a lookup is performed and the stored data is returned. Therefore first the corresponding name server has to be found. To be able to do so the human-readable names are structured in a tree. Each child node contains the name of the parent while also adding specific data. For example, the child node might be google.com which also includes the parent node which is .com. This also means at the top of the tree there is only an empty node which is the parent of the first layer of the tree [9, 11, 72]. For each node, there is a name server responsible for storing the associated data. In the case of the internet, the first layer contains the top domains [9, 11]. Some of those top domains are .com, .org, or .edu. How the structure of those domains looks can be seen in Figure 4.2. Each name server can also have secondary name servers to provide the possibilities of redundancy and load balancing [9]. Those secondary servers query the main server regularly and update their data [9].

Resolvers are used by clients to resolve human-readable names. This is done by clients sending requests to the resolver, which triggers the resolver to send requests to several name servers trying to resolve the name [103]. If a name is unknown to the resolver, it tries to find a server responsible for this name, starting with the top domains. Those domains respond with the information about which name servers have further information on those names, so the resolver is able to continue its search [103]. Whenever a name is resolved the resolver caches the entry and will use this cached data if the same name should

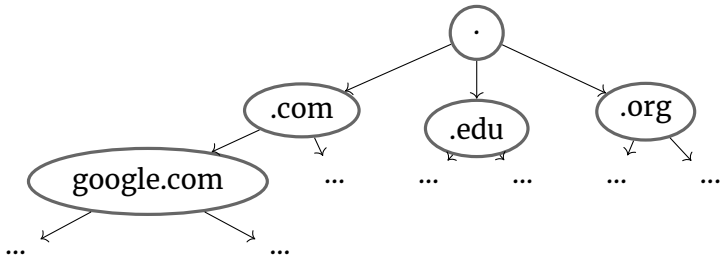

Figure 4.2: Structure of the DNS system

be resolved in the near future [103].

While DNS is part of the backbone of the internet and therefore is widely used, it is not free from vulnerabilities. DNS is for example vulnerable to man-in-the-middle (MITM) attacks, this is the case because there is no way of verifying the origin or integrity of a response [9]. Obviously because of its importance the system is also a target for several DoS and distributed DoS attacks. Those attacks aim to disable the higher layers of the tree because the impact is much higher [9]. An attack where an attacker manages to change DNS answers on their way to the user is called DNS spoofing. This could be done by a MITM attack. If this is successful, the attacker is able to connect the victim to whatever address the attacker prefers. This might be especially effective for conducting phishing attacks.

4.3.1 Domain Name System Security Extensions (DNSSEC)

DNSSEC is explained next, most of the information is taken from [9, 11]. DNSSEC works similarly to DNS but adds origin authentication, data integrity, and authenticated denial of existence. At first, all data in DNSSEC gets signed by the server. Therefore, a client can verify the data is received from a legitimate server. To allow signatures, private and public keys are required in the system, however, in DNSSEC no encryption is used [9]. DNSSEC is backward compatible to DNS by supporting four different record types:

- Resource Record Signature (RRSIG),
- DNS Public Key (DNSKEY),
- Delegation Signer (DS),
- Next Secure (NSEC).

RRSIG is the type of content applied for the signatures and used to authenticate the data. DNSKEY contains the public signing key of a server [30] while the DS entry type contains the hash of the key [30]. The NSEC record is used to show nonexistence, this is done by responding with the next entries in both directions [90]. In addition to those record types, two flags of DNS are used to indicate if all data has been authenticated by the server and if unauthenticated data is acceptable. Also, each entry has a link to the next included entry in the domain [9]. By doing this, it is ensured no unauthorized access can modify or delete any data.

DNSSEC is not able to mitigate all attacks that could be performed on DNS. DNSSEC does for example not prevent bad configuration or bad data [9]. Also, DDoS attacks are still possible [9]. Additionally, it introduces other problems which have to be solved. Because keys are required, key management or key infrastructures are needed [9]. Generally, the tree structure of DNS is used for the keys, but the root has a private key assigned. This key is then used to sign the keys of the top-level domains, those top-level domains use their keys to sign the next nodes and so on [11]. Also, the computational load is much higher than in DNS, as well as it requires a much higher synchronization [9]. However, the lack of tolerance of malicious servers is most important [9].

4.3.2 DNS for the Sensor Directory

If DNS is used to implement the sensor directory, there would be a need for parties running the system, at least for the higher layers. One of those parties would be the root party holding the root key. Several other parties have

to be selected, which would be the top-level domains. It would be an option to distribute those top domains throughout the world, so each region has its dedicated top domain. However, it should not be required for anybody to register at a specific regional top domain. Otherwise, this domain could stop users from entering the system. This means, for example, there might be a domain like "europe". Then there might be several subdomains for example "germany.europe". Sensor providers might register with their own domain in these subdomains, such as "db.germany.europe". The sensor providers might then use subdomains for their sensors like "mainentry.db.germany.europe". The entries might then be resolved using DNS. It might also be a possibility for the company domain to show lots of sensors, this means for example "db.germany.europe" would include mainentry as well as additional sensors. In a similar fashion, it would be a possibility to create entries for specific locations like Linz. This entry would hold all sensor domains in this area, which allows for an easy search. This means a PIA would search in the area domain for sensors and afterwards use the returned domain to receive further information. However, such an approach does also bear some problems. The first problem is who the provider of such servers would be. It cannot be assumed that there is one entity willing to host this service for each region. Similarly, there might be multiple entities willing to host such a service for other areas. It is also important if such a domain is expected to accept each entry, or if those domains can reject certain sensors because those are expected to be malicious. Therefore, it is also crucial if the entities running those regional servers are trustworthy. This also means that it is very important how sensor providers can provide additional sensors as well as how they can update existing ones. If such an areal approach is chosen, it is also important how big the area for each domain is. An additional problem occurs for sensors at the edge of multiple areas. For those sensors it might be unclear in which area they should be in or if they should be in multiple areas. Another option would be to use the already existing DNS system and add new domains to this system. There are two ways of using the already established system. Either an additional subtree is added to the system, for example by adding an additional top domain. Or sensors are added into domains distributed over the whole tree. The first option is to add an additional top domain for example "digidow" to the first layer of DNS and add all domains specific to this application to the subtree. This would allow to use already established servers while it does not change anything else in the structure than adding ".digidow" to all domains. By analyzing this domain and its subdomains it is possible to analyze who provides sensors and at what approximate location. Another option is for all sensor providers to add their sensors to their own domains. The problem is sensors would be distributed via the whole DNS system. However, DNS does not support verification of integrity nor does it allow to authenticate the origin of data [9]. Therefore it is not an option to use it for the sensor directory. However, while DNS does not support those features DNSSEC does and might be an option.

When DNSSEC is used, it is ensured that only verified data is used. Also, DNSSEC supports no-inclusion searches by following the chains between entries in a domain to prove something is not included [9]. Because DNSSEC is vulnerable to malicious servers, attackers who can set up such servers or take over higher layers of the system may impact the system significantly. An example of an attack by a malicious server would be a server signing and providing wrong or different data to different users (section 2.2.17). Furthermore, the subdomains of such a malicious server might be impacted. This is a huge problem for the sensor directory if it is implemented using DNSSEC.

Because there are some big parties needed which have more power than others, this might not be 100% decentralized. However, everyone still is able to

Table 4.5: Mitigated attack vectors by a DNSSEC approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
DNSSEC	~	~				X								~		

maintain their own data and provide it to the whole world. Another problem might be, that the domain providers have different power levels and therefore they may use it to influence those on lower levels. This may violate the requirement of everyone being equal. All users requesting sensors are still equal to each other, however.

Because resolvers cache addresses, name servers get relieved as well and they receive less information. However, PIAs have to use different resolvers because otherwise, those resolvers might be able to locate a person (section 2.2.14) or are able to use fingerprinting (section 2.2.3) to identify PIAs.

Based on the structure PIAs are limited to search for domains and gain the respective sensors. This means the query would only allow for certain data and therefore would be pruned. Because the data is stored by the server and is not stored in an immutable way the sensor provider might change data when needed. But because this server is local or at least managed by the sensor provider those providers are the only ones able to change data. Because adding sensors only affects local name servers, this means there is no possibility for DoS by the addition of sensors. DoS is still a high concern, this is the case due to the fact higher layers may be needed to find lower layers and if their service is impacted the service of the whole system is impacted. Because data is not encrypted and publicly available, everyone is able to search for targets (section 2.2.4) or other useful data (section 2.2.5).

Because everyone may manage their own domains, this may lead to situations where everyone can add whatever data they want. Therefore, wrong sensor data or malicious sensors (section 2.2.6) may be included. However, those might not be trusted because the domains may not be known or trusted. If an approach is chosen where one server provider manages lots of information from different sensor providers additional trust assessment might be needed. If sensors are taken over, sensor providers are easily able to replace them with new ones or manipulate their sensor data in a way this sensor is not included anymore. Storage is also not of concern due to the fact everyone is managing their own data on their own devices. Which attack vectors can be mitigated by using DNSSEC can be seen in Table 4.5. Also see Table 4.6 for an overview of which requirements DNSSEC can solve.

4.4 Secure Untrusted Data Repository

Secure Untrusted Data Repository (SUNDR), is a network file system designed to work even if it runs on untrusted servers. The functionality is explained in this section, the information is taken from [66]. SUNDR is fork consistent for all honest users having permission and using it properly. It also allows each user to detect all unauthorized modifications and ensures integrity and consistency as long as all modifications are known. This is possible because each piece of

Table 4.6: Requirements fulfilled by a DNSSEC approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
DNSSEC	~	X	~		X	X		X	X				X	X

information is requested by its hash making the system a high-performance hash table. Because each piece of information is requested by its hash this also means it is easy to verify the data. To allow for users to find data, there are links for each users and group referring to the files they have access to. Whenever a user fetches or modifies date on the server the user creates a block which contains the link, the user, and the modification and updated the link by adding this block. Because the link includes also all previous blocks a chain of signed information forms. If a user only fetches data, the last modification to this data is added again to signal this request. Whenever a user interacts with data the user downloads the newest information and verifies, the last action the user performed is included as well as all users had permission to perform their actions.

Whenever a user or the server tampers with the data, this is discovered while those links are verified. This is the case because those users do have local copies of the data and are also aware of their last actions. If those actions are not present in the link or the data does not match the changes something malicious is happening. This does mean a file can't change unrecognized, as long some users have the data stored locally and can verify the data and actions. If the server tries to hide data from certain users, a data fork is forced. This is the case because users verify their last actions are reflected by the data. This means at least two valid data objects have to exist. When the server drops a block from the links, the latest activity of a specific user is missing, this means this user will refuse to sign this data and therefore will no longer participate for this piece of information. The server might circumvent this by showing a valid version of the file to this user only. If there is communication between the forked parties, they can effortlessly detect the fork, and everyone can get notified not to trust the server. If an honest server is employed and a disastrous event happens, the server can recover or even back up from untrusted clients by using their data.

4.4.1 SUNDR for the Sensor Directory

SUNDR creates a file system for each root user, containing its individual users and groups. This means if SUNDR is used to build the sensor directory, there are two options. Either everyone is working on one file system where one root user is in charge of adding additional users, or each sensor provider creates its own file system. The first option would create a central point and an ultimately powerful user, who would be able to decide who is allowed to participate and who is not. That violates the requirements and the idea of the sensor directory

Table 4.7: Mitigated attack vectors by a SUNDR approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
SUNDR			X	X		X	X	X	X	X	X	X	X		X	

and is therefore no suitable option. The second option is appropriate for everyone who wants to provide their sensors. Each sensor provider can reach out to a SUNDR server and create their file system there. After they have achieved this, they can create files containing their sensor information. Each PIA that tries to find the sensors has to know where to look for the data and is required to be authorized to do so. This means they are required to hold user data for each specific SUNDR file system. Whenever permissions for Digidow users get exchanged, the sensor provider would also have to interchange the server data including a user. The sensor provider would correspondingly have to add a new user, executed by adding the PIA key into a file within the file system. The user of the PIA may get assigned a group that is only allowed to read data. By doing so, no one except the owner of the sensors would be able to change the sensors. Another option would also be to assign the same user to each PIA.

Sensor providers would create files containing their sensors on those servers. Whenever they update the data they would validate the server is honest by validating the link structure. However, PIAs would be expected to download those files and store them locally. This means while PIAs can search easily in those files for sensors, they would also require lots of storage. Periodically PIAs could update their local view using the server. Considering there is a user required to find data, there is no need for the sensor directory to use fingerprinting (section 2.2.3) to discover PIAs. This is due to them being already known and not anonymous. Also since many sensor providers would create such servers, PIAs would additionally have to store lots of users and the associated servers.

Because a sensor provider has to set up a whole file system it is hard to set it up in a way the information is available redundantly on several servers. If a sensor provider would still try to do so, they would be required to set up equal file systems on different servers. This increases the independence of the sensor provider from one specific server provider and increases the availability of the data but does not add a lot for PIAs. For most technologies, PIAs could use this redundancy and request data at different locations to disguise themselves. This is not the case for SUNDR because the sensor provider is in charge of all instances and receives all data, like which sensor is requested when by whom. Sensor providers are able to link this information for all file systems they are in charge. This means it might be possible for the directory to track a person (section 2.2.14). Also if a server is used by many sensor providers to host their file systems it might become a target for attackers to deny its service. Since only the sensor provider can add sensors, only traditional DoS attacks or attacks by requesting sensors by known users are an option. While it is hard to defend against the traditional attack, defending against an attack consisting of requests can be denied by removing the user from the file system. Table 4.7 gives an overview of all mitigated attack vectors.

The data is immutable and verifiable by the data structure. This means no data can be changed or added without getting logged. However, it might be possible for the server to drop data. This will be discovered because users store the

data locally as well. Regardless, the data stored by the users might not be up to date which means it might not be discovered by all. It is also possible to allow other users to change certain files, so it might be an option for larger companies to have different users in their IT teams who can change sensor data. That would also allow for non-repudiability because it gets logged who changed which data. Furthermore, because all data gets incorporated within their file system, no unknown user can search for vulnerable sensors (section 2.2.4) or valuable data (section 2.2.5). However, the server owners might be able to read data and perform such attacks. If the data is encrypted, no one without the key can collect any information. Sensor owners can update sensors whenever they want. Whenever a sensor gets taken over and the owner notices, the sensor can just be removed or updated. Additionally, each user is only able to see what the user is allowed to do, therefore the set is pruned. Due to file servers only holding data of one sensor provider, the query is pruned as well. However, this also means, a PIA is never able to find all sensors in an area using SUNDR. Further, if SUNDR is used it is still possible to fork views of different users (section 2.2.17). That could only be solved if two PIAs would communicate with each other. In theory, users could also detect this if there are changes in the data and their last requests are not logged. However, if the data is stable and does not change a lot, the server can continue splitting views. Additionally, a server might not only choose to split the view of users, but it might also decide to drop a file system entirely for any given reason. Therefore, until a redundant server exists, the sensors are lost, which would be a disaster. If this happens the sensor provider would have to start with a new file server. Even if this server could recover the information effortlessly from the user's storage, the used server might not be the same as last time because it dropped the data once already. This means all PIAs need to be informed, that there is a new server with the same data, and they have to search at this server from now on instead.

One additional problem exists if there is a sensor that does not require permission. This is the case because PIAs require a user and a server address to use a SUNDR server. Because this information would be exchanged with the permissions required to use certain sensors, if this step is skipped also the information required to use the SUNDR system is missing on PIAs. To solve this issue it could be an option to provide an online directory containing such SUNDR servers and belonging users. This user would only have read access and would be the same for all PIAs. Because each PIA would use the same user lots of information is dropped the sensor provider would receive otherwise. A similar approach could also be used by other servers to regain anonymity by giving each PIA the same user.

Last but not least, because only permitted users are able to change data a trust system might not be required. PIAs might trust sensors because they know the sensor provider providing them. A malicious user has no incentive to create a new SUNDR file system and provide their malicious sensors, this is the case because this system would not be used and therefore the sensors would not be discovered. This is the case because PIAs would not use this file system since they do not know anything about it as well as would not have a user to do so anyway. Also if PIAs recognize a file system contains malicious data, the PIA might choose not to use the file system anymore. Given that it is impossible to detect sensors that are not created by a known user it is not feasible for unknown users to locate persons and their corresponding sensors. However, a malicious known user could perform such attacks. Also because sensors can be taken over there might be the need for validators. Those validators need additional permissions because they are tasked with checking for sensors and updating trust information. But this means those validators also require users to get access to sensors. This user can only be created by the sensor provider. Because those

Table 4.8: Requirements fulfilled by a SUNDR approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
SUNDR		X		X	X	X			X	X			X	X

would be the only ones who can generate malicious sensors, malicious sensor providers would not allow honest validators in the system. That means even if there are validators, those would not be trustworthy because they are chosen by the sensor provider. This means if the sensor provider is malicious, it can be expected, also the validators are malicious. The same holds for servers discovered in any additional director used to discover SUNDR servers. Because the sensor provider might not known there is an even bigger threat of malicious behavior. This means such a directory would need a trust value for the whole server.

If a sensor provider adds nonsense data to its directory this is allowed. However, PIAs might not use this directory if this is discovered. This means sensor providers are also able to do attacks that use the address field. Since PIAs only discover parts of all sensors it might not be a problem if addresses are included multiple times. However, if an attacker manages to include such a sensor in all directories of all sensor providers this might be a problem. Nevertheless, because no one has permission to do so this is not a threat. The requirements fulfilled when applying a SUNDR approach can be seen in Table 4.8.

4.5 Transparency Logs

Today's secure web is mainly based on TLS/SSL. TLS/SSL is using Certificate Authorities (CAs) to verify certificates [55]. This is done by CAs signing certificates to indicate their validity. For this, a server creates a key pair consisting of a public and private key. The CA then signs the public key and therefore shows its trust in this key and that it belongs to the user stated. If a browser receives a website with a signed certificate, it checks a list of CAs and can verify whether the key is signed by one of those trusted CAs. If this is the case, the browser can trust the encryption, and the entity is allowed to host this website. This system is essential for the internet to work, though it has some flaws. These shortcomings were already exploited by attackers to disguise themselves [46, 55]. One problem is, that there are no restrictions for whom a CA is allowed to sign certificates [46]. An additional problem with this structure is when a new key is signed, the owner of the domain does not get a notification. When someone manages to get a key signed for a domain, its owner is unaware of it [46]. Therefore, when a CA is malicious or gets taken over by a malicious user, they can sign certificates for every domain without anyone being able to stop them from doing so [46]. Whenever someone manages to get a CA to sign a certificate for them, they can use it to mimic large sites like Amazon or Google. The only

countermeasure for browsers is to remove the CA from their trusted list, which also invalidates all other certificates signed by this CA.

To prevent such a scenario where CAs can sign certificates for any domain without the owner knowing, certificate transparency (CT) logs were introduced and first applied in Google Chrome [46]. This concept is described next and is directly inspired by [46]. A CT log is a public append-only log. These logs are monitored by monitors, whereas their cryptographic integrity is verified by auditors. Each CA has to include issued certificates into one or multiple CT logs, where they are appended to the append-only log, and a promise called SCT is returned. Whenever a browser connects to a server to retrieve a website, this SCT is sent to the browser in addition to the certificate when TLS/SSL is used. This does allow the browser to verify the certificate is part of at least one CT log. Similar to the CAs, also not all CT logs are trusted by each browser. Therefore, it is especially important which CT logs are employed by the CAs [33]. To choose the most trustworthy CT logs browser developers did some research on them and chose those they expected to be trustworthy [33]. But this also means if those CT logs do not act properly, they can lose their trust and get removed from the list similar to CAs.

The traditional way of building transparency logs is by using merkle trees (section 3.5). This is the case because a merkle tree can be used as an append-only structure and supports easy proofs for appends and inclusion. To further improve this approach, it is possible to split merkle trees into multiple smaller trees and add the root of the last tree into the next one as a leaf node [82]. This does reduce the work required to add new data into the tree because fewer hashes need to be recalculated when new data is added. This technique allows gaining the advantages of blockchains or ledgers while still being hosted traditionally by only one party [51]. However CT logs do also have drawbacks, for example, they do not support non-membership proofs, so CT logs cannot support efficient revocation for certificates [51].

Monitors can be operated by anyone. Their task is to check the content of CT logs and verify everything is legitimate [46, 55]. If malicious CT logs are detected, they can be excluded from trusted lists of browsers. Additionally, auditors verify if CT logs are cryptographically consistent by gathering information and verifying it against known data. This is accomplished similarly to monitoring in a periodic instance. Auditors can be elements of monitors, or designed as separate software parts [55].

There are still some problems with the idea of CT logs, which cannot be mitigated easily. Even if all certificates are published in CT logs, and all browsers check those, it does not mitigate the possibility of a CA creating malicious certificates and anyone using them. It only allows other users to notice the problem and allows them to do something about it when the CA publishes the information in a CT log [91]. This means domain owners have to monitor those CT logs so their domains are not registered without them knowing. Another issue is, that CT logs can grow indefinitely [33]. That is the case because CT logs only support appends, and it is impossible to remove entries from the logs [33]. Leading to a data structure that is growing and increasing its dimension over time. To avoid this, CT logs use temporal shading, where a CT log is allowed to limit entries in a specific range of time. Most of the CT logs use intervals of one year, where they only accept certificates issued in this time period. Therefore, different CT logs are required per time slot and all certificates can be added to those CT logs. That implies there may be a CT log for 2020, one for 2021, and so on. This limit supports maintenance and keeps the system running smoothly while reducing the impact of one CT log in case of failure. However this does not reduce the amount of storage required for the data but might decrease overhead.

Table 4.9: Mitigated attack vectors by a transparency log approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Transparency Logs standalone	X	X				X								X		X
Transparency Logs verification	X	X				X								X		X

4.5.1 Transparency Logs for the Sensor Directory

The sensor directory could be built similarly to the CT system. There could be several sensor directories, operated by non-trusted operators, which get monitored by multiple non-trusted monitors and auditors. If a sensor directory turned malicious, the monitors would flag it, and the PIAs do not have to use this directory any more. PIAs could also get the root of the merkle tree from the monitors, which would prevent the sensor directory from splitting view for several users (section 2.2.17). Sensor directories can be run by everyone who wants to do so. But this might imply, there are numerous small, rarely used directories. Therefore PIAs would have to query several of them to receive a total set of sensors. Sensor providers would also have to publish their data into many directories because they do not know which ones will be queried by PIAs and which are trustworthy. It would be much more convenient if there were some big logs operated by known companies, like Google and Amazon. Which would always be queried by the PIAs. Sensor providers would still have to add their sensors into multiple logs because they cannot trust logs to stay honest, but they can expect PIAs to find their sensors much more effortlessly. Everyone is still allowed to run monitors or auditors. If a sensor provider insists and still wants to add the data into another log, the sensor provider should inform the owner of the PIA to adjust the PIA in a way it searches in these logs as well. Similarly, browsers do have their trusted CAs and CT logs also PIAs would have trusted sensor directories they are using to retrieve data. Because there are lots of logs, PIAs can distribute their requests among them and therefore mitigate fingerprinting (section 2.2.3). By doing this, the risk of the sensor directory locating any person (section 2.2.14) is reduced. However, this is only possible if sensors are included in multiple logs. See Table 4.9 for an overview via the attack vectors which can be mitigated using CT logs.

Merkle trees add metadata to the stored data, making this data verifiable and immutable. Each system has to stores data inside a database of some kind. This allows the server to retrieve requests, perform searches in the database, and return a valid response set. This response set would also include information from the merkle tree to verify the entries contained in the returned set. Because the server providers have direct access to the database they easily can search within the data and gain lots of additional knowledge. Therefore, there is no protection against the owner searching for vulnerable sensors (section 2.2.4). The sensor directory might only allow interaction with the data via queries, so it is more challenging for ordinary users to perform such an attack. Those queries should be designed in a way they allow maximal flexibility for PIAs while mitigate all attacks. Obviously the most important query is for the location of the sensor or sensors in a specific area. However, because there are monitors and validators it is required to allow those entities to do their work

and design queries required by them as well. Because everyone is able to request data in this way it is also easy for anyone to collect data on potential targets via the sensor directory (section 2.2.5). Verifying a response is easy by verifying the root of the merkle tree. Because each piece of information is included in the log it is easily possible for a monitor or auditor to verify the content of the logs. It also prevents the sensor directory from deleting any information. It also allows sensor providers to update their sensors by adding additional information into the log updating the sensors. The owner of a sensor can be identified via the key used in the creation and update.

It should be an option for everyone to run their own monitors or auditors. This might create trust in several logs which means there is no need to create additional logs, this should lead to a decrease in needed logs and therefore work for sensor providers and PIAs. This means fewer servers are monitored by numerous devices and correspondingly those are much more trustworthy. Monitors also have the capability of verifying content, this means those devices might be able to verify sensors and their trust as well. This data might be needed to flag not only which CT logs are trustworthy but also which sensors on which CT logs can be trusted. To support this CT logs might hold additional information which is provided by those monitors. To make this data immutable as well, this data has to be incorporated into the underlying data structure as well. If this is not done it might be changed by someone. This inclusion might be possible by adding additional nodes to the merkle tree verifying those entries. If data is retrieved also this data has to be sent and verified using the merkle tree. However, this does increase the work required drastically because for one sensor there might be lots of trust entries that have to be verified. If the sensor directory additionally verifies a sensor before it is included in the directory, it acts as a notary and therefore mitigates some additional attacks because only verified entries are allowed. However, neither such a form of monitoring nor such a verification at insert is used by the standard CT log approach.

The need for storage of such a system is still a problem because this means the storage grows infinitely. The problem is even worse because everyone can publish sensors even if they are not trusted. Each log can be targeted by a DoS attack denying its service for a specific amount of time. Such a scenario is also possible by adding lots of sensors. To limit the impact of many sensors being added, sensor directories may verify sensors before they are inserted into the directory, however, this does require resources and therefore increases the initial impact of such an attack. However, if this is not done the storage requirement drastically increases and the system eventually breaks down. Correspondingly, a sensor provider might add sensors into multiple logs, to increase their availability via sensor directories.

Another way of using transparency logs to create the sensor directory would be, to have servers that actually store the sensor directory, while additionally running logs including verification data of those servers. This means the actual sensor data would be stored on the server. The log would store information needed to verify the entries in the server which could be for example the hashes of the different sensors. By doing this the storage required on the immutable system is minimized while the data provided is maximized and integrity is still provided. However, this means it might be possible for sensors to be dropped as well as information to be outdated. Also if trust is stored on those servers there is a need to store those values securely, otherwise the server can manipulate that data. However, this trust assessment cannot be made by monitors because those only have access to the hashes. An overview of all requirements fulfilled by using CT logs is given in Table 4.10.

Table 4.10: Requirements fulfilled by a transparency log approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
Transparency Logs standalone	X	~	X	X	X	X		X						X
Transparency Logs verification	X	X	X		X	X		X					X	X

4.6 Append-Only Authenticated Dictionaries

The goal of Append-Only Authenticated Dictionaries (AAD), is to improve the performance of transparency logs. This is needed because, in the traditional approach for CT logs, either lookup proofs or append-only proofs are of linear size [91].

An append-only proof is used after a user adds data to the log, it is used to prove the new version of the log contains old entries and the new data is only appended. AADs, realize this by proving the old data is a subset of the new structure [91], which will be shown later. A lookup proof is used if a user requests data of a certain key. Using this proof the system is able to prove that the response contains the entirety of the data for this specific key. If this proof is not available, the system could show different sets of data for the same key to different users [91]. This proof alone does not mitigate the problem of two users receiving different responses, however. The server is still able to fork and create distinct logs for different users. This situation allows the server to answer correctly under the condition mentioned above [91]. But the server would have to respond with the same set of changes to the same user each time [91], which means the server has to store lots of additional data. Furthermore, the user has to be identified. If users communicate with each other such a situation can be detected easily [91]. Merkle trees as they are used in CT logs have the problem, of either append-only proofs or lookup proofs being of logarithmic size and never both [51, 91]. If the merkle tree is organized chronologically, append-only proofs do have logarithmic size. If the merkle tree is organized lexicographically, the lookup proof is of logarithmic size [51, 91]. The issue is the corresponding other proof is always of linear complexity [51, 91]. A possible solution would be to combine both versions of the tree. However this means there would be two trees and each work step has to be conducted twice, additionally proof is needed both trees hold identical data at any time [91]. AADs, try to tackle this problem and show how to combine logarithmic-sized append-only proofs, as well as poly-logarithmic-sized lookup proofs and poly-logarithmic worst-case time appends [91].

An AAD is a tree storing key-value pairs and supporting lookups as well as (non)membership proofs. The structure of AADs and how those function is explained next and is directly taken from [91]. First of all the data elements are ordered lexicographic. Because the values of one key should be located next to each other in the tree, the key is concatenated with the values. Each value

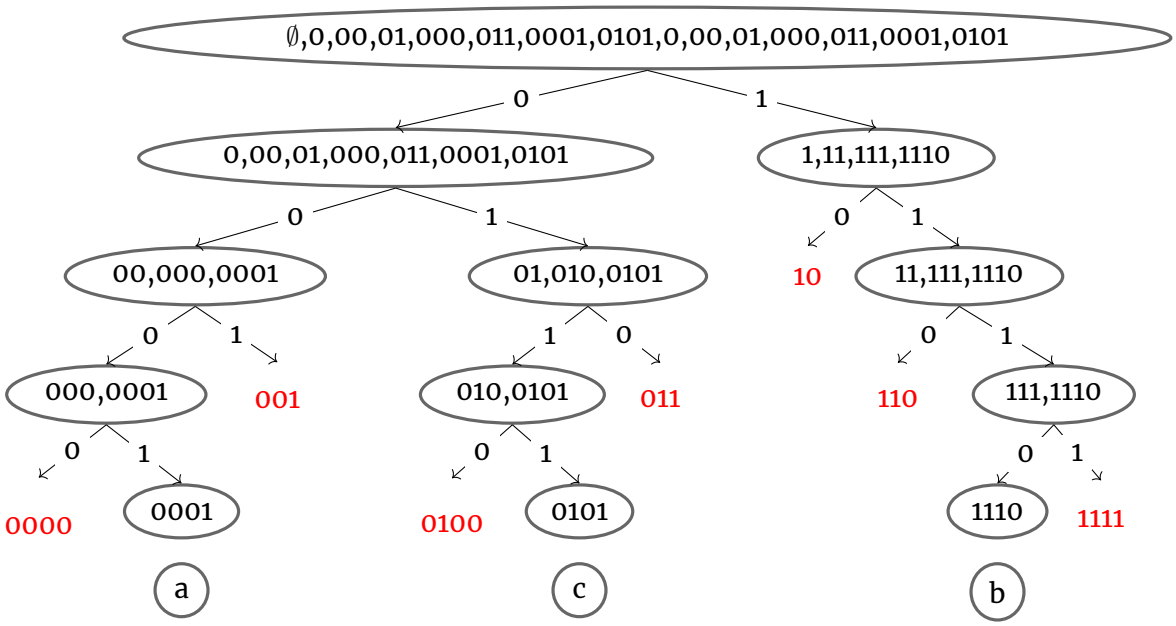

Figure 4.3: Example of a bilinear tree which contains a,b and c

is represented by a leaf node in the tree created. The tree is bilinear and each layer holds the union of the two nodes below. This means the root contains the union of all nodes. Because this is the case once included data cannot be changed or deleted. This structure already allows for membership proofs. To support non-membership proofs as well it is required for each block to store all elements from the nodes below as well. Such a structure is called a bilinear prefix tree (BPT), and an example can be seen in Figure 4.3. A membership proof is now possible by viewing each node as a subset of the node above. Therefore, if a membership proof is performed, a subset proof for each higher layer is done to ensure an element is stored in the tree, leading to a proof of logarithmic size. To perform a non-membership proof it is sufficient to prove an element is not in the subtree it should be in. This proof can also be used to prove similar elements are not included. For example, if d=0111 and f=0110 are given, it is sufficient to prove that 011 is not included in the tree, and therefore neither of them can be part of it. Users may periodically request the latest version and make sure the system is append-only by performing an append-only proof with their old data. This can be conducted by performing a membership proof using the old data. The lookup-proof can be done easily by proving all nodes under the key without a value are dead ends. This proof is possible in O(log n) per node. Because AADs still have very high append times as well as enormous storage consumption, it cannot be seen as valid option for the sensor directory.

4.6.1 AAD for the Sensor Directory

Because AADs are not yet ready for a productive system, this might not be an option. Because AADs are developed to work the same way CT logs do, they would work similarly to CT logs would work for the sensor directory. This means AADs also bring the same advantages and disadvantages to the table as they do for their initial purpose. This means while they would allow for verification as well as ensuring integrity, they would also have very high append

Table 4.11: Mitigated attack vectors by an AAD approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
AAD standalone	X	X				X								X		X
AAD verification	X	X				X								X		X

Table 4.12: Requirements fulfilled by a AAD approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
AAD standalone	X	~	X	X	X	X		X						X
AAD verification	X	X	X		X	X		X					X	X

times as well as enormous storage consumption. The overview of all attack vectors, and which could be mitigated, is shown in Table 4.11.

Whenever a PIA tries to query the system it sends a request containing the key to a server. Said server could perform a search in the tree and return only the response set fulfilling the query as well as some integrity information in the form of some intermediate nodes. This search would be done by always searching the next node containing the searched information. However, because the root does contain all nodes below the system could also return that root requiring the PIA to perform searches on its own.

Because an AAD maps keys to values it is important which characteristic is selected to be used as keys. One option is to use the keys of the sensor provider as a key. This would bundle all sensors of one sensor provider below one key. This would allow PIAs to search for trusted sensor providers and would always return all sensors created by them. Another option would be to use the location as a key. This would allow PIAs to search much more easily. Also because in the AAD the union of child nodes are included in the parent this means there is already an option to search for an area. This is the case because all sensors in an area are located below the same node which means only this parent node has to be returned. It also allows for non-inclusion proofs, which means it is also able to check for locations without sensors. Another option would be to add all sensors without any key and search within all of that data. This would decrease the amount of storage required in the system. Also, similar to transparency logs, it would be an option to use this structure as an integrity information server storing only hashes while the data is stored on vulnerable data servers somewhere else. Doing this might decrease the used storage capacity drastically, while still maintaining confidence in the data. However, the different data structure an AAD provides would not provide any benefits in this situation. All fulfilled requirements can be seen in Table 4.12.

4.7 Merkle^2

Merkle^2 is another approach to improve the performance of transparency logs. It enables users to efficiently monitor data while supporting low latency updates [51]. Whereas this is not possible with traditional CT logs, where it may be necessary to wait more than an hour for updates, for the purpose of lowering monitoring costs [51]. This improvement is possible by using, two types of merkle trees in combination. The used structure allows for improved complexity of monitoring, appending, and lookups.

The data structure is based on the different methods available to build a merkle tree. There are two different merkle tree options [51, 91]:

1. lexicographical trees,
2. chronological trees.

Lexicographical trees enable very fast lookups while having the disadvantage of linear complexity for append-only proofs [51, 91]. On the other hand, there are chronological trees that enable very fast append-only proofs while having linear complexity for lookups [51, 91]. Both have complementary advantages and disadvantages, which is the reason why Merkle^2 combines both. That is accomplished by nesting multiple merkle trees in each other. To be more clear, lexicographical trees are nested in chronological trees [51]. A chronological tree is used as the base of the data structure, this means the leaves are ordered according to the time they were committed to the system. In those leaves, the key-value pairs of the users are stored and given a sequential number. For each internal node of the calculated tree not only the hashes of the nodes below but also the root of a lexicographical tree is used to find the hash value. For this lexicographical tree, the nodes are ordered by their index and then a merkle tree is built. If one user adds multiple values, the data gets concatenated in one node. How this structure looks exactly can be seen in Figure 4.4.

The server is responsible for maintaining this data structure and providing a way to interact with it for other participants. Three participants interact with each other in the system [51]:

- Server,
- Client,
- Auditor.

Servers and Clients work similarly to their counterparts in the CT environment with the distinction of using another data structure. Also, clients have to check for their keys regularly and only the owner of a domain is allowed to publish new data. In this case, the owner is the first user to publish something for one domain. Also, Auditors have a similar goal in mind, they verify the consistency of the server and also guarantee, that clients and other auditors are provided with the correct view. Auditors also gossip with each other to verify each other's views and confirm the server is honest. Everyone is allowed to participate in one of those options. Using a chronological tree as the outer tree allows for a very efficient append in log(n). To also enable efficient proofs, auditors have to check for consistency. That means old versions of the tree are included in the new structure. Doing so allows clients to only check for their key values every few iterations of the tree.

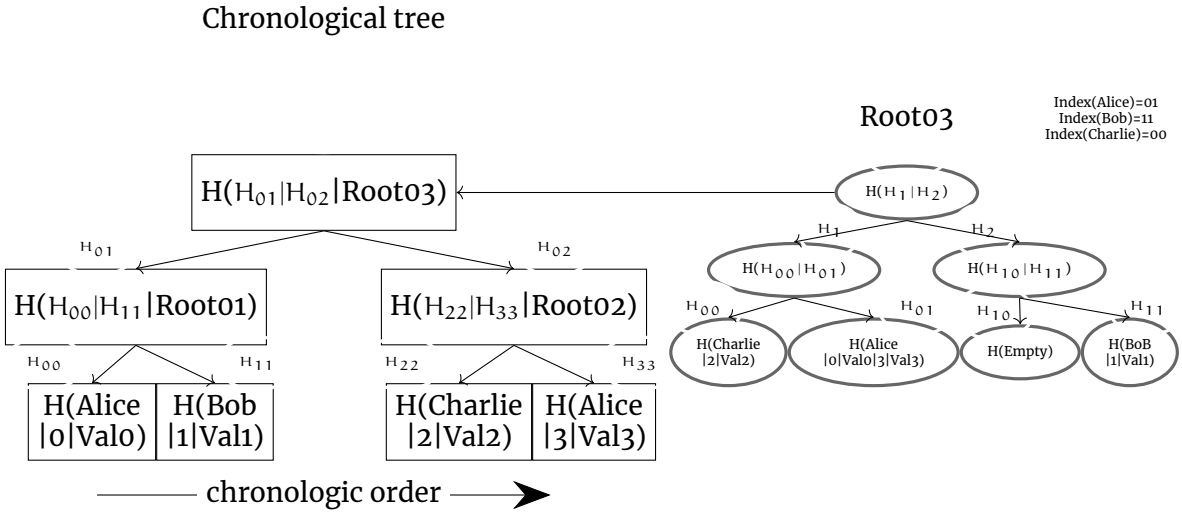

Figure 4.4: Storage structure of Merkle²

4.7.1 Merkle^2 for the Sensor Directory

Because Merkle² has the same functionality as a transparency log, Merkle² would work similarly when used for implementing the sensor directory. Similar to AADs and CT logs, several options for the key exist:

- the location,
- the address,
- or the sensor provider.

When using location AADs might have an advantage, this is the case because the union is stored in the parent, and therefore the information gets less accurate with each layer and therefore an areal search is possible. This is not the case for Merkle², this means sensors might be stored with limited accuracy so they would be found by PIAs. Otherwise, PIAs are required to search for their exact location PIAs do not have. The address is also an option for all of such systems. However, if the address is used, PIAs do have to know an address space they are interested in, this means it is not possible to find specific sensors without prior knowledge. However, already in this prior knowledge the sensor could be contained. Similar to AADs also the sensor provider could be used to allow PIAs to search for trusted sensor providers. Additionally, it would be possible to use the system to verify information stored in an additional server as described for transparency logs. However because this data is stored on another server it is not possible the Merkle² verification server could interact with the data directly. In Table 4.13 an overview of all mitigated attack vectors is shown. Table 4.14 shows the fulfilled requirements.

Table 4.13: Mitigated attack vectors by a Merkle² approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Merkle² standalone	X	X				X		X	X	X	X			X		X X
Merkle² verification	X	X				X								X		X X

Table 4.14: Requirements fulfilled by a Merkle² approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
Merkle² standalone	X	~	X	X	X	X		X						X
Merkle² verification	X	X	X		X	X		X					X	X

4.8 Software Distribution Transparency and Auditability

Software Distribution Transparency and Auditability is an extension to the Advanced Packaging Tool (APT) to gain additional security features. APT is a widely used Linux package manager [48]. The improvements are accomplished by following concepts and features from transparency logs. How the system works is explained next and the explanation is heavily influenced by [48]. The concept of a transparency log is adapted, and a similar system using a merkle tree is created [48]. The server holding the merkle tree should allow package managers to link source code to binaries, to enable better audit and forensic methods. To allow this, the source code has to follow the rules for reproducible builds, so all binaries of the same source are equal [61]. The system requires the following participants to work [48]:

- User,
- APT server,
- Maintainer,
- Public log and
- Monitors.

Users want to download or update binaries. They want to do this most securely and efficiently. The APT server is the file server which is used to manage software packages. It allows users to download their software as expected. There is no difference from a conventional APT server. Maintainers deliver the software packages, sign the source code, and publish the packages into one or more public logs, as well as APT servers. The public logs maintain a merkle tree. The merkle tree includes the hashes of the signed build environment, the

Table 4.15: Mitigated attack vectors by a Software Distribution Transparency approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Software Distribution Transparency	~	X			X	X		~	~	~				X		X

source code, and the metadata. Metadata can, for example, contain all maintainer keys. Those keys can, once included, never get discarded but they could be labeled expired. The log allows the user to verify a package is included in the merkle tree. If a new root is created, the log pushes the root of the merkle tree into the merkle tree of another log and therefore commits to its view. Monitors work like that of CT logs, they verify the consistency of the logs. Monitors can also be used to further investigate items added to the tree. A monitor can, for example, verify a release is reproducible. A monitor can also commit to validating the view of a log, this is done by validating that the root of the merkle tree is included in another log. This prevents the log from splitting views (section 2.2.17).

4.8.1 Software Distribution Transparency for the Sensor Directory

If this improvement of APT is applied to implement the sensor directory, it means an APT server is needed where all sensor providers publish their sensors. It might even be possible to distribute the sensors via multiple servers, but this implies the servers have to be known to PIAs or there is some form of infrastructure in place distributing requests. They could all be run by mostly trusted entities, like Google or Amazon, and would therefore be known and even more trusted. However, they do not necessarily need to be trusted, because they are checked as well as they can be replaced by another server. Also when the sensor directory is built this way there is no need to implement a way to interact with the data due to the fact this is already possible using the APT server. Public logs can be run by everyone, but it might be favorable if some entities which are mostly trusted volunteer to set up some standard logs. It should be considered those running the servers and logs should be distinct to increase security. However, the most used logs might also be those of big companies, like Amazon. Also, since the logs are not trusted at all, everyone could set those up and the system would work. That leads to many servers in parallel, as well as redundant logs, which can get queried. If PIAs use several of them, there should be no possibility of fingerprinting (section 2.2.3) or tracking someone (section 2.2.14) by anyone. However, this requires sensors to be registered at multiple servers. Yet, APT does not allow for packets to be deployed on multiple servers, this is the case to prevent dependency issues, nevertheless, those problems are not present for sensors which means sensors may be present on multiple servers. Because the system allows for every participant to be redundant, the impact of one actor being denied is limited. The only impact is this one server being down for the duration of the attack. An overview of all attack vectors is shown in Table 4.15.

For APT, devices are required to download an index file which allows those devices to search for specific information, before downloading the actual package [53]. For the sensor directory, sensors could be indexed by their location or

their sensor provider. This would prevent malicious use of any other field like someone trying to receive all sensors using a specific software version. Also, the description of the sensor or if it is currently active could be masked that way. It would also allow for searches of location which is probably the most important query. However this means PIAs are required to download a huge index list for each server including query data, this might be lots of storage required on the PIA. Additionally, those indexes should be unique to identify the sensor which means if it is the location, this field has to be precise.

Everyone can run monitors and therefore validate that a log is working honestly. This means there would be multiple monitors verifying each log. Since everyone could set up a monitor and verify a log, logs have to be honest, or they lose all their trust. Sensor providers publish their sensors in APT servers following the use case. Because APT is built to distribute software, it can also support lots of data, therefore even if sensors require lots of additional space, it might still be an option. The sensor provider would most likely commit their sensors to multiple logs and servers to ensure their data is secure and available. This way PIAs can choose which servers they use. If sensor data has to be changed for some reason, this is possible by creating a new version of the data in the APT server and additionally adding a new entry in the logs. The data in the APT server is not immutable. However, the data in the log is and it will be discovered if the server changes data. APT servers are also able to manage their storage, this might lead to problems but also may solve storage issues. PIAs need to know which APT servers they have to search in to find the corresponding sensors. To allow this there might be some standard servers a PIA would use, those could be the ones hosted by Google, Amazon, or other big companies. Sensor providers can also decide to publish their sensors on additional or completely different servers. If the sensor provider resolves to use different servers instead, PIAs have to be able to find them. This could be done by informing PIAs about those servers when other information is exchanged. There is also the possibility of using a form of load distributor knowing lots of servers. PIAs would use this distributor to receive lots of knowledge from servers they do not know. However such a load distributor also receives lots of knowledge and could use this knowledge in a malicious way. After PIAs receive the sensor information, they can verify the sensor is included in logs. When the data is verified, it can be assumed this server is honest.

Monitors should be used to verify the consistency of logs as well as check if the merkle trees' root is incorporated in other known logs. This ensures logs cannot change their views and have to commit to their data for a specific point in time. Because monitors have to be able to read the data, this holds for everyone else as well. This means it might be an option for everyone to search for vulnerable sensors (section 2.2.4) or other useful data (section 2.2.5). Until now, the sensors have not been verified, and therefore there is no trust assessment. Monitors are allowed to invest data further, which means monitors could also have looked into sensors while investigating logs [48]. Only some monitors would have to do this to verify those sensors. The details of this investigation have to be stored somehow with the information. It might also be possible to create additional devices to verify sensors. This means the data might be verified periodically, and it could be hard for a malicious user to perform some attacks. Thus it might be hard to add malicious data (section 2.2.6), or sensors with hostile address data (section 2.2.12). A system like this might also flag compromised sensors (section 2.2.7) if monitors check for them and the system is correctly implemented. However, sensors are not validated by inclusion which means those attacks are possible until a monitor checks for them. This is the case because there are no notaries. It would also be possible to use a web of trust and store which user trusts which sensor, or how many times a sensor is

Table 4.16: Requirements fulfilled by a Software Distribution Transparency approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
Software Distribution Transparency	X	X	X	X	X	X	X	X				X	~	X

used. Another option to add verification at insert would be to include notaries in the form of CAs into the system (section 4.1). What requirements are accomplished when software distribution transparency and auditability are applied is demonstrated in Table 4.16.

4.9 Accountable Key Infrastructure

Accountable Key Infrastructure (AKI) was designed to increase internet security by raising public key security and decreasing the required trust in CAs. AKI also supports important features like swapping out trusted CAs or re-creating key pairs and certificates after losing them [57]. Following the functionality of AKI is explained, the information is mostly taken from [57]. In contrast to current systems, AKI requires additional participants to work together, however, all participants can be expected to distrust each other. The participants required are [57]:

- Server,
- Client,
- Certification Agency,
- Integrity Log Servers and
- Validators.

The server is the entity that should be contacted using the public key. This means the key pair is bound to this server. The client is the requesting entity, requiring the key to communicate with the server. The certification agency has to do the same work a certification authority has to do, but because it has less power in AKI, it is called an agency. The integrity log servers (ILS) are publicly available hash trees. Those trees contain all registered certificates in lexicographic order. Validators are the type of participants expected to check if ILS are valid and managed adequately. Validators have to check for consistency and validate whether updates and policies are respected.

A short use case could be as follows, Alice has set up her server and now wants to set up an AKI certificate. To do so, she sets up rules which have to be followed to gain her keys. In this rule set, trusted CAs and ILS get included, and a minimum number of CAs is defined. After doing this, Alice reaches out to more than the minimal number of CAs to sign her key and the rules she specified. Alice then

Table 4.17: Mitigated attack vectors by a AKI approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
AKI standalone	X	X				X		X	X	X	X	X		X		X
AKI verification	X	X				X								X		X

sends the signed certificates to the ILSs she trusts. Each of those ILSs adds the certificate to the hash tree. Alice downloads the verification information from those ILSs. Each browser used to request data receives this verification information from the ILS from now on when connecting to the server via HTTPS. If browsers receive data, they can verify the key the same way as done nowadays by checking the signatures of the CAs, but additionally, the browser can check the ILS information. To prevent malicious users from creating certificates on their own, and immediately using them, certificates only become active after a domain-specific time. That is in place so the actual holder of the domain could take some action to prevent harmful activities from happening. Also because there is a minimum amount of CAs required releasing new keys is much harder.

4.9.1 AKI for the Sensor Directory

Using AKI for the sensor directory would work in a similar way CT logs work. This means there are two distinct ways of using AKI to allow the implementation of the sensor directory. The first option is to use AKI as a standalone system to store the sensor information. The second option is closer to the initial use case by using AKI to verify the content of an untrusted server.

For the first option, AKI has to be changed slightly. The ILS would store the sensor information instead of the hashes of the keys. Also, those ILSs would have to implement search options to allow PIAs to send requests and send a response with verification information. If those servers would not allow for searches they would respond with the whole dataset and require the PIA to search the data itself. While the first approach requires some additional APIs to be developed, the second approach would require lots of network and computational capacity of the PIA as well as the ILS. In such an approach CAs would act as notaries and would only sign sensors they trust. This would be already a sort of trust however the data would only be verified at entry. Because the ILSs are monitored they cannot get malicious without raising suspicion. Because ILSs run in parallel a redundant server structure is available which prevents fingerprinting (section 2.2.3) and tracks any person by them (section 2.2.14). But this only is true if the sensor data provided on such servers is partially redundant. However, because those ILSs have to be monitored the data has to be public and therefore everyone can gain knowledge about vulnerable sensors (section 2.2.4) or other information (section 2.2.5). See Table 4.17 for an overview of those mitigated attack vectors.

Another clear problem of such an approach is, that PIAs do not know which sensors would be in which ILS, so they do not know if they ever have found all important sensors as long they do not have additional information. Because there

might be a permission exchange with the sensor provider, the sensor providers could also provide a list of ILS that they trust, so the PIAs know which ILS they have to crawl for all the essential sensors. There may be sensors, which do not expect the user to have any permission but still try to verify the identity of a user, that might be for example, in public transport. This means PIAs would still have to crawl additional ILS for this sort of sensor, but it could be an option to concentrate this sort of sensor on several known ILS so the PIAs can limit their required resources. This ILS could be run by countries that are at least a bit trustworthy. When following the rules set, an owner can update their data by following the normal use case and adding new information. In this process, multiple CAs would be contacted. Those would act as notaries, check if updates are valid, and mandatory conditions are set. For example, they would also check if any problems with the address field (section 2.2.12) would occur. However, because the system has many ILSs it might be hard to mitigate those attack vectors because those directories might interfere with each other.

The second option of using AKI, would be for sensor providers to set up their sites and use the ISL as a verification as in the original idea of AKI. This means for each sensor provider there would be a server holding the sensor information. When PIAs connect to those servers they receive this list of sensors from the sensor provider. Those sensors are handled like keys in the original AKI system, this means they are signed by CAs and added into ISLs. Whenever a PIA receives a sensor, the additional information from the ISL is concatenated. The PIA may use this information to verify the data. By including the ISL information there is also no longer the problem of not knowing which ISLs are used by which sensor provider because this information is included in the rule sets of the data and therefore known. Nevertheless, PIAs are still required to know on which servers the information they are interested in is provided. This means the server the sensor provider is using has to be known. This server could also be a server combining lots of data from different sensor providers and supporting searches for location. By doing this the afford required decreases a lot for PIAs, this is the case because there are fewer known servers holding lots of sensors that also support queries. However, this means providers for those servers are needed. While those servers provide the sensor data, this data could be verified by using ILSs. Several such servers are required so the data as well as requests can be distributed to prevent fingerprinting (section 2.2.3) or tracking persons (section 2.2.14). However, such servers do not support immutability, if no data structure is enforcing the server to do so. This means those servers can delete unwanted data. This also means they can delete malicious data as well as solve storage problems. The server is also able to change the stored data. Because the ISL still verifies the original content validators or PIAs can flag such a situation. Regardless, the structure of the ILS does enforce immutability which means the data of those systems will grow indefinitely. It is also possible to verify which information got dropped by a server and therefore if a server is trustworthy. For an overview of such an approach see Table 4.17. Whenever new data is added to a server the data has to follow certain rules. The data has to be signed by multiple CAs as well as get included in ILSs before the data is added to the distribution servers. Using this approach also means only the owner of a sensor can update it. Also, malicious data should not get into the system. However, because those servers distributing sensors are not validated or bound to any data structure, they could show different views to different users. Yet, this might be discovered when PIAs gossip with each other.

Trust for such a system is not that hard to achieve. First of all, CAs would act as notaries and check the data before it is inserted into the system. This means it should not be possible to add malicious or wrong data into the system. Additionally, the system supports validators who are expected to frequently check

Table 4.18: Requirements fulfilled by a AKI approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
AKI standalone	X	~	X	X	X	X	X	X			X	X		X
AKI verification	X	X	X		X	X	X	X			X	X	X	X

the content of the logs. This means those could check the data as well, if any data is malicious those might flag this. To be able to validate this, validators have to communicate with the entity. Validators might then inform the sensor provider or the server about this situation so they may take some action. It might also be an option to have a list of malicious sensors which is filled by those validators so PIAs would be able to query this list and only use sensors not listed by them. However, this might allow for trust to be destroyed so PIAs may not trust those statements. Nevertheless, this might only be needed if the responsible entity does not take action. Because there are lots of validators it is also not possible for the system to split views for different users (section 2.2.17) because those would notice such an action. The requirements fulfilled by using an AKI approach can be seen in Table 4.18.

4.10 Attack Resilient PKI

Attack Resilient PKI, or ARPKI, is designed to solve problems current CA structures suffer from. It tries to allow users to get their certificates signed while not having to trust anyone and not allowing anyone else to create certificates for their domain. Following ARPKI is explained, how it works is taken from [14]. The idea of ARPKI is based on AKI. Therefore the entities of the systems are similar. Additionally, there is also a rule set that has to be fulfilled for each certificate. ARPKI requires the following participants:

- Server,
- Client,
- Certification Authority,
- Integrity Log Server and
- Validators (optional).

The server and client do not differ from those of AKI considering their tasks. They only differ in the content of their messages. The certification authority in ARPKI has additional tasks assigned. It still has to check for ownership and sign certificates as in most other systems, but they have additional validator tasks

assigned, which will be explained in the next paragraph. Integrity log servers (ILS) are, similarly to AKI, hash trees that cannot be changed and are publicly available. Comparable to CAs, also ILS got additional tasks. Besides maintaining a hash tree, they have the chore of distributing new certificates to most other known ILS if they get chosen as trustworthy ILS. Validators can be part of the system, but they are not required. If validators are present, they perform the same actions CAs perform, but they do not sign new certificates.

A short use case could be as follows, Alice is setting up her new website and wants to serve her certificate with ARPKI. To do so, she has to select at least 2 CAs and an ILS she trusts. ARPKI guarantees security, even when n-1 trusted entities are compromised. When a higher number of CAs get selected, the systems security guarantees increase in parallel. Alice creates a rule set, like in AKI, and sends her keys to the minimum number of CAs to sign it. Alice then receives a certificate she needs to add to the system. To do this, Alice contacts one of the selected CAs (CA_1), and CA_1 verifies the ownership of Alice and signs the certificate. Alice now has to wait while CA_1 takes the next steps. CA_1 contacts the ILS, which was selected as trustworthy by Alice, and sends the certificate to this ILS. The selected ILS now has the task of distributing the certificate to a variety of other known ILSs. When the majority of the ILSs agree to add the certificate to the integrity tree, the selected ILS also adds the certificate to the tree and sends an acceptance to the second selected CA (CA_2). This acceptance is proof for CA_2 most ILS added the certificate into the tree. CA_2 now takes the role of a validator for this certificate and starts to monitor ILSs, which are added to the certificate. CA_1 also monitors CA_2 as well as the ILSs. The acceptance is relayed to the client of Alice, so she can provide it as proof if someone wants to connect to her website. Each time the ILS gets updated, all selected CAs download the whole data set and verify the content is valid and contains Alice's correct key. The ILS also connects to the CAs and verifies the tree root those CAs use. If this is done, the domain owner gets informed about the validation information. Furthermore, validators could be used to download and proof the ILS information as well.

4.10.1 ARPKI for the Sensor Directory

In the same way, AKI could be used to build the sensor directory also ARPKI could be used. This means ARPKI could be used in two ways to implement the sensor directory by either using it as a standalone system or by using it as a verification system for additional servers.

If the system is used as a standalone version the system would work the same way AKI would work in its standalone version. The difference is a user adds sensors by sending it to a CA which then relays it to ILSs which then automatically further distribute the data. This means the data is available in a redundant way on multiple ILSs automatically. However this is not only an advantage, this means many ILSs grow rapidly and require lots of storage and if PIAs have to download the whole list and filter the data themselves this means lots of load. Therefore an API that supports adding new sensors and searching for sensors is needed. Additionally, automatic distribution does increase the risk of attacks by someone adding lots of sensors at once and therefore creating a DoS attack (section 2.2.1). This is the case because this would attack all ILSs at the same time and therefore disable the whole system for the duration. Also while in AKI CAs only verify data when inserted, here CAs and validators do have the task of repeatedly verify the content of those sensors. See Table 4.19 for an overview of the mitigated attack vectors.

Table 4.19: Mitigated attack vectors by a ARPKI approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
ARPKI standalone	X	X		X	X	X	X	X	X	X				X		X
ARPKI verification	X	X				X								X		X

Table 4.20: Requirements fulfilled by a ARPKI approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
ARPKI standalone	X	~	X	X	X	X	X	X			X	X		X
ARPKI distributed	X	X	X		X	X	X	X			X	X	X	X

If ARPKI is used only to verify sensor information provided on an additional server, the advantages and disadvantages are the same as using AKI. However here the information is relayed to more ILSs and CAs verify the content of those ILSs. However, the additional effort might not change a lot in terms of the sensor directory. See Table 4.20 for an overview of all fulfilled requirements when using ARPKI.

One problem is if any problems occur CAs and validators can only flag this situation and there is no actual action they can perform. In the standalone variant, there is no action at all. The only approach possible is to create a list of untrusted sensors or inform the sensor provider about the situation. In the second version, CAs and validators could inform servers about this situation. Those servers could either store additional trust information or they could drop the sensor completely after checking. However, because sensor providers choose which CAs they use those trust assessments might not be trustworthy. However because PIAs might not trust each CA, sensors signed by those CAs might not be trusted.

4.11 CONIKS

CONIKS is a key-verification system that could be used to get automated trust establishments, even with untrusted providers. In this section CONIKS is ex-

plained, many of the information provided is from [71]. CONIKS is designed in a way that expects numerous providers to be present in parallel while talking to each other. Each of those providers is expected to have their own distinct user base. While in most key-verification systems clients can verify the correctness of keys, which means if the key of Alice is controlled by Alice, in CONIKS, clients confirm the consistency of keys. That means they check if keys do not change rapidly or unexpectedly and all clients see similar keys. To allow users to check keys and verify they are included in the data set, the data is stored in a merkle tree in CONIKS. This means the data storage is tamper-evident. Additionally, the root of the tree is signed and called signed tree root, or str, which is done to ensure the non-repudiation of this tree. The str can also be seen as a commitment to the merkle tree, and the provider cannot opt out of it anymore. In addition, if a new merkle tree is built, the old str is included in the tree to construct and create a verifiable hash chain. CONIKS provides its users with two security guarantees:

- No unexpected key changes and
- Non-equivocation.

No unexpected key changes mean all changes have to be included in the merkle tree and are verified by the str. This can be ensured by the client, by checking the authentication path given by the system (section 3.5). Non-equivocation means all users see the same data. This is not given until this point because even if the hash chain of the strs is checked, it is still possible another user gets a completely different, yet valid, dataset. To ensure this identity, providers cross-verify each other and collaborate with clients to ensure non-equivocation. Whenever a provider creates a new merkle tree, and therefore a new str, they publish this str into the merkle trees of other CONIKS systems. These providers verify the hash chain to make sure the old str is included in the tree. If a client receives data, they do not only check the validity of the hash chain but additionally connect to other providers and request the str of the used provider. The client compares the strs, and if they match, the client can be sure non-equivocation is given.

If Alice wants to opt into CONIKS, she has to provide her key to any CONIKS server she trusts. If Bob wants to communicate with Alice, his client requests the key at the CONIKS server and performs consistency checks. To be certain all keys are correct, each client performs this check for their keys regularly. This means they notice if their key is not correct. If the provider wants to provide them with a different data set, this is perceived when the str is compared with strs from other systems.

Privacy is an important feature of CONIKS. Therefore CONIKS does not only protect the user information but also how many users there are. Also instead of using the user name as key to find the stored information a user index is used internally. Doing so does prevent the system from leaking usernames. This user index is calculated using a verifiable unpredictable function. CONIKS allows providers to disguise their real user numbers by adding arbitrarily many fake users, which are not distinguishable from real users. That means no one can guess the actual user number or which entry is from a real user.

4.11.1 CONIKS for the Sensor Directory

CONIKS could be used to implement the sensor directory. Because CONIKS applies a merkle tree, the data is immutable and easily verifiable. Also, non-repudiability is ensured, by requiring the server to sign the merkle tree. Additionally, because other servers hold the strs to other servers as well, no server

Table 4.21: Mitigated attack vectors by a CONIKS approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
CONIKS standalone	X	X	~		X	X							~	X		X
CONIKS verification	X	X			X									X		X

can provide different clients with different views. CONIKS would have to be slightly adapted to map-specific sensors instead of the public keys. The different options for the key value mapping are the same as already explained in AADs. However, the location would require some changes. In AAD higher layers get less accurate because they contain the union of the data. Such a convenience does not exist in CONIKS, this means either the data has to be less accurate so the sensor is found more often, or when a request is sent a radius should be required. If this is the case, this means the server has to search the data for this radius. The system may also be used to search sensors committed by a specific sensor provider or might be used to verify sensor data of another server by verifying a hash. To allow for such searches an API would be required which would allow users to interact by searching and adding sensors.

To allow for the sensor directory to be built using CONIKS, it is important to disable some privacy features employed. This is the case because, unlike CONIKS, the sensor directory requires all data to be public. Instead of using an index to increase anonymity, the sensor directory should store the key in clear text. If sensors are found by location this is required because else near sensors cannot be discovered. Also, the way such an index is created is especially important when the sensor provider is used as a key, this is the case because only the name or key should be required to get the index. If it is possible to keep some of CONIKS privacy elements, it might be possible to increase the difficulty for malicious users to gain valuable knowledge. If these security mechanisms were disabled, it might allow users to search for whatever information they like and allow for target discovery (section 2.2.4) and data collection (section 2.2.5). If the system is only used to verify data discovered on another server it might be possible to keep all of those security mechanics intact and use CONIKS as it is. In that event, all data is known which means the index could be found by the server.

CONIKS relies on a redundant server structure, which means it is expected many distinct servers run in parallel. Those should be run by different users, some of them known as big players. This redundancy is required for CONIKS to avoid servers being able to split views (section 2.2.17) for different users because they store the str of the other servers. If not only the structure but also the data is redundant this may also mitigate fingerprinting (section 2.2.3) and sensor directories that try to locate anyone (section 2.2.14). This redundant structure also limits the impact of someone denying the service of one sensor directory. On the one hand, this redundancy is a huge advantage, on the other hand, it requires sensor providers to publish their sensors on multiple servers. That is especially important if one server becomes malicious or is attacked. However, this redundancy also requires PIAs to request data from different servers because they cannot expect one server to contain all relevant sensors. See Table 4.21 for an overview of all mitigated attack vectors.

Sensor providers might update their sensors anytime by adding additional data to the merkle tree. Besides, old data is always available and can be requested if

Table 4.22: Requirements fulfilled by a CONIKS approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
CONIKS standalone	X	~	X	X	X	X		X	~	~				X
CONIKS verification	X	X	X		X	X		X	~	~			X	X

needed. Sensor providers might be able to add any sensor to the system. That is the case because there is no validation which is conducted by the CONIKS servers. Therefore, it is possible to add malicious sensors (section 2.2.6). Additionally, all attack vectors emerging from the address field (section 2.2.12) might still be a problem. In CONIKS, the only validation is performed by the data owners, this may be used to prevent servers from changing data. If the sensor owners also connect to the sensors, they might verify if they are compromised (section 2.2.7). However, if a sensor provider is malicious, they might choose not to flag their own sensors, and therefore hostile sensors (section 2.2.6) are not detected. Furthermore, no trust system is embedded in the technology. That means there is still the requirement for additional trust which cannot be solved initially. CONIKS servers might have to check sensors before they add them into the system to limit the number of malicious sensors in the system. Also, the storage capacity would shrink and DoS by adding sensors would not have as much impact. For an overview of all requirements that could be fulfilled using CONIKS see Table 4.22.

4.12 Contour

Contour is developed to allow for transparency in distributed software package binaries. How it works and its advantages and disadvantages are described next, the information is directly from [15]. While most ledger and transparency systems deal with a quite low amount of data, Contour is designed to work with binaries, which means it has to work with arbitrarily large data objects. The following participants are required for the system to work:

- Service,
- Authority,
- Monitor,
- Auditor,
- Client,
- Ledger and
- Archival Nodes.

The service is the type of actor creating the content of the system, in Contour, this content consists of binaries. This data is then sent to the authority, which

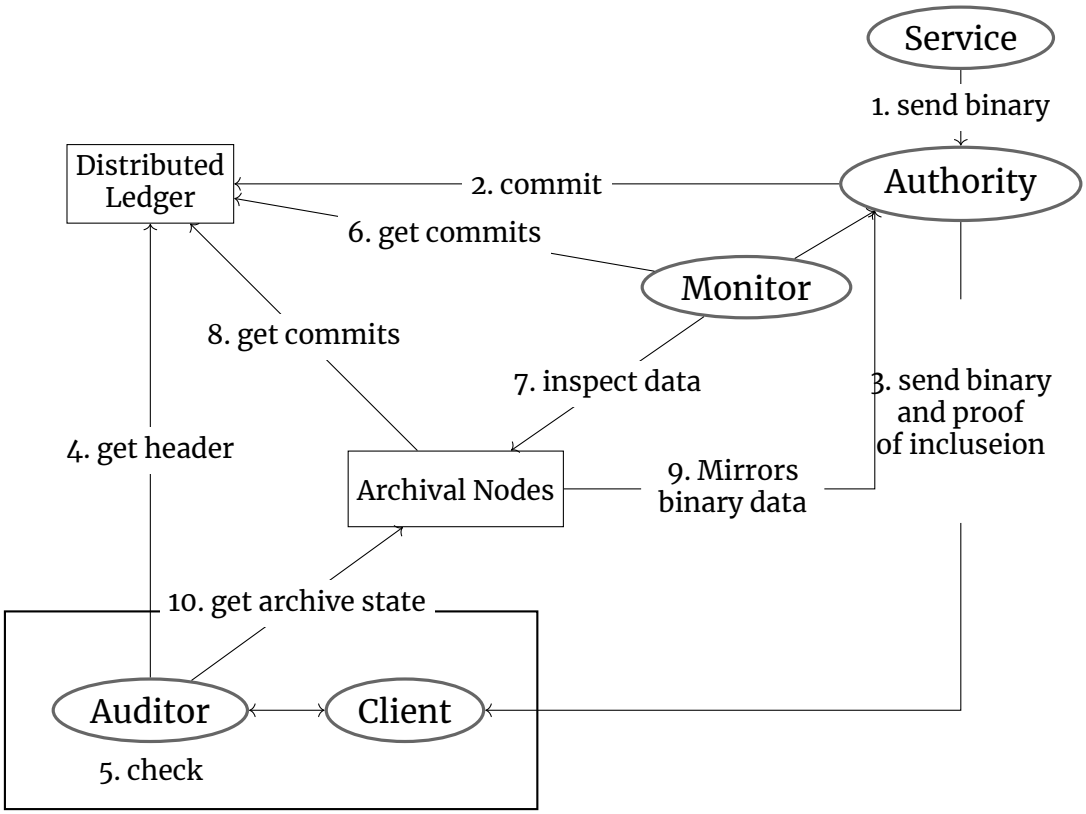

Figure 4.5: Interaction of the different actors in Contour [15]

has the task of publishing the data, so monitors and auditors can conduct their work. Authorities do this by creating a merkle tree. The root of this tree is then committed to public audit logs in the form of a distributed ledger. Monitors scan the data and ensure no malware is included in the binaries. Auditors ensure data that is claimed to be published is public and can be verified. Clients request binaries at services or authorities. The data is publicly available and monitored by unconcerned parties, thus the binary cannot be changed without raising suspicion. Also, monitors can build the binary and verify the content. Contour can use each established ledger available, for authorities to commit their roots into. Because the Bitcoin blockchain is currently the most expensive one to attack, Contour uses it but could also use any other append-only log. By using the public keys of authorities, auditors and monitors can verify the merkle root and therefore verify the view for each authority. Archival nodes are used to keep track of the history. This type of actor is not required, and the system works properly without them. How these actors interact can be seen in Figure 4.5. After an authority creates a new merkle tree and commits its view to a ledger, all clients can request and receive the binaries. Whenever a binary is sent, a proof of inclusion into a ledger as well as information of the merkle tree is transmitted with it. A client might choose to run an auditor on the same machine and verify the binary before executing it. So the auditor reaches out to the blockchain and gains the needed header. Then, the header and the proof of inclusion are compared. In parallel, after the data is included in the blockchain, monitors can reach out to the blockchain, as well as to the authority, and verify the binaries. Also, after data is included in the ledger, archival nodes can

request the data and mirror it.

4.12.1 Contour for the Sensor Directory

Contour could be used in the same ways Transparency logs and similar systems could be used. This means it could be used as a standalone or to verify entries by storing hashes. However, a huge difference between Contour and other systems presented is, that Contour is designed to store much bigger data in the form of binaries. This does mean it might also be better suited to store sensors instead of hashes. Also, it already has to support search capabilities even if they might not match the required search parameters for sensors. Similar to other technologies the redundant structure of the system allows PIAs to use different authorities to get the required data, however, this is only possible if also the data is stored redundantly on some of those servers. This requires the sensor provider to provide their sensors on multiple authorities and PIAs to query multiple authorities at once. Contour also incorporates monitors which are not only expected to check authorities but also to verify the content. This perfectly matches the requirement of the sensors being validated repeatedly. Therefore those monitors would be able to raise suspicion if a sensor acts maliciously. PIAs could also run auditors which would allow them to be more secure by checking the view of each authority in a ledger. If Contour is used to build the sensor directory there is also the need for a secure ledger. There are two distinct options for this ledger:

- **Use an established ledger.**
  By using an established ledger, it can be ensured the ledger is run by many users and also the security of those established ledgers is already known. But in most cases, they already have a purpose and therefore would be hijacked for the sensor directory. That means if any changes do not work with the sensor directory, this will not work anymore. Also, most of these established ledgers are used for cryptocurrencies, which means these currencies are required for publishing data in those ledgers.
- **Set up a new ledger.**
  Setting up a new ledger might decrease the danger of changes in established ledgers. Furthermore, the ledger that is set up may not demand crypto currencies. However, if a new ledger is set up volunteers running the system are required. This means the new ledger is most likely not run by many devices and therefore its security is considerably lower.

When creating the sensor directory with Contour, it is essential to choose one of those options. However, it might also be possible to swap between those options later. This ledger would be used to prevent authorities from showing different sets of sensors to different users and therefore split their views (section 2.2.17). An overview of mitigated attack vectors is given in Table 4.23 while Table 4.24 shows which requirements would be fulfilled by using Contour.

4.13 CHAINIAC

CHAINIAC tries to improve the software update process by changing from a centralized to a decentralized system design [75]. CHAINIAC can do this while still guaranteeing client security, using minimal bandwidth and computational overhead [75]. CHAINIAC also allows simple verification of integrity and authenticity [75]. That is all possible because CHAINIAC uses a skipchain to store all data.

Table 4.23: Mitigated attack vectors by a Contour approach

Technology	Attack Vectors Mitigated														
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Contour	X	X			X	X							X	X		X

Table 4.24: Requirements fulfilled by a Contour approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
Contour	X	X	X	X	X	X	X	X				X	X	X

Skipchains

Skipchains are best compared to blockchains. However skipchains are chains that allow traversing the chain forwards and backward, also each block does not only have one reference to the last block but multiple references to previous once [75]. This structure allows to skip some blocks when traversing the skipchain. The same holds true in the opposite direction, this means blocks also hold references to future blocks [75]. The structure of such a skipchain can be seen in Figure 4.6. Those references to future blocks are added to a block after their creation [75]. However, because this future block does not exist yet when a block is created this reference is not a hash link [75]. Instead, those links into the future are created as digital signatures which are not included in the hashing algorithm for the next blocks [75]. This structure allows much quicker block verification because the blockchain can be traversed considerably faster [75]. This also allows users to compare their chains without the need for a server because a matching block can be found much quicker [75]. It also allows for efficient traversal of arbitrarily long timelines both forward and backward from any reference point [75]. Also to validate a block of the chain instead of all intermediate blocks only a logarithmic number of them is required [75].

Functionality of CHAINIAC

The concept of CHAINIAC is explained next and is taken from [75]. CHAINIAC requires the following participants:

- Users,
- Developers,
- Download center,

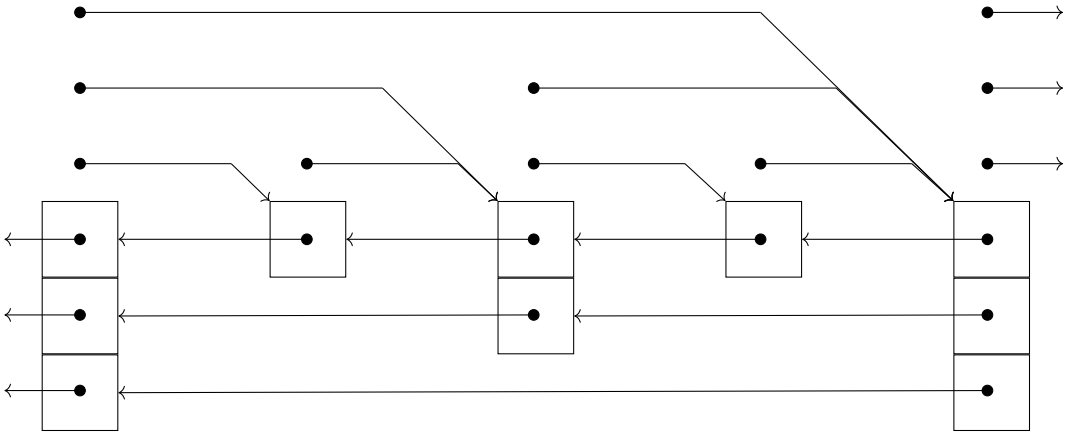

Figure 4.6: Structure of a Skipchain

- Witnesses and
- Build verifiers.

Users are those participants of the system who download new software releases, install and run them. They do not want to trust anyone while still being certain of the connection from source code to binary. Developers are the developers of the source code. Those are expected to act independently of each other, and at least their majority is honest. Developers are expected to prove their source code and sign it. The download center is a server storing the releases and allowing users to download this data. Witnesses are servers chosen by the developers, which should be run by them, as well as by third-parties. They are used to make trusted statements and check if release policies are fulfilled. Witnesses are also required to change trusted keys or policies. Witnesses are not trusted individually but as a whole group. This is possible because after they check a policy they produce a collective signature. This signature is nearly as effective to verify as a conventional signature [34, 75]. Build verifiers are a subgroup of the witnesses. Those are chosen by the developers and used to verify the link from source code to binaries. To do this, they download the source code, compile it, and compare it to the binary delivered by the developers. If those do not match, a malicious action is detected. Build verifiers can be obligated to sign a non-disclosure agreement. All witnesses store a synchronized skipchain. The skipchain stored includes the hash values of the releases and the developer's policies.

Whenever developers want to create software they are required to create a policy, this policy contains how many of them are supposed to sign a new release. Later this number is enforced and only source code with enough signatures is considered valid. If a new binary is created a hash tree containing the source code and the binary is created. Developers verify the source code and sign the root of the tree. The minimum number of developers defined in the policy is required to sign the root. Data and signatures are then sent to the witnesses. They are used to combine all signatures and check if all requirements are met. The witnesses then sign the whole package and add it to the skipchain stored. This is done to improve the performance of verifying the correctness of the release and enforce validity. Build verifiers are used to compile the source code and compare the result with the included binary. For this process to work it is required for the source code to follow the rules for reproducible builds so each binary created matches [61]. This means also the binding between source code

Table 4.25: Mitigated attack vectors by a CHAINIAC approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
CHAINIAC	~	~		X		X	X	X	X	X	X			~		X

and binary is guaranteed. The binary is also published at download centers to enable users to download the new binary. If users download the binary, they validate the content by reaching out to the witnesses and verifying the binary is included in the skipchain and considered valid. If keys are compromised or the developers decide to change their keys for any reason, it is possible through changing the policy and publishing it into the skipchain stored [75].

4.13.1 CHAINIAC for the Sensor Directory

CHAINIAC could be used to implement the sensor directory. Its design allows download centers to hold the data and verification data being stored into a skipchain which is verified. Therefore, there are multiple servers validated by a DLT. This design seems to be quite similar to several other technologies. Examples of similar ideas are transparency logs, Contour, and others. Therefore CHAINIAC also shares most characteristics with those technologies. See Table 4.25 for an overview of all mitigated attack vectors. However while many of those technologies are not designed to validate data stored on a download server and would be tweaked to fit this structure, CHAINIAC is designed with such an idea in mind. Also, CHAINIAC is already designed to work with bigger data in the form of binaries and therefore easily could store sensor data [75]. Also, those servers already have to support some sort of search which easily could be updated to match the needs of the sensor directory. Also, sensors would be verified by witnesses on insert, and build verifiers could be used to further investigate those sensors. However, because binaries cannot change their behavior over time CHAINIAC does not include continuous verification.

CHAINIAC also uses quite a different data structure to most of the other technologies. While most of the comparable technologies use merkle trees, CHAINIAC is using a skipchain. This skipchain might be used to easily compare and find entries in the chain. Also, CHAINIAC does allow multiple developers to work together on one binary, in the same way, it would be possible to require multiple users to work together to publish a sensor. This means a sensor provider would not only be one user but multiple ones, this might be used by big companies for example to make sure their sensors are validated by multiple employees. Also, this increases the trust in a sensor if multiple providers validate it. If the policy of a sensor provider only requires one signature, a sensor provider is still able to act independently. In the same way, an update for a binary would be published also an update to an existing sensor would be published.

Also similar to many other technologies, PIAs would be required to know some download servers, preferred those used by known sensor providers. Also, sensor providers should publish their data on multiple download centers to allow PIAs to distribute their requests and also decrease the impact of failure or DoS. Which requirements are met when CHAINIAC is used can be seen in Table 4.26.

Table 4.26: Requirements fulfilled by a CHAINIAC approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
CHAINIAC	X	X	X	X	X	X	~	X			X		X	X

4.14 The Onion Routing

The Onion Routing, or Tor, is a network privacy technique that can be applied to browse the internet anonymously. To achieve that, thousands of voluntary relays are embedded [50, 54]. Clients encrypt their packet several times. Afterward, the packet is sent through multiple relays, where each relay decrypts one layer and further relays the packet [54]. By doing this, no single node can read the data until the packet reaches its destination. Also, the response packet is sent through the Tor network to protect the users. It is impossible to form a connection between sender and recipient without network analysis, making it an optimal tool for preventing censorship [54]. The problem that has to be solved is how one client can find different relays. This task has to be fulfilled by several directory authorities [50]. Those authorities are hard-coded into all Tor clients and can be seen in [80, 83]. Each new relay registers at all those authorities and only receives traffic after those added the relay to the consensus [50]. Those authorities publish their view of the Tor network recurring in a period of an hour as a single document [50]. Then, the consensus gets created within an hour by collecting the views of all authorities (including their own). The consensus incorporates all relays contained in the majority of views [50]. The authorities' views do not only comprise the relays but also some statistics about them, for example, if a relay is fast or stable [50]. This information gets transmitted into the consensus [50]. Authorities then sign and publish the consensus [50]. They also exchange and distribute the signatures, so each authority provides all signatures, and it is verifiable that all of them hold the same data [50].

4.14.1 Tor for the Sensor Directory

It might be possible to create a system similar to Tor's directory authorities to establish the sensor directory. Meaning, that there would be a need for some trusted parties, which are hard-coded into PIAs. Therefore those parties have to be specified beforehand. That may lead to several selected big players, which earn the most trust throughout the whole world. At any time the majority of those trusted players have to stay honest. When they are selected, they are hard-coded into PIAs, which means if they should ever change, it is required for those PIAs to be updated. There are two different options to use such servers to implement the sensor directory:

- standalone or

Table 4.27: Mitigated attack vectors by a Tor consensus approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Tor consensus standalone	~	X		X	X	X	X	X	X	X	X	X		X		X
Tor consensus discovery	X	X														

- discovery.

The first option is to use those known servers directly as storage servers, the other option is to use them to discover servers of sensor providers where sensors are provided. For the first option, each sensor provider would have to register their sensors at those servers. The servers would check the properties of the sensor before including it, with an option to review connection and validity. That could prevent wrong sensor information (section 2.2.11) or sensors with malicious address information (section 2.2.12). Because a signed list would get published on all of those servers, the whole list is available for all users at any time. That may lead to a lot of sensors being found, implying that everyone can search for potential targets (section 2.2.4) as well as gaining additional information about anyone (section 2.2.5). See Table 4.27 for an overview of mitigated attack vectors using this approach.

After a sensor is provided to one of those servers, one cannot connect to it directly. Those sensors are only available as soon as the next consensus is found, which may take some time. The time depends on the period chosen when designing the system. An option would be to increase the frequency for the sensor directory to find newer sensors faster, however, this does increase the load on those servers. This is the case because for each period new consensus has to be found and eventually all sensors have to be validated. Due to the periodic structure, it might also be feasible for the servers to drop sensors in the next period, if the majority considers it reasonable. This might be the case for malicious sensors (section 2.2.6) or compromised sensors (section 2.2.7). But this also implies if there is a situation where the majority decides to boycott someone, they may drop the sensors of a specific sensor provider. That is also true because those sensors always provide the latest data exclusively, meaning there is no effortless way of receiving old data. Because all of those servers provide the same consensus, PIAs can distribute their requests preventing fingerprinting (section 2.2.3), and tracking people using the sensor directory (section 2.2.14). Since the consensus is signed by all servers, no server can show different sets to different users (section 2.2.17).

Because so few actors are required for such a system to be created, it might be an opportunity to run this redundantly and create distinct systems with different trusted players. However, this might be hard because those servers have to be hard-coded in all PIAs. The low number of servers may make the system a target for DoS attacks of any kind, however, such big players should have enough power to sustain such attacks. Also, attacks by adding lots of sensors are only slightly more problematic than other DoS attacks, this is the case because only a small effort is involved. Servers may check sensors for their connection or validity but they would not have much more work to do. It might be a possibility for those servers to require the signatures of notaries before adding sensors,

Table 4.28: Requirements fulfilled by a Tor consensus approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
Tor consensus standalone	X		X		X	X	~	X			X	X	X	
Tor consensus discovery	X	~	X					X					X	

which might decrease the work demanded from the servers even further. For Tor, the relays as well as their properties are agreed upon, which might also serve as an alternative for the sensor directory. Possible properties for sensors might be similar to those of Tor and might include if it is a fast sensor or if it is stable, how long it is already available, and maybe also a trust value. That may be checked each iteration by those servers. However, if all of those properties are tested for all servers, it would lead to a lot of network traffic, which may not be feasible for those servers. This means it might be an option to do this with a subset of all sensors for each iteration if checking all sensors is not feasible.

After each period a new consensus is found which is published as a new document containing all signatures. However because it is only one huge document this also means PIAs are required to download the whole information at once and perform their searches on their own. This would be a huge storage commitment by the PIA as well as lots of computational power required. To prevent this those servers could additionally provide an API with search capability, however when this is done it is not possible to verify the consensus found. Table 4.28 depicts, which requirements are met when using a Tor consensus approach.

Using those servers to discover download servers would be a different approach that could be implemented. This approach would be much closer to the traditional Tor approach. This would mean each sensor provider would choose one download server where they provide their sensors. If the data has to be checked, is up to those servers. Also the way the data can be queried is up to those servers. This also means in general every sensor and each piece of information can be included. Thus all malicious actions regarding data are not prevented using this approach. The big advantage of this approach is the low resources required for the servers. This is the case because the number of elements is much lower when each download server hosts lots of sensors. However, there are many servers providing sensors so this approach is also distributed. Similar to the other approach also here PIAs would have to download the whole document and search download servers they are interested in. However because the number should be much lower and those servers could be cached, the difficulty can be seen as much lower. Also the actual sensor querying is done by the download server afterwards. Such an approach could also be used in combination with lots of other technologies where servers have to be discovered. Examples of this might be to use such a system to discover servers storing sensors which are then validated using a transparency log, AAD, Merkle[2] or similar approach

to verify the data. Also, those servers might require data to be validated by CAs before they are added to those servers. However, those improvements depend on the chosen server and might not be implemented in all of them.

4.15 Distributed State Machine

Distributed state machines, or byzantine fault-tolerant (BFT) state machines, are distributed and therefore need many nodes working together. By doing so, they achieve high availability and security [59]. They also fulfill reliability, confidentiality, low overhead, and can recover from failures [59]. Also, they can resist byzantine faults (section 3.1) [27, 59]. In such a system all nodes have to be known beforehand and cannot easily be adjusted while the system is running. This system can implement almost every functionality in a distributed fashion, however, the focus of such systems is more centered around computation than data storage [24]. A major disadvantage of this technology is the limited system throughput, for big clusters of nodes [59]. This is the case due to the fact each node has to perform the same actions in the same total order [59]. One possibility to improve performance is to have fewer calculating nodes. If the number of nodes is kept down, it allows the system to gain a very high data throughput, hence the quantity of nodes should be somewhere between 10 and 20 [96]. This is mostly the case due to the communication strategy chosen [96]. This communication strategy is discussed later. To be able to keep the number of working nodes low, the system consists of replicas and clients [59]. Replicas are nodes performing actions, while clients request actions and state changes [59]. Each request from a client always triggers two actions, agreement and execution [59]. Those are closely coupled and may be operated on the same or different devices [59]. In the agreement stage, an agreement protocol is run by the replicas to agree on the order of the requests. This consensus stage has to be byzantine fault-tolerant. In the execution step, all replicas execute the same requests in identical order which was found by the agreement step [59]. The time each node needs for specific actions may vary due to the hardware and architecture of those nodes. Each node needs to finish the executing request i before starting to perform request i+1 [59]. The distributed structure of such a system can be seen in Figure 4.7. Each replica executes requests performed on a state machine which is explicitly built to conduct one service exclusively. This state machine always consists of a state, some variables, and some commands to alter the state. For all non-faulty state machines, it is required to finish in the same state when using the same variables and actions if they start in the same state [59]. Most state machines support any of the following requests [59]:

- Read a subset of state variables,
- Modify or write a subset of state variables,
- Produce an output.

These requests get sent to the replicas by clients and are processed afterward. For the distributed system to work, non-faulty nodes have to fulfill several requirements. These requirements are [59]:

- Deterministic,
- Agreement and
- Order.

For the system to work, all non-faulty nodes need to be deterministic, which means their output only depends on their input and their internal variables

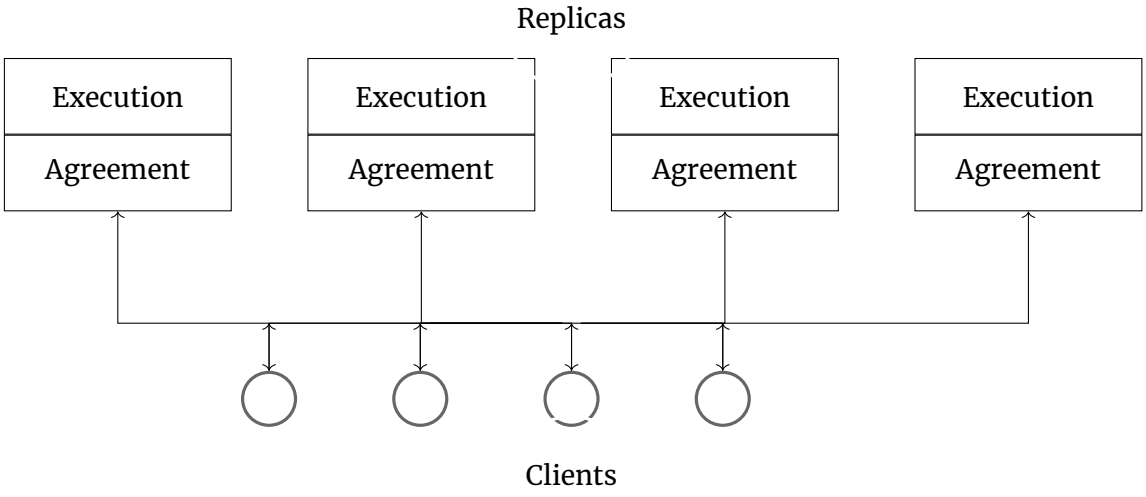

Figure 4.7: Structure of a BFT state machine

which should be the same for all nodes [59]. It is also required for all non-faulty nodes to receive all requests, so they can follow the same steps all other nodes follow [59]. Last but not least, the order needs to be absolute which means each node has to execute each request in the same order [59]. By following these requirements, it can be ensured each non-faulty node finishes in the same final state and produces the same output sequence [59]. It can be shown a BFT state machine implementation can handle up to $\lfloor \frac{n-1}{3} \rfloor$ faulty nodes [27]. This means it is a requirement for clients if they request data to wait until they receive at least $\lfloor \frac{2n-1}{3} \rfloor + 1$ messages with the same content, so the client can be sure this result gets accepted by the majority of replicas [27]. Whenever a client requests data by one node, all will answer because replicas multi-cast the request to all other replicas. This behavior can be seen in Figure 4.8.

During the agreement step, all requests are ordered. One way of getting an absolute order is by using the pre-prepare, prepare, commit steps [27]. To create a total order first pre-preparations are made. This means the request is assigned a sequence number by the first node that encounters the request. The node then distributes the request which is only accepted by all nodes, if those state machines have not encountered this sequence number yet. When the request is accepted by a node, the node enters the prepared state and sends a message to all other replicas to notify them about this step. By doing this, all nodes know how many other replicas accepted the request, and a total order is achieved [27]. After this step, execution is triggered in the commit step, and again all other replicas are informed. Afterwards, the client gets provided with a response set. This sequence leads to a communication similar to the one shown in Figure 4.8.

The main problem of BFT state machines is their limited throughput, caused by the fact each step has to be done on all nodes in the same order and therefore parallelization is not an option. This problem can be tackled by adding a parallelizer into the construct of the replicas [59]. Whether this is possible strongly depends on the agreement and execution steps [59]. This approach leads to a structure shown in Figure 4.9. The parellelizer identifies requests which are

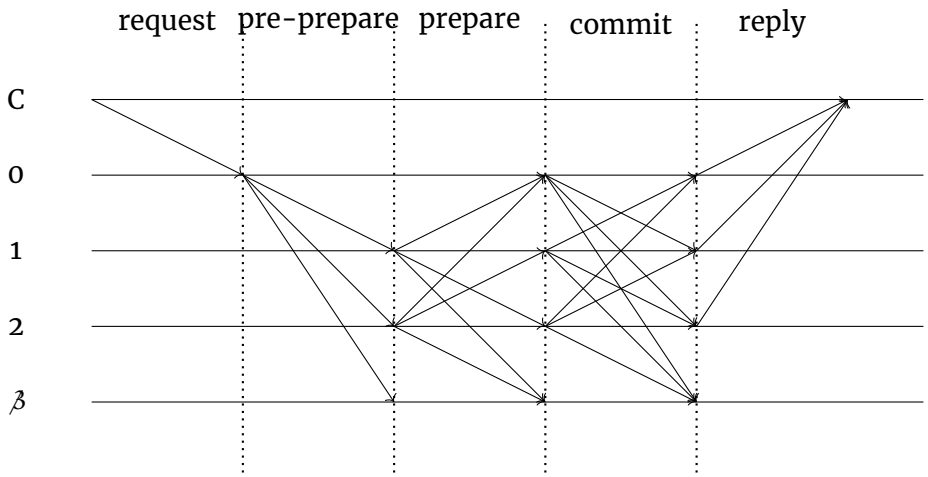

Figure 4.8: Network communication of a BFT state machine

not dependent on each other, enabling their parallel execution. Being independent means one request does not depend on the output of another request or does change any variable needed by this other request. Conducting those non-dependent requests in parallel, should not change anything in the final state [59].

4.15.1 Distributed State Machines for the Sensor Directory

Because for a BFT state machine, all nodes of the network have to be known, if it is used to build the sensor directory big players are needed. Those big players would operate those nodes and would be known and trusted as a unit but not as an individual. Also, the number of nodes is quite limited. Changing participating nodes or their number is quite hard using BFT state machines. All clients would have to support requests to add new sensors as well as to add new ones. Whenever a sensor provider would have to add a new sensor they would send it to a client which would forward it into the system. However, those clients would be responsible for the data relayed and therefore would check the content first. By doing so they can prevent malicious sensors (section 2.2.6) as well as wrong sensors (section 2.2.11). They would also validate only the owner can update sensors. However, because the nodes are hard to change and not trusted it might be necessary this check is performed in the acceptance step by all nodes. If a request is required to find data this request might be pruned and only a finite set of sensors could be contained in the response. To trigger such a request a PIA would send a request to the sensor directory, by sending a request to any client of the directory. The client then requests the data on one replica, which distributes the request to all other replicas. The response is sent back to the client which bundles them and all responses are sent back to the PIA. This means the PIA always knows the view of all nodes but also has to receive the content from all nodes. It might also be an option for the client to bundle those responses and only respond with those sensors the majority agree with. While distributed State Machines are able to solve each issue when employed properly, they also require lots of individualization to match the needs of those problems. The sensor directory is no exception to this rule which means lots of

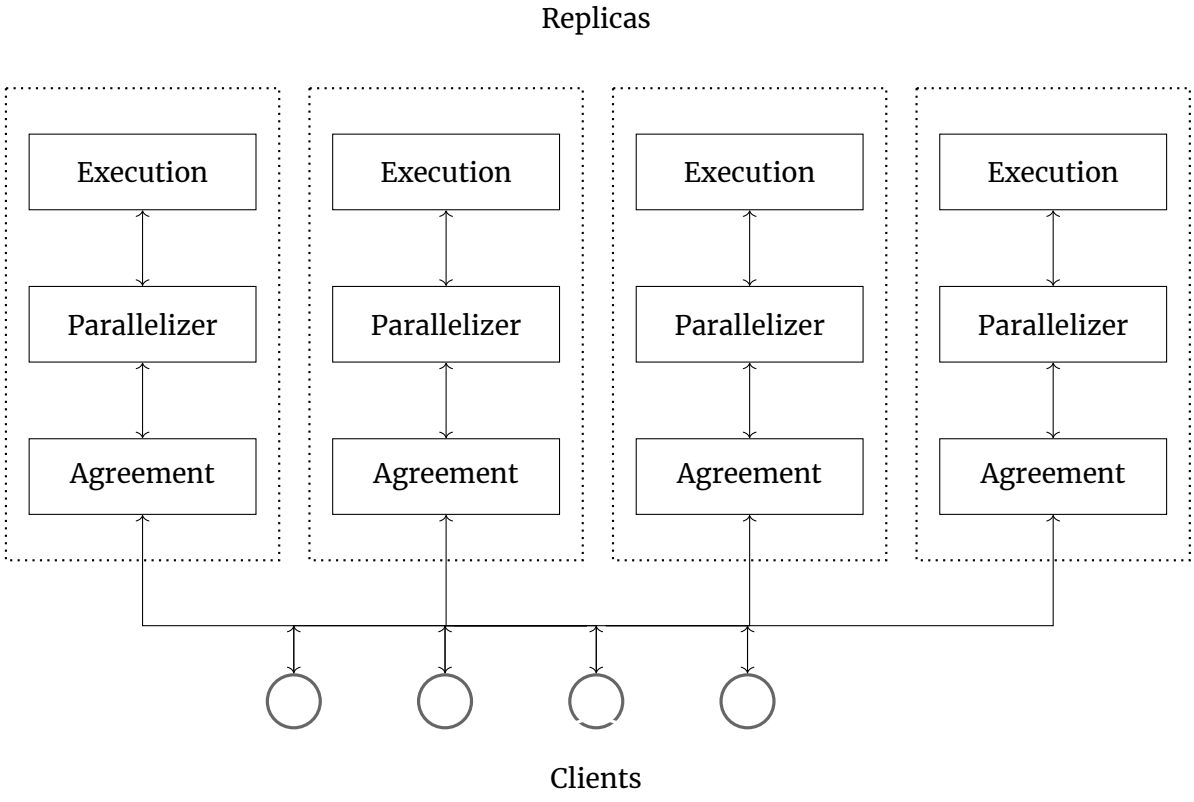

Figure 4.9: Structure of the improved BFT state machine

individualization would be needed to build the sensor directory. Such implementations could be done by using available libraries [59, 67].

Due to the fact the system is distributed via some nodes, PIAs can distribute their requests and mitigate fingerprinting (section 2.2.3), and the sensor directory tracking people (section 2.2.14). Also, no node can split the view between users (section 2.2.17). Because PIAs would use clients to request data their identity is obscured by them. However, because there are so few nodes this might not work as well. Because there are only so few nodes this also means the big players running them are only few and there might be users not trusting the entirety of those big players. Therefore there might be several sensor directories run by different big players trusted by different users. Consequently, sensor providers and PIAs have to use multiple directories to provide and find sensors. This would lead to distinct distributed systems. An overview of all mitigated attack vectors when a state machine attempt is used can be seen in Table 4.29.

BFT state machines do not define the way the data has to be stored. If no immutable data structure is used, the data might be changed by a node. Therefore, the data is not immutable by default and can also not be verified. It might be an option to use a blockchain or merkle tree instead to store the information. BFT state machines are also not designed to work with trust data, therefore a trust assessment has to be added. To allow for this clients would have to support additional requests depending on the trust system chosen. An example would be a request to increase trust in a sensor and one to decrease it. The clients request-

Table 4.29: Mitigated attack vectors by a distributed state machine approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Distributed State Machine	~	X	X	~			~	~	~	~				X		X

Table 4.30: Requirements fulfilled by a distributed state machine approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
Distributed State Machine	~	X	~			X		X			~			~

ing such actions would be accountable for those actions. Also because there are so few clients those have to be selected carefully. Because there are no clients which validate sensors stored in the system it is hard to find corrupted sensors (2.2.7). It is also not possible to add additional validators to the system due to the low count of supporter nodes. In general, a trust system requiring communication might have additional problems due to the communication intense nature of BFT state machines. Depending on the applied trust system, it is possible to mitigate old trust values (2.2.13) or destroy someone's trust (2.2.13). The requirements that can be fulfilled by using distributed state machines are shown in Table 4.30.

4.16 InterPlanetary File System

InterPlanetary File System (IPFS) is a system to store data in a distributed fashion, where everyone can store and download files. How this works is described next and directly inspired from [52]. Everyone who wants to publish data can set up a node and publish their data on that node. Nodes communicate with each other to make data available system wide. This is done by maintaining a distributed hash table. This hash table does store which node stores which information. If a block of information is received by a node, the information is temporarily stored to improve lookup times for this information. This also means if data is requested frequently, more nodes store this data, and the information is easier and faster available in the whole system [19]. Each node can choose to pin a block, which means it remains in the local storage until it gets unpinned and removed. Pinning blocks is also a transitive function, meaning if block A is a file consisting of blocks B and C, and block A is pinned, then blocks B and C are also pinned [19]. Because the system is distributed, there is no single point of failure, and it is not required for the nodes to trust each other. If data gets requested at one node and this node does not hold the requested data, the request is forwarded to another node until the data is found and returned. In IPFS, each node participating in the system can choose to store data, relay

data, or do both.

IPFS is also able to convert data structures like blockchains and merkle trees into merkle DAGs, which are utilized by IPFS [52]. This means the data can be transferred and stored in IPFS. Merkle DAGs function similarly to conventional DAGs (section 3.4). The difference is the nodes of the graph store the hashes of the data, instead of the content. IPFS splits bigger files into smaller blocks, which are then linked in the DAG. Higher layers of the DAG represent a larger junk of the whole file. This structure allows users to download different blocks from distinct nodes and combine them later into the complete file. This also means if different files or folders share information the block is stored only once, leading to less storage consumption. By using this structure, IPFS is also able to represent whole filesystems. For example, a folder is characterized by the hash of the files and folders below. Files and folders are consequently again represented by the files, folders, and blocks they consist of. That leads to a merkle DAG, where each block can be addressed effortlessly by its content. This is also the way used if someone wants to search for something in IPFS.

There exist two options when someone is searching for anything, searching for content or location. In content addressing the search term is part of the searched object. In contrast in location based searches, the query only includes the location the object should be at. In content addressing something part of the object is used to find an object, for example, this might be an ISBN for a book that identifies this book. In contrast, when searching based on location in a library, a person would look for the second book in the third row on a specific shelf. If the book was placed on another shelf the book will not be found that way. However, location based searches are used frequently in computer systems. In IPFS the addressing scheme used is content addressing and the cryptographic hash is used as key to find the desired object. By using the hashes to address the objects, some additional benefits arise [19]:

- *Tamper resistance,* this is the case because the hashes cannot be changed and it is easy to validate the blocks and
- *Deduplication,* which means if two blocks are the same, they have to be stored only once.

If someone wants to access data in the system, they need access to the system itself. This may be done by creating once own node or by using an API. Each node is able to search the distributed hash table and discover which node stores the required data to request it at this node. For IPFS it is also easily possible by using a web browser and connecting to *https://ipfs.io/ipfs/{hash}* where the hash is the one of the searched block.

4.16.1 IPFS for the Sensor Directory

IPFS might be not the best fit to build the sensor directory. The reason for this is the address scheme used. This is the case because discovering something in this system is impossible. This is the case because to receive any information from the system the hash of this information has to be known already. Therefore also the data could be known already. Generally, there are two options IPFS could be used for the sensor directory.

Because IPFS requires the hash to discover the associated data, this hash has to be known to PIAs. Therefore PIAs have to be given the hashes they should be searching for. This might be done ahead of time so PIAs can scan this data. But because the hash is unique to those sensors and changes whenever the sensors change, this does mean also the sensors could be transmitted directly. This

means IPFS would be a distributed storage extension to PIAs. This also means it might be possible for PIAs to just receive all relevant sensors ahead of time. However, such an approach also allows sensor providers to bundle all their sensors into one data object and provide this via IPFS. Doing so would save additional storage for each PIA. Sensor providers could also create groups of sensors and provide different users with different sensors. This could be useful if some users do not need to know anything about those sensors.

Another possibility to use IPFS for the sensor directory would be to use an additional server storing a key and the sensor hashes. This key could be one of several figures, it could be the sensor provider, which would enable PIAs to discover all sensors of one sensor provider. Accordingly, PIAs would only search for sensor providers they also store permissions for. This also enables the sensor provider to create one file storing all their sensors, therefore PIAs would discover all their sensors at once and the amount of entries required is low. Another option for this key would be to use the location. This would create a similar situation to transparency logs where only a limited accuracy should be used or sensors would not be discovered. Using the location as the key would also mean each sensor hash has to be stored and therefore lots of hashes and entries exist. PIAs would use those servers to discover hashes and then use the IPFS system to discover the actual sensors. However, here IPFS only acts as a storage extension to this server. This means this server is the actual sensor directory and has to fulfill its requirements. Nevertheless because IPFS is used, integrity would be guaranteed due to the hash cannot be changed. The initial server could be implemented using another technology explained in this thesis, it might also be any other system that does not fulfill the requirement of integrity.

Independent of which approach is chosen sensor provider would not have an incentive to add wrong or malicious data into the sensor directory, this is the case because this data would be stored on their own devices and would not be discovered regularly. Using the first approach their sensors would not be discovered at all, while in the second approach, it depends on the key used. If the sensor provider is the key their sensors would still not be discovered, if the location is used their sensors may be discovered. Also, IPFS would guarantee integrity for both approaches, however, this also means whenever there is a change in the data the hash changes, and therefore everywhere where this hash is stored an update is required. The content addressing also prevents splitting the view for different users (section 2.2.17) because those could prove the data is valid.

Because IPFS is already a distributed system there is no reason to deploy multiple systems in parallel, rather all participants should be working together on one system. This also ensures there are lots of nodes working together on one system distributing the data as well as the distributed hash table via lots of nodes. This allows PIAs to distribute their requests via lots of nodes and therefore prevent fingerprinting (section 2.2.3) or the sensor directory searching for a person (section 2.2.14). Also because the system is based on hashes, everyone can be kept anonymous in this system.

Depending on the used approach the trust varies a lot. If the hashes are transferred directly from the sensor provider there might be no need for a trust system because this sensor provider is trustful already. If an additional server is required for the system also this server has to hold the trust information because IPFS is not able to do so. This is the case because there would be no way to discover this trust value. This means the server not only has to store key and hash but eventually also a trust value. This server may also check entries at entry or after some time because IPFS does not provide such a feature. This means

Table 4.31: Mitigated attack vectors by a IPFS approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
IPFS	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X

Table 4.32: Requirements fulfilled by a IPFS approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
IPFS	X		X	X	X	X	~	X	X	X			X	~

in IPFS problems like those of the address field are possible. However, if PIAs only discover a known subset of sensors this is not a problem. If a server is deployed additionally this has to be prevented. Using an additional server creates a way for malicious sensor providers to create malicious sensors and get them discovered again. Independent of the approach used, sensors may be compromised, and sensor providers may check their sensors independently to make sure this is not the case. This may lead to a situation where a sensor has to be changed. If the hash of this sensor is stored on an additional server this has to be updated or the trust information is lost. However, this also means if the trust is negative malicious users may use this to drop trust problems and pretend this sensor is brand new. Additionally, the fulfilled requirements are visible in Table 4.32.

4.17 DAT

DAT is a distributed file storage system that supports versioning. It allows all users to publish data and make it available to all who know the necessary information. It can be best compared with IPFS. DAT also uses content addressing, how it works is explained next, and the information is mostly taken from [78]. Also in DAT data is not stored on one single device, it is rather stored on the device of the owner and is available to all other devices in the system via communication. This means similar to IPFS nodes have to discover which nodes hold which information and then communicate with each other to receive this data. For DAT there are three different options to discover which node holds which information:

- DNS,
- Multicast DNS and

- Distributed Hash Table.

While IPFS nodes use the data hash to discover information in DAT the public key of the user is used. These public keys have a length of 32 bytes and are not only used to discover entries but also to decrypt them. This means data stored in the system is encrypted and users have to know who the owner of the data is to decrypt it. Therefore, even if someone can obtain the data, it is only possible to read it when holding the key. To prevent data leaks and increase anonymity the public key is not used directly to search for information, the key is hashed, and therefore only users who already know the key can obtain the information and can decrypt it. However, this also means everyone who holds the key can read the whole data related to this key and there is no way of restricting this access. However, because the system is anonymous nothing prevents a user from obtaining multiple key pairs and therefore creates distinct data objects. One key also can store lots of files within the system. Whenever this is done the system allows users to download the whole data or only selected files.

To publish data in the network a node holding the private key is required. The user can publish data on their node whenever they want to do so. Internal the node creates a merkle tree and signs the root so everyone can verify the data. Whenever a node discovers and downloads data, the node stores the data for a limited amount of time. Each node may also choose to mirror this information and provide it to the system as well. The node itself may not even be able to decrypt the information because for this a key is required. This feature increases the availability of data which is required more frequently than other data. Also like in IPFS it is possible to download parts of the information from multiple nodes and verify their correctness by checking the merkle tree root. The system can also include archival nodes that store data from the past including the corresponding metadata.

Comparing DAT and IPFS, DAT has some clear advantages over IPFS. In IPFS, a new hash is created for each data change, this hash is required if someone wants to find the new data. This is the case because the process of addressing is designed the way it is. For DAT, this is not the case because the address is the public key. This key stays the same no matter how often the data is changed. Therefore, all versions are available at the same link, while for IPFS a new hash is needed to find the updated version or each change that is made [94].

4.17.1 DAT for the Sensor Directory

When DAT is used to build the sensor directory, everyone who wants to release data needs to set up a peer and generate their key pair or needs to find someone publishing data in their names. If those sensor providers publish their sensors, they can publish those sensors to those peers. Because the key is required to create valid data, only the owner of the key pair can create or change the data. But this means DAT is not immutable because data could potentially get changed or dropped. If archival nodes are available, the history is stored and available. However, this violation of immutability is not a problem because only the owner can change data. Each sensor provider can use one key pair to publish all of their sensors or use multiple key pairs to create different data sets not readable to everybody. The sensor directory would also automatically use encryption. If DAT is used, all users would work on the same system and increase its capabilities by doing so.

If PIAs search for sensors, they are required to hold the public key of the sensor provider. This means there is a need to distribute public keys to PIAs that are

allowed to find those sensors. This data could be transmitted whenever permissions for users are exchanged. However, there might be additional sensors that do not require permissions but still need to identify people or verify certain attributes. The public keys required to discover those sensors could be published via a public key infrastructure. It might be a good idea to use already established key servers because the only requirement is to make the public key known.

If a peer that is known not to store information about connections is used, the search process is anonymous [78]. Because PIAs can distribute their requests there is no option for fingerprinting (section 2.2.3) or locating anyone (section 2.2.14). Also, every node receiving information can choose to mirror this data and therefore not only increase the speed of the system but also increase anonymity because the data is available on multiple nodes. A problem however is the limited search capability of the system, this is the case because it is only possible to search for public keys and therefore sensor providers. This means PIAs are only able to search for one sensor provider after another. Also, PIAs are only able to search for known sensor providers and those discovered via a public key infrastructure. When they search for a sensor provider, they receive the sensor data of this provider and have to search for sensors they are interested in inside of this set. Whenever a PIA receives a set of sensors, it can verify them by using the merkle tree. This also means only the data provider has to be trustworthy to trust those sensors. The sensor provider is the only one able to change the data which means if sensors are malicious the sensor provider is accountable. Because the owner of the PIA has added the keys to the PIA, it means there might be no requirement for a trust system. This is the case because the owner already selected only several trusted owners. It might still be necessary to implement a trust system for those sensors discovered by public keys found on public key infrastructures.

PIAs cannot discover new sensors that did not get published at their known public keys. That means users are not able to use sensors that are not included in those sets. There is no need for sensor providers to create wrong sensors (section 2.2.11) because those only consume storage on their own devices and are also never discovered. Also, no other node would choose to mirror this information. A sensor provider might still create malicious sensors (section 2.2.6), but if the PIAs owner did a sufficient pre-selection, these sensors are not found and therefore used. If sensors are compromised (section 2.2.7), this might be not discovered and might be a problem. However, sensor providers should be able to verify their own sensors and if they discover malicious behavior they may swap the sensor or change the data set. The set of sensors one provider has is always limited. Therefore, the result set of sensors is also finite, and the query only allows for specific sensors or a sensor provider. This means the query as well as the response set is always pruned. Because PIAs only discover sensors of specific sensor providers, it might be hard for non-trusted sensor providers to locate anyone by using sensors (section 2.2.15). That means it is not possible to search in the sensor directory for vulnerable sensors (section 2.2.4) or to search for information in the sensor directory (section 2.2.5) because they are required to hold the public keys to analyze this data. While problems with the address field (section 2.2.12) may occur, they might be no issue because for honest users, those problems might not happen and for attackers, their sensors might not be discovered. A disadvantage could be, in case the data is not mirrored by any other peer, an attacker might be able to DoS a peer, and therefore those sensors might not be available throughout the time of the attack. Additionally, a peer holding the private key might be able to provide different users with different response sets (section 2.2.17), which is the case because the peer can create a new, valid data set. An overview of all mitigated attack vectors is available in Table 4.33. See Ta-

Table 4.33: Mitigated attack vectors by a DAT approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
DAT	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	

Table 4.34: Requirements fulfilled by a DAT approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
DAT	X	X	X		X	X	~	X	X	X			X	X

ble 4.34 for an overview of the requirements.

4.18 Algorand

Algorand is a DLT designed to solve problems of other DLT systems, like the enormous amount of computational power required [43]. Algorand is a permissionless PoS (section 3.2.4) solution. Because PoS is used coins are required. However, the core system of Algorand does not demand such, because the core system can be swapped from a major coin to a major user system [28]. Therefore, not the majority of coins decide about the next block, but the majority of users do. That allows the system to consist of many nodes cooperating to find consent. How Algorand works is explained next, the majority of the information is taken from [28, 43]. Algorand is using two distinct messages to communicate. First, there is the common data packet, containing the block and all-important fields. As a second message, there are control messages, which are much smaller and are used to keep the algorithm running. Because the messages have different content, and the control message is much smaller, those messages take different times to propagate the network. The time it takes for the data message to propagate to all other nodes is called λ while for the control message, the time is called Λ. By using those two distinct message types, Algorand is even able to sustain an ultimately powerful malicious user under the condition this user never has control over the majority of nodes. Algorand can resist such types of users while guaranteeing the following attributes:

- All nodes agree on the same block after each round. This also means if the majority of nodes are honest the blockchain never has the issue of forks.
- Each round where the leader is honest, is assured to take a maximum of $4\lambda + \Lambda$.

- Each node is guaranteed to know the block for round r, λ after finishing this round.
- If the leader is malicious, the round is assured to terminate in a specific time.

The algorithm is organized in rounds. Each round leads to a block getting added to the blockchain of Algorand. Each node starts round r directly after the node knows the block of round r-1. There is a leader l and some selected validators (SV) for each round. Both get nominated by cryptographic self-selection. The leader's task is to create a new block containing his candidates for the block for the current round r. The validators have a look into the block and confirm the content. In general, a majority of 2/3 of honest users is needed for the system to work.

Cryptographic self-selection is a method to find a leader and several SVs without requiring any network communication. This is especially important because if any communication would be needed, a powerful attacker would know which node becomes the next leader and could compromise this node, the same holds true for the SVs. Cryptographic self-selection requires the block of the last round r-1. The node adds its identifier to the block and uses an oracle to generate a semi-random object. The oracle is a semi-random function where the output cannot be guessed without the input. In Algorand this oracle is a hash function. The calculated object is then transformed into a number between 0 and 1. Following this, a threshold check is performed. If the threshold is higher than the calculated number, the node is part of the SVs. Also, a second threshold check is conducted, which requires a much lower threshold. If this threshold is higher than the number, the node is considered a potential leader. The node and only the node itself knows if it is a potential leader or part of the SVs.

Each potential leader creates a block containing all of its candidates, signs it, and distributes it, so all other nodes know the block as well. Additionally, the node also sends a control message to all nodes. This control message contains information, so all other nodes can verify the node, is a potential leader. This additional message also allows all nodes to compare all potential leaders to each other. Only the block of the potential leader with the lowest number generated is considered valid and used as the next block. No single node knows which node the actual leader of round r is until they have seen the control messages of all potential leaders. That means powerful attackers are not able to corrupt the node in time to influence their block. Attackers could still corrupt the node if they know the leader‘s identity. However, this leader has already sent its candidate, which means the attacker‘s attempt is useless. To prevent the malicious user from creating a second message with different content the leader uses a cryptographic key pair which is only used once and deleted directly afterward. Even if an attacker can create a new block, only the first block a node receives from a potential leader is considered valid. In theory, users can see the block of round r-1 and add a new node that has an identifier that creates a low enough value to be the next leader for round r. To mitigate this potential attack vector a minimum duration of membership is required. Therefore, each node that is a potential leader has to be part of the network for k rounds (k is a high number, in the standard case, 100). The value k has to be set high enough so the leader of r cannot know who the leader of r+k potentially is. It would also be possible for an attacker in control of the current leader to manipulate a block in a way another malicious node will turn to the next leader. To mitigate this attack vector, the credentials of the current leader are added to the selection of the next leader. Because the leader cannot influence the value of its credentials, the next leader cannot be influenced by the last one.

Similar to the potential leaders the SVs are not predictable. To lower the re-

quired network capacity only a subsection of all nodes is selected as SVs. The SVs are used to find the true leader of a round. That is done by comparing all calculated values by the cryptographic self-selection to each other and voting for the block of the leader with the lowest number. A majority of 2/3 of all validators is required to be honest, for Algorand to work as intended.Comparable to the leaders, the SVs send the signed block and their credentials in two different messages. Also, those SVs create a key pair, which is used once and dropped directly after usage so no one can abuse the node to their advantage.

4.18.1 Algorand for the Sensor Directory

Because Algorand is a DLT, it requires a lot of nodes distributed via numerous locations and parties, if used for the sensor directory. It requires the majority to be honest. The number of those nodes is crucial, because the speed, as well as the security, depends on it. Because Algorand is a DLT, it is also not a very good idea to run multiple instances of the same system, rather the number of nodes in one system should be higher. The DLT also already ensures immutability as well as the system being verifiable. Because PIAs only send requests to nodes of the system and do not use any identifier they are anonyme. PIAs also have to distribute their requests between lots of nodes and therefore even increase their anonymity by preventing fingerprinting (section 2.2.3). Similar to other technologies it is possible to access the data by either running a node on one's own or using an API.

This system would require sensor providers to send their sensors to any node participating in the system. Most likely, they will choose a known and hopefully honest node. This node will potentially become one of the potential leaders and will distribute the sensor to all other nodes. Even if the node is not the true leader of this round, all other nodes are aware of the sensor and may also choose to add it in the next round when they become a leader. It is also possible to select an attempt similar to the one possible for Contour (section 4.12) and use Algorand as an integrity list. This would mean a download server holds the information while the blockchain is used to store the hash values of those sensors. This would allow PIAs to download the sensors from an untrusted source and verify the data using the blockchain. However, this would add additional actors as well as new tasks for PIAs where they have to find the correct download server. PIAs themselves would have two options, either they are part of the Algorand system themselves, which would be rather stressful and would require lots of storage. Or they request data from the system and verify it by confirming the blockchain. However, also this might need lots of storage.

Because Algorand is distributed, the sensor directories is not able to locate anyone (section 2.2.14) or split views for different users (section 2.2.17). Such attempts can be simply mitigated by PIAs if they distribute their requests between different nodes. Because new sensors are distributed to all nodes, this might increase the impact of a DoS (section 2.2.1) attack by adding lots of sensors. The impact could be decreased by nodes checking candidates and adding timeouts for users so they are not able to add an unlimited amount of sensors instead. However, those nodes may be operated by the attacker which means the node would not reject malicious sensors. Also, because the system is distributed timeouts are not an option considering those users can choose to use another node instead. Also, each user might hold multiple keys and prevent this mitigation this way. Such an attack does not only deny the service for a specific amount of time by blocking the network, but it may also increase the load of wrong information (section 2.2.11) in the system and this information has to be stored on all nodes.

Table 4.35: Requirements fulfilled by a Algorand approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
Algorand	X	X	X	X	X	X	~	X			X	~		X

A problem using a DLT for the sensor directory might be, the verification of entries when inserting. This means if the sensor data is checked before the data is included in the directory this step has to be performed by all nodes. This generates lots of calculation power required within the whole network as well as lots of network capacity. For Algorand, this might be a little better because only the leader and the SVs have to perform those actions. The SVs could be used to verify the sensor provider owns the address and mitigates all address problems (section 2.2.12) as well as wrong data being inserted (section 2.2.11). This means those would act as a sort of notaries before the data is inserted. Those notaries would also be able to ensure only the owner of a sensor can update sensors. It would also be an option to add CAs to the system which sign the sensors before they are inserted into the system. Doing so would reduce the stress on the sensor directory and would require honest CAs instead. Another issue with using DLTs is everyone setting up a node has access to the whole data set. Therefore, everyone can search in the data, which allows everyone to find vulnerable sensors (section 2.2.4), as well as find additional interesting data (section 2.2.5). Another big problem of DLT is also not tackled by Algorand. The data has to be stored at all nodes, which increases the required storage capacity drastically. If the data increases to a crucial size, it might lead to many users, except for big companies, to stop running nodes, because the required storage capacity is just too high. This is especially problematic if the whole sensor data is stored in the blockchain. If the data is only used to verify an additional server this is far less of an issue. This is the case because the data would not increase as fast.

While it might be hard to implement a trust system that requires lots of data using Algorand, it might be possible to use a different approach. Initially, it is hard to implement a trust system because this trust information has to be stored in the blockchain, increasing the length and therefore the storage as well as the verification time. However, by the design of Algorand, it might be an option to create a suspicion block including a sensor that seems suspicious. This block would be added by a leader and validated by all SVs. Because those would check the content before this is added into the system they would verify if the sensor is malicious first. If the sensor is malicious, the block is added to the blockchain, and everyone knows not to trust this sensor. If the sensor is not the block is rejected. The same could be done when a sensor becomes valid again. However, the blockchain would always hold the time the sensor was malicious which might prevent users from using it. Algorand supports all initial requirements from section 1.2. An overview via all met requirements can be seen in Table 4.35. Also, see Table 4.36 for an overview via all mitigated attack vectors.

Table 4.36: Mitigated attack vectors by a Algorand approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Algorand		X		~	~	X	X	X	X	X	~			X		X

4.19 Nano

Nano is a DLT, and it is used as a cryptocurrency. Nano's biggest advantage is the high transaction speed, and the nearly unlimited scalability [65]. This is possible because Nano is not based on a blockchain but rather on a DAG (section 3.4) [18]. Nano was developed with problems of PoW blockchains in mind. The goals of Nano were to deal with latency, scalability, and power consumption. Nano is explained next, the majority of information is taken from [65]. Nano was one of the first DAG-based approaches. However, it does not use a standard DAG like those described in section 3.4. Unlike most other DAG-based structures, Nano uses a block-lattice structure. It does not use one data structure including all information but rather has one blockchain for each account created, which is the reason why those blockchains are so-called account chains. These account chains hold the transaction/balance history for the account, and only that account can modify the chain. That enables the technology to update each chain nearly immediately and asynchronously to the rest of the system. The basic structure of such a DAG is shown in Figure 4.10 For further improvement of performance, the transactions are all small enough to fit into single packets. All blockchain technologies need a genesis block containing the start balance, similar to this start block Nano has a separate genesis chain. This genesis chain holds the whole balance at the start, and cannot be increased later. Therefore, all other chains get their balance from the genesis chain in a transitive way. To create a new account in this system someone has to transfer coins to the corresponding public key, which is then the representative of the account. The hardware required for Nano is also minimal because no actual work is involved. Nodes storing the DAG are easily able to validate each block. For a block to be valid, 5 conditions have to be fulfilled:

1. The block has to be distinct.
2. The account owner has to sign the block.
3. The previous block is the last block of the account chain.
4. The account has a starting block.
5. The PoW is fulfilled.

In contrast to most other technologies where only one entry is required to show a transaction, Nano demands two entries. One block is in the blockchain of the sending account chain, the other block is included in the receiving account chain. This structure is demonstrated in Figure 4.10. Because everything is asynchronous, there could be multiple consecutive transactions that cannot be ordered properly. The reason for this could be network latency or other connectivity issues. This leads to the state where the receiving account can choose which block is first within their chain. Forks can only occur if two blocks claim the same block as their predecessor. Because the only one able to create blocks

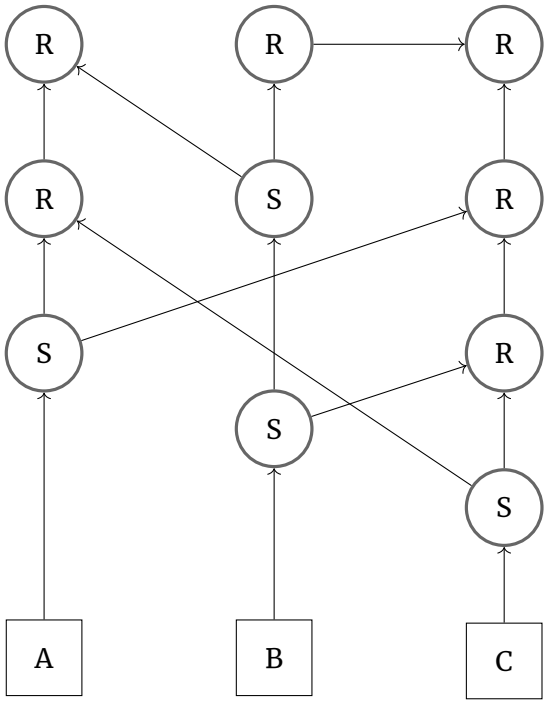

Figure 4.10: An example for Nanos DAG

in an account chain is the account owner, this can only appear if there are programming errors or there is the intention of double spending of a malicious user. If this problem occurs, the system will start a vote and only the winning block stays in the ledger. This scenario is the only scenario where voting is necessary for the system, and only malicious actions from the account owner can trigger this. Nanos voting system is similar to DPoS (section 3.2.5) where each account chooses one node that votes on their behalf. The weight each vote has depends on the balance the combined supporters of the node have.

The power consumption problem of pure PoW systems is enormous, and Nano can drastically decrease this. But Nano also requires each block to fulfill a PoW. This proof of work has to be accomplished by the account owner or their representative node. As already explained in transaction cost (section 2.4.1), the PoW in the system prevents users from spamming the system. This is especially important because, without any fees or other regulations, a user could spam the system by sending transactions between two of the user-controlled accounts and therefore DoS the system. Because the PoW is not required to determine who creates the next block, the work required can be chosen to be much less compared to a pure PoW system. The PoW of Nano is solvable even with little computational power within a few seconds.

Storage is a problem because the amount of data can increase indefinitely. This means not everyone can operate a full node storing everything. Therefore light nodes exist. Light nodes are nodes not store anything relating to the DLT. Light nodes however participate in the network communication and observe several accounts they are interested in. Full nodes in contrast store the whole DAG and are also able to vote on behalf of a user if they are selected as representatives.

4.19.1 Nano for the Sensor Directory

If Nano is used for building the sensor directory, the system is distributed. That means while it is possible to build many systems in parallel, there is no reason to do so. This means the basic structure of the network looks similar to those of other DLTs. This also mean Nano has the same advantages and disadvantages following this structure. In Nano, each sensor provider would create an account and therefore their own account chain, which would be used to publish their sensors. The key would be claimed by adding the first sensor into the account chain. Because each user has its account chain, all sensors added by the same user end up in the same chain. Because they are the only ones who can add sensors to this chain, it would be possible to search for sensors of a defined user by following a specific chain. However, the public key of this user might be needed to do so. This is the case because PIAs have to discover the account chain first. This would also mean when following the account chain updates of sensors are found much faster because the blockchain is much shorter. If a malicious user wants to DoS the system by adding sensors the costs of such an attack add up very quickly because this user would have to solve the PoW. The system itself would not have lots of work to do but the attacker has, which means this attack vector is mitigated.

The system would be run by different nodes, some could even be powerful PIAs which choose to participate in the system. This would add lots of complexity to those PIAs and would not match their intended purpose. All PIAs would query the system using any nodes. Because there is only one system, PIAs could easily search in all chains for sensors. But this also means all data is public and therefore can be searched easily. That allows malicious users to scan the system for potential targets (section 2.2.4), as well as gain additional information (section 2.2.5). Because Nano is using a blockchain for each account, this means the storage is immutable and verifiable. In addition to that, only the account owner can add data. If PIAs are running the system they do have direct access to the storage and could search for data. If they do not host the system by themselves they could send queries to the nodes running the system. It would be an option for those nodes to filter the data. It might also be an option to only allow searches for sensor providers and always return the whole account chain with all their sensors. This would also easily allow PIAs to verify the chain due to the fact it is a blockchain. PIAs could also analyze one account chain after another and receive the whole data set.

There is no trust except the trust in the account owner. It is also not possible to add a trust system. At least not using any structure Nano is providing. This is the case because only the sensor provider can manipulate the account chain. This means it is not possible to add data to a sensor in the same account chain. Therefore, if the owner is trusted, it might be possible to trust the sensor. However, if the owner is not trusted, there is no way of knowing whether or not a sensor is trustworthy. It might be an option for other users to create blocks in their chains and show their trust in several sensors, but PIAs would have to discover these blocks and trust them. This could be done by scanning all blocks in the system however this would lose the advantage to several other technologies. In general, no trust system could be implemented that solves the problems of trust in section 1.2. Also, each user can add data to their chains without the data being verified. This means the data can contain malicious data. It would be possible for nodes running the system to check sensors before those are inserted into the system. However, Nano only requires votes if malicious transactions are discovered. If those nodes would have to check for the validity of sensors before they are inserted voting would be required each time a new sensor is included. Another option would be to add CAs which verify sensors before they

Table 4.37: Requirements fulfilled by a Nano approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
Nano	X	X	X	X	X	X		X						X

Table 4.38: Mitigated attack vectors by a Nano approach

Technology	Attack Vectors Mitigated														
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Nano		X				X								X		X

are inserted into the system. This would mean the data is verified and nodes of the system only have to check if a signature is available. PIAs would then choose themselves if they trust this CA. An overview of all requirements which can be fulfilled when using Nano can be seen in Table 4.37. Also, the mitigated attack vectors are summarized in Table 4.38.

4.20 Byteball

Byteball is a distributed ledger using a DAG (section 3.4). That means a block can reference multiple previous blocks to verify them and could also get referenced multiple times. Byteball is described next, the information is taken from [29]. In Byteball, everyone can store everything, as long as they pay for the used disk space their block needs. To pay the fee for the storage, Byteball has a cryptocurrency bound to the technology named bytes. Meaning, that everyone who has enough bytes, can store whatever they want. This means it is possible to transfer any cryptocurrency, including, but not limited to bytes, and also store other data. To prevent users from exploiting the structure of the DAG by placing malicious blocks in different parts of the DAG, for example, to perform double spending, there is a way of selecting a main chain. A main chain is one chain within the DAG which is then used to decide which blocks came first and resolve problems in the DAG if they occur. In order to determine the main chain, there is a need for so-called witnesses. Those witnesses are known users who are supposed to act like regular users. As long as the majority of the witnesses are honest, all problems can be solved. Each block consists of three different parts:

- **Messages**
  There can be multiple messages which are included in one block. Those

messages contain the whole data. There is also a sub-type of message called payment. This sub-type is used to send bytes or other cryptocurrencies.

- **Signatures**
  The users are identified by their keys, which is the case in most ledgers. Those keys are used to sign the blocks.
- **References**
  Each block can reference multiple blocks. It may reference as many as the user wants it to. This is done similarly to other technologies by including their hashes, however each block has to reference at least one other block.

Blocks are supposed to reference only blocks that are not referenced transitively already from any other block. This means they are expected to use the last blocks they know, which are not yet referred to. Each block is also supposed to transitively reference all blocks created by the same user. This is important to avoid attack vectors like double-spending. All nodes may construct blocks whenever they want to. They may even create blocks in parallel while others do the same. By doing so, many joints and forks are created and therefore a DAG is formed. This means there are nearly no latencies.

The currency nested in Byteball is called bytes. It is strongly interconnected to the storage of the system. A user who wants to add a block has to pay more if their block is bigger. This should require users to think about which data is worth storing and how to manage their data. To encourage users to still reference as many blocks as possible the references are not used for size calculation and it is assumed each block references two blocks, even if they refer to many more. A part of the fee is given to the first user who creates a referencing block on the main chain. But bytes cannot only be used to pay for storage in the system it is also a real cryptocurrency. Therefore, it could be used to pay for everything else as well. The most important reason for bytes to exist is to prevent users from spamming the system. This works because users need the currency to post blocks. Therefore, spamming has a very high cost for the attacker.

Creating order within the blocks is an essential task (see section 3.4 for more information). Creating order is very easy if there is a direct or transitive connection between two blocks. This is the case because there is already a link ordering the transactions. If two contradicting malicious blocks are detected, they both get included in the DAG, but the system will take care of them later. As already explained, a new block has to refer to all previous blocks created by the same user. Blocks of the same user not referencing each other are expected to be malicious. By including all non-referred blocks when creating a new block, a node proves it has seen all blocks directly and indirectly referenced. This means the node proves its view to all other nodes. If blocks do not have a direct link between them it is hard to figure out which block came first. To solve this riddle the main chain is developed. This main chain is one chain chosen from the DAG, from a certain point in the DAG, back to the genesis block. This means a blockchain is selected inside the DAG. To be able to create such a chain, a function is needed which selects for each block the best parent. This function has only access to blocks that are created before the block, the best parent should be found for. For each non-referred block, a chain is created following the main parents back until the genesis block is reached. The established chains are the different main chains for the distinct end blocks. Such a chain can never change, which means once the main chain for a block is known, this chain will always stay the same. It is also important if those chains intersect at any point, they follow the same path until they reach the genesis block. Once the main chain is established, one can create an order for all blocks. This is easy for all elements on the chain because those refer to each other anyways, therefore it is known which one was first. All blocks on the main chain get a

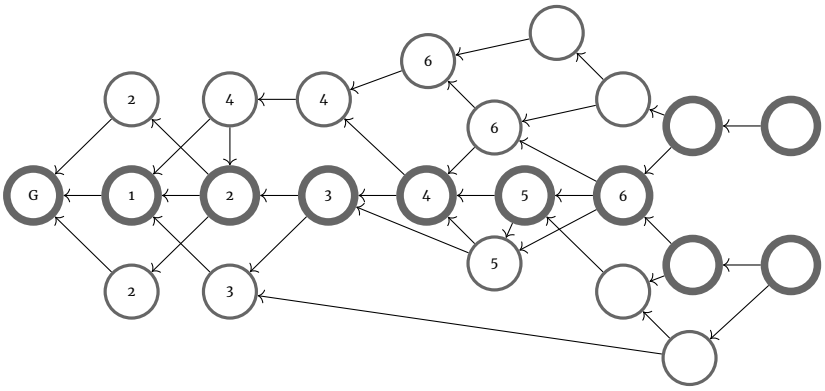

Figure 4.11: Example for the main chain in Byteball

number assigned to them, so everyone knows how far they are from the genesis block. For all blocks, that are not directly on the main chain, the first block of the main chain referring to the block is selected. These blocks get assigned the lowest number of the first block of the main chain they get referred by directly or indirectly. If there is the need to determine which of the two blocks was first, it is only necessary to compare their assigned numbers and decide in favor of the block with the lower number. This procedure is shown in figure 4.11. If both blocks still have the same number assigned, the block with the lower hash is selected as the first. Blocks that are suspected to be malicious are not removed from the DAG because they are still valuable information about the DAGs structure. Instead, the data of malicious blocks is replaced by their hash, and therefore, the data is lost. This is necessary to reduce the required storage. The best parent of a malicious block is expected to be malicious as well. Thus, they may not be trusted either, because they could have been created by an attacker as well. To find the best parents for each block a test is required, for this test witnesses are needed.

Witnesses are users, which are not anonymous, but rather known as individual companies or persons, which are expected to be honest and to post regularly. They also have a personal interest in keeping the system healthy and are more trusted than other users, but it is not reasonable to blindly trust them. If only the majority of witnesses are honest, this is enough for the system to work. To find the best parent for each block, the main chains of the blocks before is followed. In this process, each distinct witness is counted. When the majority of witnesses is discovered the remaining length of the chain is calculated and is called the witness level. Parents with a better witness level are considered to be more real and therefore considered a more suitable parent. This process gravitates the main chain to blocks the witnesses have created, but because they are more trusted this is expected. This procedure also eliminates scenarios where attackers create whole chains with malicious content entirely on their own and then post the complete chain into the DAG. Because no witness was able to post in such a shadow chain, it cannot evolve to the main chain. Therefore, the impact of such an action is limited, and it is easy to find malicious blocks. This also shows why a majority of honest witnesses is required. Serious problems would arise if they would post on the shadow chain and therefore make the shadow chain the main chain. A witness's job consists of regularly posting blocks and referring to blocks they think are more real than others, but they are not safe by all means. Witnesses can be easily swapped by the system's users if they think this is a good idea. Each block may store the trusted witnesses of the creating user if they differ from those of the referenced blocks. By doing so the witnesses

Table 4.39: Mitigated attack vectors by a Byteball approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Byteball	X	X		~	X	X	~	~	~	~	~			X		X

can change over time. If there are a lot of changes in the witnesses, this block is excluded from being the best parent of the next block. The best parent is expected to only change the witnesses from his child by one, which means they have to agree on most of the witnesses. This means over time, it is possible to change all witnesses slowly but steadily. There are always exactly 12 witnesses in Byteball. Those witnesses are rewarded for posting regularly with some of the fees collected from users posting into the DAG.

4.20.1 Byteball for the Sensor Directory

Byteball can be used to build the sensor directory. However, it might not be the best option because it requires its users to pay for used storage with the cryptocurrency bytes. It is also not possible to remove the cryptocurrency from the system. However, there are lots of useful concepts in Byteball that could be used to build a DAG-based ledger for the sensor directory. Because Byteball is a DLT consisting of many nodes similar advantages and disadvantages emerge like those explained in Algorand (section 4.18). While in Algorand a subset of nodes could be used to verify content before this is included in the data structure, this is not possible in Byteball. If a similar approach would be implemented in Byteball each node would have to check the data on its own. While it might be possible to do so, it would create lots of costs for the system. The system itself could be either run by PIAs which would allow them to search in the data, or it could be run by separate devices which allow queries. Both approaches allow for the anonymity of the PIAs. Because Byteball is an open ledger, everyone can see the data and get every information included. This means everyone can search the system for potential targets (section 2.2.4) as well as gain additional information about users (section 2.2.5). Because a DAG is used the impact of many sensors getting inserted at the same time creating a DoS (section 2.2.1) is decreased. However, if enough sensors are added the system can still be downed for a certain time and cannot be used, even more, if PIAs are running the system this would potentially also impact them. However, because bytes are required to add data such an attack would be very costly. See Table 4.39 for an overview of which attack vectors a system based on Byteball can mitigate.

The ledger would be a DAG, including the main chain, which is selected based on the best parent function. Byteball requires witnesses to find the best parents for all blocks. Those witnesses are expected to post regularly to keep the system secure. This would not be realistic because blocks would contain sensors or updates for sensors. However, it is not reasonable to expect those witnesses will add or update sensors in a regular fashion. Therefore, witnesses would have to add empty blocks or blocks with random content to allow the system to establish the main chain. This wastes lots of resources and therefore another way to find the best parent might be needed. However, witnesses could be used to build a trust system. To do so they may reference sensors they trust more, this might

Table 4.40: Requirements fulfilled by a Byteball approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
Byteball	X	X	X	X	X	X	~	X			~	~		X

only be an initial classification. This is because this classification would be outdated after some time. Witnesses could also verify addresses in the process to prevent attacks using this field (section 2.2.12). They could also check for wrong sensor data (section 2.2.11) as well as malicious sensors (section 2.2.6) that are set up. However, this is not only an option for witnesses but for all users. Whenever a user adds a block they may choose the most trustworthy sensors they find to refer to. They may also check transitively and therefore older sensors. By doing so sensors would be checked regularly after some time. Regardless, this leads to a problem, if there is a lot of data the verification step for inserting new sensors increases each time. Also, because this is not done on insertion of the data but when new blocks choose blocks to refer to, new blocks start without any trust assignment. Additionally, if a sensor is corrupted the branch including the sensor will not grow anymore. Destroying not only its reputation but also the reputation of those referring to it. All met requirements can be seen in Table 4.40.

Byteball would also allow saving data in already established systems. This means it would also be an option to use the already established Byteball system to store the sensor data. This is possible because Byteball allows users to store whatever data they want. However, users require bytes to do so. If someone chooses to do so, the witnesses of the system would automatically create trust in the blocks. Yet, problems with the data could not be mitigated or detected. This would also have additional advantages, if users ever lose trust in the witnesses they could be swapped.

4.21 IOTA

IOTA is a distributed ledger and is designed to be used as a cryptocurrency without any fees. The core benefits are being lightweight, and supporting devices from the Internet of Things (IoT), while keeping energy consumption very low [39]. IOTA uses a blockless DAG (section 3.4) which means each transaction is represented by its individual block [84]. In IOTA, each new transaction verifies two previous transactions by referencing them using their hashes [39]. The selection of these referred transactions is essential and therefore explained in the next paragraph. Because the establishment of new entries takes some time, it can be assumed a selected block is already cited by another block. However, this does not pose a problem because a DAG is used, and a block could be referred to by multiple other blocks. Before a block gets appended, a PoW has to be solved, this PoW is the reason why this step is time-consuming. The PoW

is designed to be very simple, so even IoT devices or smartphones can solve them in a reasonable time [39]. The PoW is needed to protect the system from malicious users trying to spam the system with new blocks [39]. By adding a PoW costs are added to each transaction and therefore spamming gets cost-intensive. A similar system is used in Nano (section 4.19).

The selection process of which blocks are referenced, by a newly created block can differ. If a block is selected and an issue is detected in any directly or indirectly verified block, the selection process has to be repeated. This is important to prevent double-spending and similar problematic actions [39]. This means the selection process has to be constructed in a way only one part of the DAG grows while the other ones are left behind, and no transactions are added to those parts [39]. However, all correct parts should end up in the growing part. There are two main possibilities for such a process [39]:

- Random selection or
- Monte Carlo Markov Chain (MCMC).

The first technique is random selection. In this technique, two (or more, depending on the technology) randomly yet not referenced blocks of the DAG are selected. The problem with this technique is the system may generate issues regarding double-spending [38, 39]. Whenever two contradicting blocks exist this means those, or those referring to them can never be selected by one block as references [38]. Whenever such a situation occurs the process has to be repeated. However each block is allowed to refer to one of those blocks but never both, doing so means there will emerge two distinct parts of the DAG which will never merge again [38]. Another option to select blocks to reference is MCMC [39]. When using this technique the user starts at the genesis block and follows each reference to the next block with a certain probability [39]. The probability depends on the cumulative weight of the blocks [39]. The cumulative weight describes how many other blocks refer to the next block directly or indirectly. If a not yet cited block is reached, this block is one of the selected options. The problem with this technique is some blocks may never be reached and therefore never get referred. If malicious users create their own part of the DAG, they need comparable computational power to the honest users, or their part of the DAG will not grow fast enough for honest users to also create blocks in this part of the DAG [39]. To circumvent the growth of only one part of the DAG and also give other parts a chance, the algorithm should not always follow the highest cumulative weight. This is the case because other parts need to be referred to be valid. This means each block has a certain probability of being selected. To allow that, a random factor is included in the probability calculations. If the factor that determines the randomness is chosen too low, the block is always set according to the cumulative weight. Whereas many parts of the DAG never get approved. If the value is selected too high, the same problems of random selection rise, and the possibility of double-spending increases. Therefore it is essential to find a compromise between those two thresholds. Hybrid Selection is the third example [39]. It tries to solve the problem of the MCMC and makes it easier to choose a random factor. It does so by working in two steps which are conducted sequentially [39]:

- Security Step and
- Swipe Step.

In the security step, an MCMC with a low random distribution is performed, which leads to longer chains getting selected more frequently. Subsequently, the second step is initiated, the so-called swipe step. In this stage, a method with a much higher random distribution is applied. Such a method would be a random selection or an MCMC with a much higher random factor. By doing so,

the first step makes sure no double-spending is possible, while the second step ensures no valid blocks are left behind.

The problem is, that the developers of IOTA mentioned IOTA is not yet able to maintain its own [84]. To prevent unfavorable events, centralization is still required in the form of a so-called Coordinator node, which is maintained by the developers themselves. This node is used to generate milestones in the DAG. These milestones are regular blocks, but each transaction validated by a milestone is expected to be confirmed and can be seen as legit and persistent [84]. There are three other types of nodes in the IOTA system [84]:

- **Full Nodes**
  These nodes are standard nodes that distribute transactions. However they do not store the whole history of the DAG, instead they only store the state of the DAG [84].
- **Perma Nodes**
  Perma nodes store the whole transaction history. They additionally work as full nodes [84].
- **Light Nodes**
  Light nodes do not distribute transactions or store any data. Data is solely delivered by the other two types [84]. Light nodes requests information from the other two node types [84].

4.21.1 IOTA for the Sensor Directory

If IOTA is used to implement the sensor directory the structure would be similar to the structure Byteball and Nano would provide. This is the case because also IOTA is a public permissionless DLT and therefore has all the advantages and disadvantages that arise from that. This means many nodes are storing and distributing the data. Everyone is allowed to create additional nodes and participate in the system. Also, every sensor provider can create sensors in the system using their own nodes or nodes providing APIs. The same holds true for queries sent by PIAs. Nodes participating could be additional nodes, PIAs themselves, or a mixture of both. It might also be a possibility for them to be different types of nodes, for example for PIAs to be light nodes sending requests and therefore being part of the system but not storing the whole information or participating in the distribution of sensors.

To decrease stress on perma nodes storing the whole DAG, it might be an option not to store the whole sensor data but rather store hashes of those sensors instead. Such an approach would require some additional servers to store the complete sensor data. This verification approach is similar to those explained in the chapters before and would have the same advantages and disadvantages. See Table 4.41 for an overview of all selected attack vectors that can be mitigated by this approach. Table 4.42 shows the requirements met by an approach using IOTA.

By using IOTA the sensor directory could implement a completely different sort of trust system by using a different approach by selecting the referred blocks. Sensor providers can create malicious sensors (section 2.2.6) as well as wrong sensor data (section 2.2.11). In this trust system, the selected blocks which are referred to are those a user trusts. Therefore, if a user creates a new block, this block refers to trusted sensors directly and indirectly. This may create a much shorter DAG because users may not trust the last blocks added to the DAG but the previous ones. This means blocks may not only refer to not yet referred blocks. Because IOTA is a DAG, it is possible to refer to many different blocks,

Table 4.41: Mitigated attack vectors by a IOTA approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
IOTA standalone	X	X		~	~	X	~	~	~	~	X			X		X
IOTA verification	X	X				X					X			X		

and it is also not a problem if there are multiple blocks without any references. However, for such a selection method to work users need the ability to read the content of the block. Therefore this is not an option if hashes are stored in the system. The system could also include dos and don'ts as is the case in Byteball. Examples of this might be a block that has to refer to all trusted sensors directly or indirectly, and also all blocks of the same sensor provider ahead. This trust system allows users to show which parts of the DAG they trust and which parts they disagree with. However, once a block is included, this block is immutable, and there is no option to revoke the trust. This means if a sensor is compromised (section 2.2.7) the trust is outdated (section 2.2.13). However, if a timestamp is embedded in each block, it may be possible to check if new blocks are verifying a sensor. This situation however creates a much more problematic situation, if users are required to refer to all of their old blocks when inserting a new one because compromised sensors would still gain trust. Because only positive trust could be shown it is not possible to destroy someone‘s trust (section 2.2.13). Also if a sensor is compromised new sensors do not verify this sensor anymore. The problem is also those sensors that have already verified the sensor would not be verified anymore because by doing so a new sensor would also verify the old compromised one. This means compromised sensors would decrease the trust in honest sensors as well. It is still possible to implement every other trust system by adding blocks containing the information required. However, blocks showing trust information to a sensor should refer to the sensor and all other trust entities regarding this sensor to allow for faster discovery.

Milestones might not be required for the sensor directory. This could be the case because there is no truth in the system, which has to be maintained. There is also no risk of double spending because there is no cryptocurrency involved which could be spent multiple times. Sensor providers would be able to create sensors in different branches of the DAG contradicting each other, however, if those are required to refer to each other this is not possible. Also if each sensor has its own reputation malicious ones would still be detected and rejected. Furthermore, each user is using and trusting different sensors, which means a milestone might support the wrong sensors for some users. Therefore, there is no need for a coordinator node maintained by the system‘s developers. However, milestones might still be useful to establish time or to verify sensors. Milestones could be inserted into the DAG referring to all not referred blocks to show at which time those nodes were open. This would show how fast which part of the DAG grows. If blocks include a timestamp this would be useless. Coordinator nodes might also be used to verify sensors are honest and then create milestones referring to all honest sensors. While this might not happen on insert this could be seen as a notary which does this after some time. However,

Table 4.42: Requirements fulfilled by a IOTA approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
IOTA standalone	X	X	X	X	X	X	X	X				~		X
IOTA verification	X	X	X		X	X						~	X	X

this link might not be transitive because this would mean a malicious sensor would destroy trust in later sensors. Coordinators might also check for older sensors already referred by a milestone. But this means such a milestone would have to refer to each block decreasing the effectiveness of this milestone in the first place.

4.22 Fabric

Hyperledger Fabric is a blockchain-based approach of a distributed ledger. Therefore, each node maintains a copy of the ledger, and consent has to be distributed via all nodes. But in difference to many other technologies, Fabric is a permissioned private Ledger, which means all parties, as well as their possible actions, are known. Those parties are not expected to trust each other completely but to work together to reach a common goal. The private structure enables some consensus algorithms a conventional blockchain would not be able to use [6]. The principles of Fabric are described next, the information is mostly taken from [6]. Fabric can include smart contracts in its transactions similar to other technologies. The difference is nearly all other technologies need contracts written in a specific language. While Fabric supports all programming languages and is, therefore more or less the first distributed operating system for permissioned blockchains. Additionally to smart contracts, Fabric also supports multiple channels, where each is its individual blockchain which is managed separately by the technology. Channels can run in parallel and might even use different nodes.

A Fabric network is divided into different node types, which have to work together, those node types are [6]:

- Client,
- Peer,
- Membership service,
- Ordering service and
- Gossip service.

Clients are nodes that request transactions to be executed on a channel. Peers are those nodes holding the ledger and executing the changes. These should make up the majority of nodes in the network. Membership services are the services providing unique identifiers to nodes and also holding them. Ordering services are applied to bring transactions into absolute order and achieve consensus overall transactions. There might be a few of this type, but the number should be kept low. It is important to note those nodes do not hold instances of the ledger. Gossip services are not required for the network, but if there is a high number of peers and several ordering services, their task is to distribute the response of the ordering service to all of them. In general, not a single node in the system trusts any other node, but it is possible to group peers in case they are from the same party. This means those peers trust each other. This option is not available for order services because those have to be suspicious about all other nodes.

The ordering services are those nodes deciding which block is included and when. They could be centralized and represented by just one node, which is most useful for testing purposes, another option is to distribute the order service in a cluster of nodes. These clusters can run consent algorithms like BFT-SMaRt, which accept up to 1/3 faulty nodes, or others [6, 81]. Some of those are no options for public ledgers and allow Fabric to be much more efficient. Because Fabric is private only some known entities can interact with the system. It is possible to use the system in a similar way a mailing service is used [47]. This means everyone else who wants to participate in the system who is not known by the system, sends their transactions to those nodes known and those relay them into the System [47]. The relaying node is responsible for the data, so it may check the transaction first or risk getting punished for the content. This is not given because Fabric has no fixed use case and may be used for any system.

Most blockchains, regardless if they are permissionless or permissioned, follow the structure order-execute, which means a block's transactions get ordered by a node and then distributed to all network members. These network members then execute the transactions sequentially where each node has to conduct them on its own [6]. This structure creates problems, some of which are listed below [6]:

- limited performance,
- non-deterministic transactions and
- consensus is hard coded.

The performance is limited because each node has to perform the transactions and smart contracts on its own one after another. This also means attack vectors like DoS (section 2.2.1) are huge threats. Transactions and smart contracts are provided for most distributed ledgers only in their own programming languages, which are in most cases specially designed to support deterministic statements exclusively. This is necessary because order-execute cannot execute non-deterministic transactions. In Fabric, all programming languages are supported, this can lead to issues but is also an explicit advantage.

Fabric is not following the same steps and therefore, can overcome some of the problems. In contrast to the order-execute structure, Fabric uses a structure of execute-order-validate. By using this structure, a communication similar to Figure 4.12 evolves. To support these steps, each peer consists of two separate parts. The first part is the *chaincode*, this code runs in the execution step. Untrusted developers are able to change this part. The other part is the *endorsement policy*, which acts like a static library for transaction validation, and is therefore used in the validation step. In the execution step, a transaction is sent to a few selected peers which simulate the transaction and respond with a

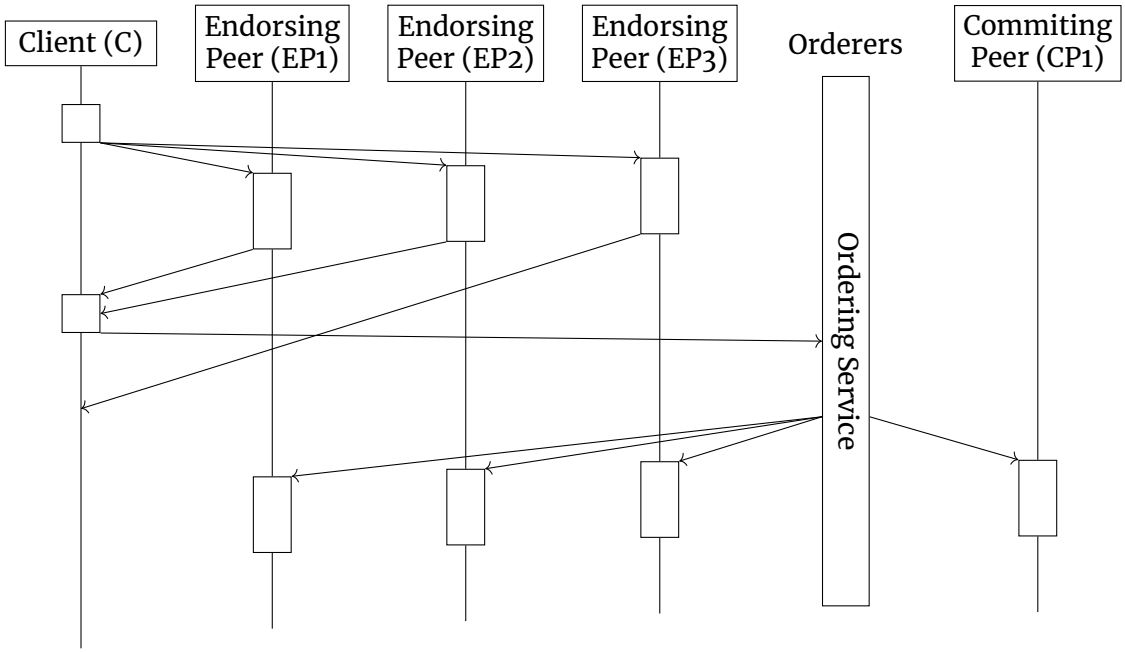

Figure 4.12: Example of the network communication in Fabric

set including which records were written and which were read. The initial node sending the transaction collects a minimum of sets and sends it to the ordering service. By doing so many transactions can be processed at the same time. This also means if non-deterministic transactions are encountered only a few nodes are impacted and if those do not come to a conclusion the transaction is rejected. Peers are also able to stop the execution if a threshold is exceeded. The ordering service uses the sets to establish an absolute order between transactions per channel. If the ordering service is a cluster this has to be done via consensus algorithm, this algorithm can be chosen and depends on the application Fabric is used for. The ordering service then creates blocks which are then distributed to all peers by the ordering service or a gossiping service. In the validation step all policies are checked. All honest peers should come to the same conclusion and add correct blocks to the blockchain.

4.22.1 Fabric for the Sensor Directory

If the sensor directory is based on Fabric, there is a need for some selected known big players who can be expected not to collude. This is the case because Fabric is private and permissioned and therefore not everyone can participate easily. Those big players have to set up the nodes of the system. This means to create the sensor directory lots of big players are required. Everyone else has to send their sensors and queries to nodes run by the big players. Those nodes would check sensors and add them into the sensor directory, this means also those peers and clients would be responsible for them. Therefore the sensors would be checked before insertion. Also, all users could remain anonymous because they send their requests to different clients. While big players do have direct access to the blockchain, no one else has. This means everyone else has to send queries to nodes of the big players. Consequently, users running

Table 4.43: Mitigated attack vectors by a Fabric approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Fabric	X	X	~	X	X	X	X	X	X	X	X	X	~	X		X

nodes, especially peers, would be able to search for vulnerable sensors (section 2.2.4) or other vulnerable data (section 2.2.5) directly. All others may still be able to do so but it is harder for those because they have to do it via queries, which might follow certain rules. Those rules might be part of smart contracts that are used to query the directory and only allow for pruned sets and queries. This does not allow for any identification because the smart contract is executed by a big player. However, to verify a block, the whole blockchain until a certain point is required, this means the data is needed by the PIA. Because the users running nodes are known, it is possible to keep them responsible for their actions. Because the system is distributed between lots of big players, issues such as fingerprinting (section 2.2.3), the sensor directory tracking people (section 2.2.14), or the sensor directory showing different sets to different users (section 2.2.17) are mitigated. This is the case because PIAs can distribute their requests between multiple clients of different big players. However, anyone still can add lots of sensors and try to track a person using them (section 2.2.15). Because Fabric is based on blockchain, the data is immutable and verifiable once the data is included in a channel. However, this also means the storage will grow indefinitely.

Clients and peers of big players would check sensors they receive from sensor providers before they insert them into the blockchain because otherwise, they would be responsible for them. This means the insertion of malicious sensors (section 2.2.6) or wrong sensor data (section 2.2.11) is mitigated. Also, the address field is checked (section 2.2.12). The node would also check only the owner is able to update a sensor by checking the key used to sign the sensor. If the sensor is expected to be malicious it is rejected. The sensor provider can then try to use another node of a different big player to add the sensor to the directory. Because those big players investigate sensors before they get into the system this also directly protects from DoS by adding lots of sensors (section 2.2.1). This is the case because only the checking node would be impacted.

Because Fabric is supporting channels, it might be an option to use another channel to hold trust information. Because a blockchain is used the data is immutable and might take lots of storage. Because all nodes are known, everyone voting is responsible for their action. It is also possible to have different users responsible for different channels which means the users responsible for running the sensor directory may not be those running the trust system. This trust system could be established in lots of different ways. Depending on the applied trust system, it might be possible to mitigate attack vectors where sensors are compromised (section 2.2.7). The trust system may also mitigate outdated trust (section 2.2.13) information or destroy the trust of a sensor (section 2.2.13).

The ordering service can use a wide variety of consent algorithms, for sensors and the trust system. Because there is no contradicting data for the sensors, other than address issues, it might be sufficient to go with a quite minimalistic approach for this algorithm. This may allow for an efficient and fast process. Also because Fabric is a private DLT some different consensus algorithms could

Table 4.44: Requirements fulfilled by a Fabric approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
Fabric	X	X	~	X	X	X	X	X	~	~	X	X		X

be used. An example of such an algorithm is Raft which is lightweight and fast. But also, a lot of other algorithms would be alternatives. For an overview via all mitigated attack vectors from section 2.3 see Table 4.43. Which requirements are met when Fabric is used is shown in Table 4.44.

4.23 Ethereum

Ethereum is a distributed ledger maintaining the history and the overall state within a blockchain [3]. Ethereum can be used as a public or a private version [93]. Ethereum is not only capable of storing data but also computes smart contracts [3, 36]. Those smart contracts consist of code that defines a specific behavior in a specific coding language defined for Ethereum. This contract code is executed for each transaction and has to run on each node whenever it downloads and verifies a transaction [36]. Contracts can be written by anyone and can be stored on the blockchain [23]. After this, they can be called and can use blocks stored in the chain to calculate data, or generate new data [23]. In contrast to other ledgers, in Ethereum, a cryptocurrency is bound to the technology. This currency is called Ether and it is used to pay transaction fees [36]. Under certain conditions, it is possible to disable this cryptocurrency. Ether is required because Ethereum relies on miners, who want something to compensate them for their spending. Miners calculate how much work they have done to execute a specific transaction or smart contract. The result of this calculation is given in gas. The gas value is used to calculate the price for each transaction later [3]. The work miners expend is similar to the effort done by miners in Bitcoin or other blockchain technologies. Miners gather transactions, bundle them into blocks, then perform a consensus protocol, like PoW (section 3.2.3) or PoS (section 3.2.4), and then distribute the block to all other nodes [3]. For private applications, other consensus protocols, like Proof of Authority, can be used [81]. Those protocols also do not require miners and may therefore allow disabling Ether. This is possible because such protocols swap currency based trust by additional trust communication and trust into known entities. The fees in the public version, required to trigger a transaction are also used to prevent users from spamming data into the network and consequently create a DoS. By adding the fees, the cost for an attack of this type should be high enough to prevent malicious users from doing so [36]. Because in the private version, all users are known, there is no need to do so.

In Ethereum, there are two types of accounts able to interact with the system. First, there are externally owned accounts, which are controlled by whoever

has the private key. Those accounts can be used to interact with the system directly. Additionally, there are contract accounts maintained by their contract code, which means those accounts are not maintained by a person. Those two account types do have different functionalities and abilities. For example, an externally owned account can send transactions and is not able to send messages, while a contract account can send messages [36]. Transactions are interactions with the system created by an external account and can be used to create blocks or interact with contract codes, while contract codes only communicate via messages. Each transaction in Ethereum consists of the following parts [36]:

- Recipient of the transaction,
- Signature of the sender,
- Amount of Ether,
- Optional data field,
- Startgas value and,
- Gasprice value.

The recipient, the signature, and the amount of Ether are three pretty ordinary fields for standard cryptocurrency use. The data field is not applied in standard transactions, only if smart contracts are required, this field can be used to fetch additional information needed for these contracts [36]. The startgas value defines the maximum number of computational steps, however that threshold is represented as gas. Each operation has a gas number assigned to it. The miner has to calculate the used gas and must not exceed the startgas value. The gas value estimated is then combined with the gasprice, which is the value the sender is willing to pay per gas, to calculate the fee the sender has to pay [36].

Messages are the packets created by a contract account, which means they are created by code running on nodes and are always followed after some kind of transaction. A message triggers the same steps as a transaction but consists only of [36]:

- Sender of the message (implicit),
- Recipient of the message,
- Amount of Ether to transfer,
- Optional data field and,
- Startgas value.

The startgas value here is the gas of the transaction that triggered the contract code, minus the already used gas by the contract before the message was created. Some Ether is subtracted additionally, this is the case to keep reserves for the contract code to keep running after the message was handled.

Blockchains in general allow for searches from block to block and inspecting each block's content [23]. It is also a possibility to have a look into each block directly when the id of those blocks is known [23]. Ethereum does not only support those options but also has a specific query language which is specifically designed to allow for SQL-like queries in an Ethereum system [23]. The query language for Ethereum is called EQL and allows for querying by using smart contracts [23]. This approach is not the standard for Ethereum and was proposed in 2018. By using the contract code querying the system is possible.

Ethereum's blockchain while being similar to those of other blockchain technologies, is still different. While others only store transactions Tx in the chain,

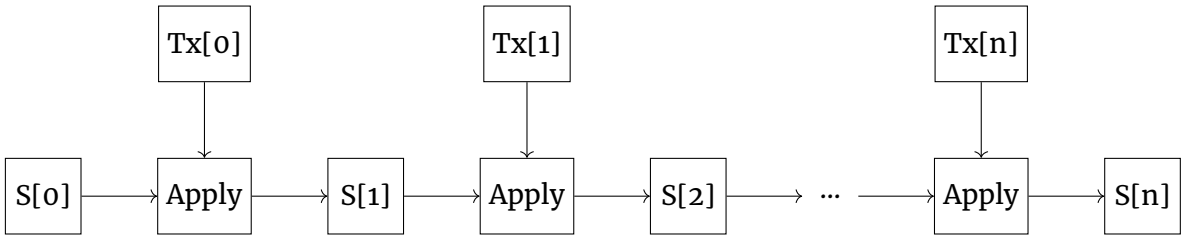

Figure 4.13: The structure of the Ethereum blockchain

Ethereum decided to also include the most recent state S [36]. This state contains all blocks in a combined form which means in terms of currency the balance for each user. This decision changes the structure of the blockchain quite a bit. An example of the structure is shown in Figure 4.13. In order to save storage space the state is stored in a tree-like structure [36]. For each step, only slight changes are made to the tree, and the rest is not touched and therefore stays unchanged. Consequently, the same elements can be used for those parts and can be referenced from an older version. This approach allows nodes to store only parts of the blockchain, while still possessing the whole information [36].

4.23.1 Ethereum for the Sensor Directory

If the sensor directory is implemented using Ethereum, the first question that comes into mind is how to proceed with the cryptocurrency Ether. Because Ether highly depends on the consensus algorithm, this question is also bound to which consensus algorithm should be used. If a public version is applied, the cryptocurrency is essential because the miners require an incentive. This means it is hard to disable Ether in a public version of Ethereum to use it for the sensor directory. If it is applied regardless, the miners require some other motivation. Because there is nothing to offer from the sensor directory, those would be users who are purely interested in the state of the system itself. However, those users could also be those running a private version. Using a private version of Ethereum also allows additional advantages over a public version, besides the option to disable Ether. By doing so, it is possible to run much more efficient consensus algorithms. However, because the system would be private, the users running the system have to be known. Those users would be the same as in other private solutions and might be big countries, companies, or known users, like Amazon and Google, which are interested in a running system. Depending on the consensus technology, those are the users running it, and therefore the majority has to stay honest. This also means the structure of the system highly resembles the structure of Fabric. This means everything following this structure is the same. PIAs and sensor provider would send their queries and sensors to nodes which verify them and then perform the actions. On top of that, those nodes are responsible for those actions. Also, all users not directly running the system would be anonymous. Additionally, the access to the data would be similar where big players have direct access and others would use those to gain knowledge.

Because Ethereum is using a blockchain all advantages and disadvantages following from that apply to the sensor directory if it is used. This means data is immutable and verifiable. However, the blockchain of Ethereum is slightly altered as discussed before. This change may lead to a longer chain due to the

Table 4.45: Mitigated attack vectors by a Ethereum approach

Technology	Attack Vectors Mitigated															
	2.2.1	2.2.3	2.2.4	2.2.6	2.2.7	2.2.9	2.2.11	2.2.12	2.2.12	2.2.12	2.2.13	2.2.13	2.2.5	2.2.14	2.2.15	2.2.17
Ethereum	X	X	~	X	X	X	X	X	X	X	X	~	~	X		X

state being stored. This state is used to store a summary of the blockchain which in the case of the sensor directory could be a summary via all sensors and their state. If data is requested from the sensor directory, it might be a possibility to respond with the state and require the PIA to search inside it themselves. This would decrease the load on the sensor directory while increasing it for PIAs. If PIAs are interested in a huge amount of sensors anyway there is a storage decrease by sending the state instead of the blockchain. This is the case because only up to date data is included and deprecated data is trimmed. This function increases its value each time users update their data because the old data is no longer in the state. The blockchain could then be used to verify information.

As already explained nodes of big players would check sensors at insert and prevent several malicious actions by doing so. However, it might also be required to add a trust system where sensors are validated regularly. Because the system would be private, validators would be known entities. Those validators could crawl through the state and check if those sensors are honest. While it is not yet decided which trust assessment would be the best, for Ethereum, there might be some which are much better than others. Each trust assessment that can be summarized into a single value might be a good choice for Ethereum because this accumulated value could be shown in the state. An example would be validators increasing or decreasing a trust value of a sensor each time they crawl it. This data has to be stored in the blockchain as well. While the information may be very small, the data still adds up. This also would mean to receive a final trust assessment the whole blockchain starting from the sensor has to be scanned. However, this task could be relieved by storing the accumulated trust value in the state. This decreases the work required to find a trust value while still making the validators accountable. Therefore destroying the trust of any sensor is not an option for a validator. However, because the data is stored in a blockchain some trust data might be outdated. Following such a structure the sensor directory would support a wide range of trust systems.

As for all blockchain technologies, it is required for some nodes to store the whole blockchain. That should be no problem because the nodes are provided by big players who have the resources anyway. However, for smaller providers, it might be sufficient to create partial nodes that store only a part of the blockchain while still maintaining all information. This is possible because the state is stored in the blockchain. However, this means such nodes are not able to verify all blocks while still holding all information. This is not possible for other blockchain technologies. See Table 4.45 for an overview of all attack vectors and which ones could be mitigated using Ethereum. For an overview via the requirements of the sensor directory which might be fulfilled by a version based on Ethereum see Table 4.46.

Table 4.46: Requirements fulfilled by a Ethereum approach

Technology	Anonymity	Queriable	Equality	Immutable	Verifiable	Whom to Trust	Trust System	Redundancy/ Distributed	Prune Queries	Prune Sets	Input Validation	Repeated Validation	Storage Requirement	Owner can change data
Ethereum	X	X	~	X	X	X	X	X	~	~	X	X		X

Chapter 5

Comparison

In this chapter, all the technologies from chapter 4 are compared with each other and it is highlighted which are best suited to build the sensor directory. The technologies are best clustered by their architecture. There are six different architectures used by the technologies:

- Integrity Server,
- Audited Server,
- Multiple Servers using a consensus algorithm,
- Network of self-maintained Servers,
- Distributed Ledger and
- Private Distributed Ledger.

Integrity servers (IS) are single servers that do not require other servers to work except those creating integrity information. Those integrity servers not only store the data provided but also store additional integrity information to ensure the data cannot be changed by unauthorized parties. Those servers use third parties like CAs to create this information and store it with the data. However, because those servers are only storing the data and the integrity information, there is no link between the entries, therefore the server has the potential to drop data without leaving a trace. Also, each key owner can sign new data and replace data because it might not be possible to verify which data is the latest, except if some kind of time stamp is appended to the data packet. Technologies that are based on this idea are CAs (section 4.1), web of trust (section 4.2), DNSSEC (section 4.3), and SUNDR (section 4.4).

Audited servers (AS) are servers, that operate one server using a data structure that is verifiable and immutable and several auditors are utilized to verify the server's honesty. For most of those technologies, everyone can run an auditor. Because auditors verify the server which stores the data, this server cannot act maliciously in any way. However, this architecture requires someone to run the auditors, which are based purely on volunteers. A PIA receiving data not only has to contact the audited server but also at least one auditor it trusts to be honest. This is important to make sure the server is honest. Technologies using this architecture are transparency logs (section 4.5), AADs (section 4.6), Merkle^2 (section 4.7), Software Distribution Transparency (section 4.8), AKI (section 4.9), ARPKI (section 4.10), Coniks (section 4.11), Contour (section 4.12) and CHAINIAC (section 4.13).

If **multiple servers use a consensus algorithm (MSC)**, this means multiple known servers communicate with each other and find a global truth which is then provided by all of them. However, all servers have to be known beforehand and it is hard to change them. PIAs may choose which server they use to receive data. Candidates using this architecture are the Tor consensus (section 4.14) as well as distributed state machines (section 4.15).

The architecture **Network of self maintained servers (NSMS)**, is used by IPFS (section 4.16) and DAT (section 4.17). This architecture means all sensor providers set up their servers. Then each sensor provider provides their individual sensors at their servers. Those servers communicate with each other, and if a request is acquired, the data can be received. However, for those technologies, the user is required to already know something about the requested data to send a request. What information is needed, depends on the technology. Because this information is used to search the system for the information this might also allow for tweaks that may better fit certain use cases. This architecture allows for many request sinks while the data is stored only on the minimum number of nodes. This is the case because the request can be sent to any node and is stored by the sensor provider and potentially additional nodes that choose to do so.

Distributed Ledgers as well as their private alternatives are already explained in section 3.2. Obviously, private solutions cannot support anonymity for all their users because those running the nodes have to be known. However, only those can directly interact with the system. Consequently, for this architecture, the players operating the system would act as gatekeepers deciding which sensors are honest and should be included in the system, and which are not. Because such technologies trust the majority of their users to be honest, these players should be widespread between different countries, ethnicities, and political understandings. For some DLTs not the majority of users but the majority of computational power or coins is important, for those systems those should be distributed evenly. The distributed structure would allow everyone to participate by sending their requests and sensors to those big players. If one of them is against a specific ethnicity, it is possible to choose another party and commit the request or sensor there. The nodes of those parties then relay the request or sensor into the system and respond accordingly. While those big players are not anonymous, this is not a problem because if they are known they can be held accountable. For the public solution, everyone is allowed to participate, and as long as the majority is honest a valid state can be achieved. The majority required depends on the technology chosen, possible majorities are the majority of users, computational power, or coins. Due to the high distribution of such systems, much communication may be required. Also for such systems, lots of nodes are required to store the whole data, therefore requiring a huge amount of storage.

In Table 5.1 all technologies are mapped to their belonging architectures. In addition, Table 5.1 also shows how many nodes actually store the provided sensor data. For many technologies this is obvious. If the data is stored on one server and is audited by others or provides integrity information, the data is obviously only stored on one server. If the data is distributed via a lot of nodes, as is the case for DLTs, all nodes have to store the sensor information. However, there are technologies that fall out of line, those are IPFS and DAT. This is the case because in general the information is stored on one node but each node can choose to store the data as well. Obviously, this also highly impacts the memory consumption required by the whole system distributed via all nodes. If only one server has to store the sensor data n, this leads to a storage consumption of n via all nodes of the sensor directory. This is for example always the case for technologies using IS or AS as their architecture. However, sensor providers may choose to use multiple servers to provide their data when using those technologies, therefore the storage consumption for the whole system is in fact higher than n. All technologies require some sort of metadata, an example of this is the hash values needed to build a merkle tree or a blockchain. Because this metadata is significantly smaller than the actual sensor data this data is neglected. IPFS, as well as DAT, allow nodes to additionally store data

Table 5.1: Comparison of all technologies from chapter 4

Technology	Architecture	Storage location	Memory Consumption Whole System	Memory Consumption Each Node	Memory Consumption PIA	Trust	Request Sinks	Integrity	Request Requirements	Setup Requirements	Distribution
Certificate Authority	IS	single node	n	n	1+I(1)	CAs	1	Signature	-	CAs	Trust
Web of Trust	IS	single node	n	n	1+I(1)	Selected Users	1	Signature	-	-	Trust
DNSSEC	IS	single node	n	n	1+I(1)	Server Owner	1	Signature	Domain	Root Servers	Trust
SUNDR	IS	single node	n	n	1+I(1)	Data owner	1	Blockchain	user	server	-
Transparency logs	AS	single node	n	n	1+log(n)	Auditors	1	Merkle Tree	-	-	Trust
AAD	AS	single node	n	n	1+log(n)	Auditors	1	AAD	-	public parameters	Trust
Merkle2	AS	single node	n	n	1+log(n)	Auditors	1	Merkle Tree	-	-	Trust
Software Distribution Transparency	AS	single node	n	n	1+log(n)	Auditors	1	Merkle Tree	-	APT server	Trust
AKI	AS	single node	n	n	1+log(n)	Auditors	1	Merkle Tree	-	-	Trust
ARPKI	AS	single node	n	n	1+log(n)	Auditors(CAs)	1	Merkle Tree	-	-	Trust
Coniks	AS	single node	n	n	1+log(n)	Sensor Provider	1	Merkle Tree	-	-	Trust
Contour	AS	single node	n	n	1+log(n)	DLT	1	Merkle Tree	-	DLT	Trust
CHAINIAC	AS	single node	n	n	1+log(n)	Majority	1	Skipchain	-	-	Trust&Storage
Tor consensus	MSC	all nodes	n*N	n	n+I(n)	Majority	N	Signature	-	Big players	Storage
Distributed State Machine	MSC	all nodes	n*N	n	n	Majority	N	-	-	Big Players	Storage
IPFS	NSMS	single | several nodes	n - n*N	provider sensors - n	1	Hash	N	Hash	Hash		Location Information
DAT	NSMS	single | several nodes	n - n*N	providers sensors - n	1	Sensor Provider	N	Encryption	Public Key		Location Information
Algorand	DLT	all nodes	n*N	n	n	Majority	N	Blockchain	-	-	Storage
Nano	DLT	all nodes	n*N	n	n	Majority	N	Blockchain	- | Sensor Provider	-	Storage
Byteball	DLT	all nodes	n*N	n	n	Witnesses	N	DAG	-	Start Witnesses	Storage
IOTA	DLT	all nodes	n*N	n	n	Majority	N	DAG	-	-	Storage
Fabric	PDLT	all nodes	n*N	n	n	Majority	N	Blockchain	-	Big Players	Storage
Ethereum	PDLT	all nodes	n*N	n	n	Majority	N	Blockchain	-	Big Players	Storage

| ... semantic or, either the one side or the other

from other nodes. However, it cannot be grasped how often this feature is applied. That means it is guaranteed, that the data is stored once by the sensor providers node, but theoretically, the system can store the data on each node. For DLTs, as well as for MSC, all servers (N) have to store the data, and it is required for a majority to hold the same data. Nevertheless, while technologies using the architecture MSC only store actual data, those using blockchains or DAGs also have to store deprecated data. Some DLTs also support the ability to store a smaller part of the data, this is not represented by the data in Table 5.1 due to the fact it cannot be estimated how much of those nodes would exist.

The persistent storage required on a single node also depends on the technology, this is also shown in Table 5.1. In this context, a node is the server or node storing the data. For IS and AS systems those are the servers storing the data while for technologies using an architecture of MSC, NSMS, or DLT those are the nodes communicating and storing the data. While all DLTs, private and public, have to store all data on all nodes this might not be true for all other technologies. It might not even be true for all DLTs, this is the case because, for some of them, there are nodes that support trimmed datasets. However, those are not the main nodes of the system and are therefore not those represented in Table 5.1. Because DLTs are based on immutable data structures this does mean also all deprecated data has to be stored for all nodes storing all data. Also, Tor requires its nodes to store all data because the data is the same on all servers. However, the stored data can change for each iteration. For IPFS and DAT the required storage varies a lot, it may be as few as the sensor data of the sensor provider setting them up, or the whole data of the system if they choose to mirror the whole system. For all other systems, to be more specific those having the architecture of integrity servers or audited servers, the required storage highly depends on their reputation, their trust, and the party running them. This is due to the fact sensor providers have to choose which servers they choose to distribute their data and therefore those servers would gather more users. Sensor providers are most likely to do this on multiple servers which means the consumption overall might be higher but one single server might never hold all data. This also means it is hard for PIAs to discover the whole data set because it is never known by one server. Sensor providers will most likely choose those servers they know to be trustful, are run by known entities, or have earned a good reputation. This means those servers will hold much more data than servers not known by as many users. However, this also increases the relevance of those servers. Yet, PIAs are also able to find data much easier. Therefore, data in those systems centers around such servers.

Also the storage required on PIAs might vary depending on the chosen technology. This is especially the case if the PIAs do not only require data but also have to verify this data. While for some technologies the sensor data and the integrity information should be sufficient, others can require PIAs to hold nearly all data if they want to verify the information. For integrity servers, the requested sensor data and the integrity data are required. The requested data is of fixed size and therefore represented by 1 in Table 5.1. The additional integrity information required is represented by I(1). Audited servers do allow for multiple steps of verification. Because many of them use merkle trees, it is possible to validate the data by using additional log(n) data from the tree. However, because those are also audited it is possible to compare the root of the merkle tree with the data of an auditor as well. AADs do support very fast verification because the whole data set is stored in their root, nevertheless, this means if the root is transmitted the whole data has to be stored on the PIA. Blockchains also allow verification but to do so it might be required to hold a huge part of the blockchain. This means it might be possible nearly the whole blockchain is required to verify a sensor. However, it might also be sufficient to

send the same request to multiple nodes and compare the responses due to the distributed nature of such systems. Such an approach is possible for all technologies where servers hold the same data. Skip chains are compared in a similar way blockchains are, yet, it is much more efficient to compare and therefore validate those. IPFS uses a hash function which means it is easy to verify the data. For IPFS the sensor provider should include a signature to allow for validation of the data as well.

Obviously, it is also important whom a user has to trust to be certain that the sensor directory can be expected to be honest. This is highly dependent on the technology itself. Generally, for integrity servers, it is required to trust the entity providing the integrity for the sensor data. For the majority of audited servers, there should be some auditors which are trusted by the users and are used to verify the response from the server. However, these auditors may vary from user to user. This concept is not the same for all audited servers but in most cases, there is some auditing entity that can be used to verify this information. When multiple servers use a consensus algorithm to find a global truth, which and how many nodes have to be trusted is dependent on the consensus algorithm. The same holds true for DLTs which are highly dependent on the used consensus algorithm. Again IPFS and DAT are a little bit special due to them using content addressing. In IPFS it is not possible to change the data due to the fact a hash is used to address the data. Because the content is addressed using the public key in DAT, only the sensor provider may be able to change data and therefore has to be trusted.

The technologies also vary in the number of potential request sinks. A node is described as a request sink if a request can be sent to it, and a valid and correct response is sent back. For most of the technologies, the number of request sinks and the number of nodes storing the data are the same. This means all nodes storing the data can respond to a request. However, some technologies require an API extension to allow for such responses. However, because IPFS and DAT are using another structure this does not hold for those technologies. In those technologies, the data is at least stored once but can be requested at each node because those nodes are able to communicate with each other.

Furthermore, the way those different technologies achieve integrity, or for some immutability differs a lot. See Table 5.3 for which technology can achieve which. For integrity servers, immutability is ensured by storing the integrity information (mostly a signature) with the data. Most audited servers count on the concept of merkle trees, this is the case because the overhead is only marginal while the information can be easily verified. Most DLTs are based on blockchain and therefore changing the stored data is not possible. IPFS makes sure integrity is ensured by addressing it via the hash. The same holds true for DAT but instead of the hash the integrity is ensured by encryption and a signature. How technologies achieve integrity is shown in the according sections in chapter 4.

For each technology, it is required to know at least one request sink as well as whom to trust. It might also be possible for the PIA to be part of the system and therefore be its own request sink. If the request sinks and whom to trust is known it should be possible for PIAs to send requests and receive information. Some technologies require query data and respond only with the sensors fulfilling the query others might respond with the whole data and require the PIA to search the data. For most technologies, it is also possible to receive verification information for the data. However, some technologies require additional knowledge so a user can query data. For example, IPFS requires the user to know the hash of the data, to be able to create a request. In Nano, each user has their own blockchain, while it is possible to search blockchain after blockchain. If

the sensor provider is known it might be an option to only search in this specific blockchain increasing speed and decreasing the required computational power.

In a similar way some technologies require specific knowledge, not during the request step, but when the system is initialized the first time. This may be technology dependent and the meaning is not that specific. In a CA based concept, the CAs have to be known to the users to be able to verify their signatures. DNSSEC also requires the knowledge of the root server so a user is able to verify its signatures and therefore the data. While a request sink has to be known for all technologies, for SUNDR a specific server and an associated user have to be known. Both of these details have to be created and transmitted by the sensor provider. While this information has to be known by the users using the technologies, some things have to be available so the system itself can be set up. This holds true for all private technologies because some big players have to be known beforehand. For some technologies, those can be changed over time but this is not true for all. The same applies to the witnesses of Byteball which can be changed by the users of the system over an extended period of time. It is important to know which data is required to set up the sensor directory. Also, some information is required when the PIAs are set up, this information might even be hard coded and require software updates when data is changed.

Last but not least, for the sensor directory to stay honest some parts have to be distributed. This comes from the requirement of no powerful entity able to drop data, manipulate data, or decline information. To achieve this, some part of the system has to be distributed. First of all, it is possible for the trust to be distributed. This means trust is created by multiple cooperating entities. An example of this is a CA based system where the data is stored on one server while a CA creates signatures verifying the data. In theory, also SUNDR supports distributed trust but only if different users work on the data or communicate with each other. Because PIAs only query data it might be a possibility to split views for them if they do not communicate with each other. Another way of distribution is to distribute storage. This means the same storage is spread via a lot of nodes and the version supported by the majority of nodes is considered true. This means the majority of such a system has to stay honest or there is the potential for data to be manipulated. For those systems, a consensus algorithm is typically used to find this global truth stored by all nodes. Also, the malicious part of the network, has to work together to change the data, because also for them the majority is required to enforce their data. Last but not least a network of self-maintained servers distributes metadata via multiple nodes. This metadata stores where the data is stored. Because everyone has their servers handling data, the most important feature is to find the correct server. Integrity for those servers is created by using hashes and signatures.

5.1 Attack Vector Comparison

Already in section 2.7.1 each attack vector is assigned a category and a relevance factor. Those values were:

- Critical:
  - 8 - Insert many sensors with the same address (2.2.12)
  - 7 - Splitting views (2.2.17)
  - 7 - Data integrity (2.2.9)
  - 7 - Fingerprinting (2.2.3)

- 6 - Override sensor (2.2.12)
- 6 - Preregister expected sensors (2.2.12)

- High:
  - 4 - Malicious sensors are set up (2.2.6)
  - 4 - Sensor is compromised (2.2.7)
  - 3 - Destroy the reputation of sensors (2.2.13)
  - 3 - Reputation being outdated (2.2.13)
- Low:
  - 2 - DoS by adding sensors to the directory(2.2.1)
  - 2 - People tracking (Sensor Directory) (2.2.14)
  - 2 - People tracking (Sensors) (2.2.15)
  - 1 - Insert wrong sensor data (2.2.11)
  - 1 - Target discovery (2.2.4)
  - 1 - Data gathering (2.2.5)

Those relevance factors and categories are used to figure out which technologies can match the needs of the sensor directory best and are therefore best suited to build it. To be able to do this the tables shown in chapter 4 are used. If a technology can mitigate an attack vector the corresponding relevance factor is assigned to the technology. This is done for all attack vectors and technologies. Last but not least, for each technology the achieved factors are added, and a final score is calculated. When a technology is partially able to mitigate an attack, only half of the points are assigned. Additionally, all technologies are ordered according to their final score, but they are still grouped by their architecture. An overview of technologies which attack vectors they are able to mitigate and the final score for each technology regarding the attack vectors can be seen in Table 5.2.

For integrity servers the best score is reached by web of trust in a distributed version. This means a technology to distribute data is used which allows many additional security features but also requires many more nodes to participate so the system can be operated. If certain distribution methods are used, even further attack vectors could be mitigated, for example by using a DLT it would not be further possible for the system to split views and therefore reach an even higher score. Web of trust in this configuration is also able to mitigate all *critical* attack vectors and is therefore a valid candidate for the sensor directory.

The best scoring audited server is Merkle^2. Much more audited servers are able to mitigate all *critical* attack vectors. This means several technologies would be possible options, while Merkle^2 comes out on top. For the sensor directory to work however many users operating Merkle^2 servers, as well as auditors, are needed. Also, several other technologies are valid candidates to construct a suitable base for the sensor directory.

For multiple servers using a consensus algorithm Tor comes out on top. While, it could be used in multiple ways, using the servers directly as a sensor directory can mitigate much more attack vectors, while using it as a discovery tool creates much more flexibility. The final score of the standalone option is much higher and it also mitigates all *critical* attack vectors which means it is a suitable candidate for the sensor directory.

IPFS comes out on top for the architecture network of self-maintained servers. This is the case because the hash of the data is required to search for the data and therefore it is not possible to split views. In theory in DAT, the sensor provider could do so. That is why for the attack vectors, IPFS comes out on top for this architecture. DAT might still be a candidate for the sensor directory due to the fact only the sensor provider can perform such attacks.

Algorand comes out on top for DLTs, it does also mitigates all *critical* attack vectors and is therefore a possible candidate for the sensor directory. However, DLTs do require lots of nodes to operate. This is not the case for private DLTs where only a limited number of nodes is allowed. For private DLTs, Fabric comes up on top with its huge flexibility. Also, Fabric mitigates all attack vectors which are *critical* as well as all which are in the category *high*. However, for such a technology to work several known entities have to be chosen which are most likely not collude. Most of the technologies are still possible candidates for the sensor directory after this evaluation.

5.2 Requirements Comparison

Similar to attack vectors also for requirements relevance factors, as well as categories, were assigned. This was done in section 2.7.2.

- Critical:
  - 10 - Queriability of the sensor directory
  - 9 - Equality for everyone
  - 8 - Verifiability
  - 8 - Anonymity of all users
  - 8 - Only the owner can change data
  - 7 - Redundancy/distribution
- Low:
  - 6 - Storage requirements
- High:
  - 4 - Repeated validation
  - 4 - Input validation
  - 4 - Trust system
  - 4 - Know whom to trust
- Low:
  - 2 - Immutability
  - 1 - Prune sets
  - 1 - Prune queries
  - 0 - PIAs cache sensor data

Similar to the attack vector mitigations, also the requirements are compared. This is done by using the relevance factors and categories as well as the tables established in chapter 4. For each technology, it is checked if the requirement is fulfilled, and if this is the case the factor is added to the final score of the

Table 5.2: Mitigated attack vectors

Technology	Attack Vectors Mitigated																sum
	2.2.12 (8)	2.2.17 (7)	2.2.9 (7)	2.2.3 (7)	2.2.12 (6)	2.2.12 (6)	2.2.6 (4)	2.2.7 (4)	2.2.13 (3)	2.2.13 (3)	2.2.1 (2)	2.2.14 (2)	2.2.15 (2)	2.2.11 (1)	2.2.4 (1)	2.2.5 (1)	
Web of Trust distributed	1	1/2	1	1	1	1	1	1	1	0	1	1/2	1	0	0	0	56.5
Web of Trust standalone	1	1/2	1	1/2	1	1	1	1/2	1	0	1/2	1/2	1	0	0	0	41.0
SUNDR	1	0	1	0	1	1	1	0	1	1	0	0	1	1	1	1	41.0
CA	0	0	1	1	0	0	1	1	1	1	1	1	0	0	0	0	32.0
DNSSEC	0	0	1	1/2	0	0	0	0	0	0	1/2	1/2	0	0	0	0	12.5
Merkle² standalone	1	1	1	1	1	1	1	1	1	0	1	1/2	1	0	0	0	56.0
ARPKI standalone	1	1	1	1	1	1	1	1	0	0	1	1	0	1	0	0	52.0
AKI standalone	1	1	1	1	1	1	0	0	1	1	1	1	0	0	0	0	51.0
CHAINIAC	1	1	1	1/2	1	1	1	0	1	0	1/2	1/2	0	1	0	0	41.5
Software Distribution Transparency	1/2	1	1	1	1/2	1/2	0	1	0	0	1/2	1	0	0	0	0	31.0
CONIKS standalone	0	1	1	1	0	0	0	1	0	0	1	1	0	0	1/2	1/2	30.0
Contour	0	1	1	1	0	0	0	1	0	0	1	1	0	0	0	1	30.0
Transparency Logs standalone	0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	25.0
Transparency Logs verification	0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	25.0
Merkle² verification	0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	25.0
AKI verification	0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	25.0
ARPKI verification	0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	25.0
CONIKS verification	0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	25.0
AAD standalone	0	1	1	1	0	0	0	0	0	0	1/2	1	0	0	0	0	24.0
AAD verification	0	1	1	1	0	0	0	0	0	0	1/2	1	0	0	0	0	24.0
Tor consensus standalone	1	1	1	1	1	1	1	1	1	1	1/2	1	0	1	0	0	59.0
Distributed State Machine	1/2	1	0	1	1/2	1/2	1/2	0	0	0	1/2	1	0	1/2	1	0	30.5
Tor consensus discovery	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	9.0
IPFS	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	60.0
DAT	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	51.0
Algorand	1	1	1	1	1	1	1/2	1/2	1/2	0	0	1	0	1	0	0	49.5
Byteball	1/2	1	1	1	1/2	1/2	1/2	1	1/2	0	1	1	0	1	0	0	43.5
IOTA standalone	1/2	1	1	1	1/2	1/2	1/2	1/2	1	0	1	1	0	1/2	0	0	42.5
Nano	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	23.0
IOTA verification	0	0	1	1	0	0	0	0	1	0	1	1	0	0	0	0	21.0
Fabric	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1/2	1/2	61.0
Ethereum	1	1	1	1	1	1	1	1	1	1/2	1	1	0	1	1/2	1/2	59.5

technology. If a technology fulfills a requirement only partly, it is assigned half of the points. Also, here the architecture is ordered within each other using the final scores of the technologies to enable a better overview. The final score for each technology can be seen in Table 5.3.

For integrity servers the best scoring technology and the only one able to fulfill all *critical* requirements is Certificate Authorities. This means this is still a valid candidate for the sensor directory.

For audited servers all can fulfill the *critical* requirements. The highest scores are achieved by AKI and ARPKI when used to verify the content of an additional server. This is the case because this allows them to solve the storage requirements and gain additional points to their standalone counterpart.

The Tor consensus system is the best scoring system for multiple servers using a consensus protocol. Because it does not support continuous storage and replaces data instead updates are not considered as such. However, because the data is validated only the owner can create such data.

DAT as well as IPFS can fulfill *critical* requirements. While DAT does straight up fulfill the *critical* requirements, this is only partially true for IPFS. The sensor provider would be required to perform lots of actions to update data stored in the system. Also, the system would not support a discovery function and would rather work as a storage extension to PIAs. After all, the hash and the actual data could be substituted because the hash only matches those sensors. This means DAT matches the requirements for the sensor directory much better.

Also all DLT approaches can fulfill *critical* requirements. Algorand is the top scorer because it also allows for a trust system. Private DLTs all have a similar problem because they are all run by known parties and no one else can interact with the system directly, those parties could prefer their nodes and data over those of other users. This might not be a problem because many nodes are working together and it is only a small improvement in time if they do so. Fabric as well as Ethereum are quite capable and might be used.

5.3 Final Comparison

When having a look at Table 5.3 and Table 5.2 one sees there is no integrity server left mitigating all *critical* attack vectors and fulfill all *critical* requirements. The best option might be a web of trust systems. This system could be one using a DLT to further increase security and synchronization. In a web of trust, certain users might be more trusted than others and therefore gain more power. This may be a desired feature because honest users could create even more powerful trust assessments, however, this is also true for malicious users. If this inequality between users is not a reason to drop web of trust from the potential candidate list there is no reason why it cannot be used to implement the sensor directory. It is also possible to create the sensor directory by using any other technology but using a web of trust as a trust assessment system inside this technology. Using CAs on their own is not an option because it is not possible to meet all *critical* requirements when using them. However CAs do provide a way of validating sensors before they are inserted into the system which could be needed for the final implementation. This means while a system based on CAs does not meet the requirements, the final implementation could include CAs to increase trust into sensors.

Many audited servers mitigate the *critical* attack vectors and fulfill all *critical* requirements. This means those should be compared by their score. AKI and

Table 5.3: Fulfilled requirements

Requirement	Queriable (10)	Equality (9)	Verifiable (8)	Anonymity (8)	Owner can change data (8)	Redundancy/ Distributed (7)	Repeated Validation (4)	Input Validation (4)	Trust System (4)	Whom to Trust (4)	Storage Requirement (6)	Immutable (2)	Prune Queries (1)	Prune Sets (1)	**Sum**
CA	1	1	1	1	1	1	1/2	1/2	1	1	1	0	0	0	68.0
Web of Trust standalone	1	0	1	1	1/2	1	1	1	1	1	1	0	0	0	59.0
Web of Trust distributed	1	0	1	1	1/2	1	1	1	1	1	0	1	0	0	55.0
DNSSEC	1	1/2	1	1/2	1	1	0	0	0	1	1	0	0	1	49.0
SUNDR	1	0	1	0	1	0	0	0	0	1	1	1	1	1	40.0
AKI verification	1	1	1	1	1	1	1	1	1	1	1	0	0	0	72.0
ARPKI verification	1	1	1	1	1	1	1	1	1	1	1	0	0	0	72.0
Software Distribution Transparency	1	1	1	1	1	1	1	1	1	1	1/2	1	0	0	71.0
Contour	1	1	1	1	1	1	1	0	1	1	1	1	0	0	70.0
AKI standalone	1/2	1	1	1	1	1	1	1	1	1	0	1	0	0	63.0
CHAINIAC	1	1	1	1	1	1	0	1	1/2	1	1	1	0	0	68.0
Transparency Logs verification	1	1	1	1	1	1	1	0	0	1	1	0	0	0	64.0
ARPKI standalone	1/2	1	1	1	1	1	1	1	1	1	0	1	0	0	63.0
CONIKS verification	1	1	1	1	1	1	0	0	0	1	1	0	1/2	1/2	61.0
AAD verification	1	1	1	1	1	1	0	0	0	1	1	0	0	0	60.0
Merkle² verification	1	1	1	1	1	1	0	0	0	1	1	0	0	0	60.0
Transparency Logs standalone	1/2	1	1	1	1	1	1	0	0	1	0	1	0	0	55.0
CONIKS standalone	1/2	1	1	1	1	1	0	0	0	1	0	1	1/2	1/2	52.0
AAD standalone	1/2	1	1	1	1	1	0	0	0	1	0	1	0	0	51.0
Merkle² standalone	1/2	1	1	1	1	1	0	0	0	1	0	1	0	0	51.0
Tor consensus standalone	0	1	1	1	0	1	1	1	1/2	1	1	0	0	0	52.0
Distributed State Machine	1	1/2	0	1/2	1/2	1	0	1/2	0	1	0	0	0	0	35.5
Tor consensus discovery	1/2	1	0	1	0	1	0	0	0	0	1	0	0	0	35.0
DAT	1	1	1	1	1	1	0	0	1/2	1	1	0	1	1	64.0
IPFS	0	1	1	1	1/2	1	0	0	1/2	1	1	1	1	1	52.0
Algorand	1	1	1	1	1	1	1/2	1	1/2	1	0	1	0	0	64.0
IOTA standalone	1	1	1	1	1	1	1/2	0	1	1	0	1	0	0	62.0
Byteball	1	1	1	1	1	1	1/2	1/2	1/2	1	0	1	0	0	62.0
Nano	1	1	1	1	1	1	0	0	0	1	0	1	0	0	56.0
IOTA verification	1	1	1	1	1	0	1/2	0	0	1	1	0	0	0	55.0
Fabric	1	1/2	1	1	1	1	1	1	1	1	0	1	1/2	1/2	65.5
Ethereum	1	1/2	1	1	1	1	1	1	1	1	0	1	1/2	1/2	65.5

ARPKI gather the highest score and therefore seem to be the best match for the sensor directory. However, the workflow for both has to be altered to match the workflow of the sensor directory. This is because initially they are used to issue certificates and validate them but not to search the database. Software distribution transparency on the other hand is already a distribution software with a similar purpose and also fulfills the *critical* measurements. A key feature shared by all audited servers is the auditor, such an auditor is used to scan servers and make sure they stay honest. If the sensor directory is not created using an audited server, it might be an option to still use auditors as additional network participants scanning servers and making sure they cannot get malicious. By doing so they also make sure it is not possible to split views between different users. However, the system is required to use an easy verifiable data structure to allow for auditors.

Tor consensus in its standalone form gains the highest score for multiple servers using a consensus algorithm. However, those servers are fixed and hard to swap. This means the parties running them should be selected carefully. This is the case because replacing one server requires updates to all PIAs so they all know the updated list. Due to servers only holding the recent data, there is no storage problem when using such a system. It is also easily possible to verify sensors regularly and therefore find malicious actions. The duration of one period has to be selected carefully due to the fact new sensors are only available after this time.

If a network of self maintained servers is employed to create the sensor directory, everyone stores their own data. Due to the fact that IPFS cannot be used to discover anything, DAT is the best option to create the sensor directory. DAT allows sensor providers to create and update their data easily. Also, nodes are allowed to mirror the data of other nodes, and therefore sensors that are used more frequently are also discovered much faster. There might also be no need for additional trust due to the fact the sensor providers are all known. However, there might be an additional server so also public servers can be found. This server could be one of the other technologies and might require trust assessments. If IPFS were used it would be the same as transferring all sensors of a user to the PIA and storing it there. IPFS would only reduce the required storage space. However, this means for every system where storage is a problem it might be an option to use IPFS to decrease the required storage. A network of self maintained servers always requires trusted sensor providers to provide some sort of search key to PIAs which should find their sensors. In the same form, it would be possible to store the sensors of those sensor providers directly inside the PIAs and allow them to search the data themselves. While updating them would be harder than in a distributed system the network load would decrease. However, there would still be the need for a sensor directory to discover public sensors. This means storing sensors of trusted sensor providers on PIAs might be an addition to every possible solution.

Also most DLTs are able to mitigate *critical* attack vectors and fulfill *critical* requirements. Algorand is able to reach the highest score within this group. Algorand also has the advantage of only a small part of the system being required to find consensus, those nodes could also be required to perform additional tasks which decreases the load on the whole system compared to other DLTs where all nodes would have to perform such actions. IOTA would allow for some adjustments in the referencing algorithm to allow for a sort of web of trust where sensors refer to sensors they trust. The referencing algorithm could be further tweaked to implement witnesses similar to those of Byteball if such a system is thought to be more advantageous. Nano would also bring a whole new feature set to the table, by creating account chains it would be possible to find sensors created by specific sensor providers much more easily. However, this struc-

ture also makes it difficult to create a trust system in the system apart from the known sensor provider provided the key is known.

Last but not least private distributed ledgers are options to build the sensor directory. Fabric is a little bit more flexible and does allow for easy swapping of consensus algorithms as well as adding contract code. Ethereum might require a little more work before it could be employed. This is the case because Ethereum can be used public and private and therefore has additional features. One of those is for example the cryptocurrency byte which should be disabled for the private application. Ethereum also supports a different kind of blockchain which could come in handy after some time. This is due to the fact it also stores states which allows for some nodes to trim the blockchain. However, those states already could consume lots of storage which might lead to problems. Both technologies support contract codes which could be used to verify data in the system.

Chapter 6

Conclusion and Future Work

As shown in the previous sections finding the best technology is hard, or even impossible. This is the case due to the fact all technologies do things a little bit differently and come with different advantages and disadvantages. For this reason, it needs to be tried which approaches work best and which additional requirements should be met. Therefore, a look at the other parts of Digidow is required. Consequently, one technology for each architecture is chosen. Hence, the following technologies are the top candidates for the sensor directory:

- Web of trust
- AKI
- ARPKI
- Tor consensus
- DAT
- Algorand
- Fabric

Web of trust should be used in a distributed fashion to allow for several additional attack vectors to be mitigated. Maybe using a web of trust systems based on a DLT could also bring additional advantages. In web of trust systems, some users might build a very good reputation and gain additional power. If this is a problem, a different technology should be used. However, if those acquire enough trust they may act similarly to CAs and sign lots of sensors. It is also possible to use web of trust as a trust system for other technologies. While CAs are not an option on their own to build the sensor directory, it is possible to extend each technology using CAs. This means those would verify sensors and sign them if they are trustworthy and valid. The sensor directory would then only accept signed sensors, which would decrease the work required. PIAs would then be able to choose themselves if they trust the CA that signed the sensor.

For AKI as well as ARPKI additional servers holding the data might be required. Doing so allows a significant reduction of required storage. It also fits the original use case for those technologies much better and would therefore be much easier to implement. It would be an option to use IPFS as such a storage solution. This would allow each sensor provider to run their nodes storing their data reducing their dependency on others.

Before Tor consensus can be used, a pre-selection of big players needs to be conducted. The same holds if Fabric is used to implement the sensor directory, however, if Fabric is chosen a consensus algorithm has to be selected and contract code has to be written. This contract code could be used to verify sensors as well as query the system. Tor however could also be used to discover servers used to distribute sensors, while this approach on its own does not achieve a

very high score such an approach could be used whenever PIAs have to discover additional servers providing data.

DAT does only allow PIAs to find sensors of known sensor providers, however, there might be ways to make additional sensor providers known. PIAs might need keys beforehand to be able to search for sensors. Yet, if this is used, it might be easy to distribute sensors. This also allows for additional data to be distributed because everything encrypted with the key would be discovered. By using this technology, the required storage on all devices would be minimal.

While some of the technologies can be used out of the box this does not hold for all of them. For some, the actual system has to be planned and built, for others, it might be possible to use already established systems and just adapt them a little bit. It might also be an option to use only parts of the shown technologies and combine them into a completely new technology that perfectly matches the needs of the sensor directory. Also, the communication between all parts of Digidow and the sensor directory is not yet exactly specified which has to be done so it is possible to build the sensor directory. It was stated early on that storage requirements will be part of future work, while some technologies might not even have the problem of storage exhaustion some, like Algorand and Fabric, could run into such problems. This means some further investigations into these problems would be required if those technologies are used. If a web of trust based on a DLT is employed, such problems might occur and should be analyzed. To further investigate which technology works best to build the sensor directory, it is required to test several approaches. This might be done in a small test setup, which only supports limited nodes providing the sensor directory as well as limited sensors and PIAs.

Bibliography

[1] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J Alex Halderman, Jacob Hoffman-Andrews, James Kasten, and Eric Rescorla. 2019. Let's Encrypt: an automated certificate authority to encrypt the entire web. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, London, United Kingdom, pp. 2473–2487. DOI: 10.1145/3319535.3363192.

[2] Kiayias Aggelos, Alexander Russel, Bernardo Davir, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *Advances in Cryptology – CRYPTO 2017* (*LNCS*, volume 10401). Springer, pp. 357–388. DOI: 10.1007/978-3-319-63688-7_12.

[3] Amjad Aldweesh, Maher Alharby, Maryam Mehrnezhad, and Aad Van Moorsel. 2019. OpBench: A CPU performance benchmark for Ethereum smart contract operation code. In *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, Atlanta, USA, pp. 274–281. DOI: 10.1109/Blockchain.2019.00043.

[4] Carrillo Alexandra. 2018. An Introduction to the BlockDAG Paradigm. (2018). Retrieved 12/22/2023 from https://ancapalex.medium.com/an-introduction-to-the-blockdag-paradigm-50027f44facb.

[5] alyakubov. 2019. blockpgp. (2019). https://github.com/alyakubov/blockpgp.

[6] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, and Yacov Manevich. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, Porto, Portugal, pp. 1–15. DOI: 10.1145/3190508.3190538.

[7] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. 2018. PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain. In *Italian Conference on Cybersecurity*. Volume 2058. Rome, Italy, pp. 1–11. https://ceur-ws.org/Vol-2058/paper-06.pdf.

[8] Olha Anurina. 2023. How PWAs Work Offline: Technologies & Solutions. (2023). Retrieved 12/22/2023 from https://www.gomage.com/blog/pwa-offline/.

[9] Suranjith Ariyapperuma and Chris J Mitchell. 2007. Security vulnerabilities in DNS and DNSSEC. In *The Second International Conference on Availability, Reliability and Security (ARES'07)*. IEEE, Vienna, Austria, pp. 335–342. DOI: 10.1109/ARES.2007.139.

[10] N Arun, R Mathiyalagan, and Suchithra. 2021. Authentication and Identity Validation Blockchain Application. *Journal of Physics: Conference Series*, 1979, 1, (August 2021), 012017. DOI: 10.1088/1742-6596/1979/1/012017.

[11] Giuseppe Ateniese and Stefan Mangard. 2001. A New Approach to DNS Security (DNSSEC). In *Proceedings of the 8th ACM Conference on Computer and Communications Security*. ACM, Philadelphia, PA, USA, pp. 86–95. DOI: 10.1145/501983.501996.

[12] Raphael Auer, Cyril Monnet, and Hyun Song Shin. 2021. Permissioned distributed ledgers and the governance of money. *SSRN*, (January 2021), 71 pages. DOI: 10.2139/ssrn.3770075.

[13] Abigael Okikijesu Bada, Amalia Damianou, Constantinos Marios Angelopoulos, and Vasilios Katos. 2021. Towards a green blockchain: A review of consensus mechanisms and their energy consumption. In *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, Pafos, Cyprus, pp. 503–511. DOI: 10.1109/DCOSS52077.2021.00083.

[14] David Basin, Cas Cremers, Tiffany H Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. 2014. ARPKI: Attack resilient public-key infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Scottsdale, Arizona, USA, pp. 382–393. DOI: 10.1145/2660267.2660298.

[15] Mustafa Al-Bassam and Sarah Meiklejohn. 2018. Contour: A practical system for binary transparency. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology* (*LNCS*, volume 11025). Springer, Cham, pp. 94–110. DOI: 10.1007/978-3-030-00305-0_8.

[16] Michael Baumer, Marcel Waldvogel, Nathalie Weiler, and Bernhard Plattner. 1998. Distributed server for PGP keys synchronized by multicast. *Semesterarbeit, ETH Zurich*. https://www.researchgate.net/profile/Marcel-Waldvogel/publication/2378362_Distributed_Server_for_PGP_Keys_Synchronized_by_Multicast/links/0deec5285c01b004a4000000/Distributed-Server-for-PGP-Keys-Synchronized-by-Multicast.pdf.

[17] Marianna Belotti, Nikola Božić, Guy Pujolle, and Stefano Secci. 2019. A vademecum on blockchain technologies: When, which, and how. *IEEE Communications Surveys & Tutorials*, 21, 4, 3796–3838. DOI: 10.1109/COMST.2019.2928178.

[18] Federico M Benčić and Ivana P Žarko. 2018. Distributed ledger technology: Blockchain compared to directed acyclic graph. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1569–1570. DOI: 10.1109/ICDCS.2018.00171.

[19] Juan Benet. 2014. IPFS-content addressed, versioned, P2P file system (DRAFT 3). *arXiv preprint arXiv:1407.3561*, (July 2014). DOI: 10.48550/arXiv.1407.3561.

[20] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. 2014. Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake [Extended Abstract]y. In *SIGMETRICS Perform. Eval. Rev.* Number 3. Volume 42. ACM, New York, NY, USA, (December 2014), pp. 34–37. DOI: 10.1145/2695533.2695545.

[21] Hanno Böck. 2015. A look at the PGP ecosystem through the key server data. Cryptology ePrint Archive, Paper 2015/262. (2015). https://eprint.iacr.org/2015/262.

[22] Maria Borge, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, and Bryan Ford. 2017. Proof-of-Personhood: Redemocratizing Permissionless Cryptocurrencies. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, Paris, France, pp. 23–26. DOI: 10.1109/EuroSPW.2017.46.

[23] Santiago Bragagnolo, Henrique Rocha, Marcus Denker, and Stéphane Ducasse. 2018. Ethereum query language. In *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. ACM, pp. 1–8. DOI: 10.1145/3194113.3194114.

[24] Eric Brewer. 2000. Towards robust distributed systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*. ACM, Portland, Oregon, USA, (July 2000), p. 7. DOI: 10.1145/343477.343502.

[25] Daniel Burkhardt, Maximilian Werling, and Heiner Lasi. 2018. Distributed Ledger. In *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. IEEE, Stuttgart, Germany, pp. 1–9. DOI: 10.1109/ICE.2018.8436299.

[26] Germano Caronni. 2000. Walking the web of trust. In *Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000)*. IEEE, Gaithersburg, MD, USA, pp. 153–158. DOI: 10.1109/ENABL.2000.883720.

[27] Miguel Castro and Barbara Liskov. 1999. Practical byzantine fault tolerance. In *3rd Symposium on Operating Systems Design and Implementation (OSDI 99)*. USENIX Association, New Orleans, LA, (February 1999), pp. 173–186. https://www.usenix.org/conference/osdi-99/practical-byzantine-fault-tolerance.

[28] Jing Chen and Silvio Micali. 2019. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777, 155–183. DOI: 10.1016/j.tcs.2019.02.001.

[29] Anton Churyumov. 2016. Byteball: A decentralized system for storage and transfer of value. Retrieved 12/22/2023 from https://byteball.org/Byteball.pdf.

[30] Cloudflare. 2023. DNS DNSKEY und DS-Einträge. (2023). Retrieved 12/22/2023 from https://www.cloudflare.com/de-de/learning/dns/dns-records/dnskey-ds-records/.

[31] Deqode. 2019. S-Trilogy, Part Three: Direct Acyclic Graph, IOTA and Byteball. (2019). Retrieved 12/22/2023 from https://deqode.com/blog/s-trilogy-part-three-direct-acyclic-graph-iota-and-byteball.

[32] Ongaro Diego and Ousterhout John. 2023. The Raft Consensus Algorithm. (2023). Retrieved 12/22/2023 from https://raft.github.io/.

[33] DigiCert. 2023. Scaling CT Logs: temporal sharding. (2023). Retrieved 12/22/2023 from https://www.digicert.com/blog/scaling-certificate-transparency-logs-temporal-sharding.

[34] Raphaël Dunant. 2018. Implementation of a robust and scalable consensus protocol for blockchain. Technical report. École Polytechnique Fédérale de Lausanne, Lausanne ..., (January 2018). Retrieved 12/22/2023 from https://www.epfl.ch/labs/dedis/wp-content/uploads/2020/01/report-2017-2-raphael_dunant-certificate.pdf.

[35] Let‘s Encrypt. 2023. Challenge Types. (2023). Retrieved 12/22/2023 from https://letsencrypt.org/docs/challenge-types/.

[36] Ethereum. 2023. Ethereum Whitepaper. (2023). Retrieved 12/22/2023 from https://ethereum.org/en/whitepaper/.

[37] Sergio Ferragut. 2023. Real-time Analytics Database uses partitioning and pruning to achieve its legendary performance. (2023). Retrieved 12/22/2023 from https://imply.io/blog/real-time-analytics-database-uses-partitioning-and-pruning-to-achieve-its-legendary-performance.

[38] Pietro Ferraro, Christopher King, and Robert Shorten. 2018. Distributed Ledger Technology for Smart Cities, the Sharing Economy, and Social Compliance. *IEEE Access*, 6, 62728–62746. DOI: 10.1109/access.2018.2876766.

[39] Pietro Ferraro, Christopher King, and Robert Shorten. 2018. IOTA-based directed acyclic graphs without orphans. *arXiv preprint arXiv:1901.07302*, (December 2018). DOI: 10.48550/arXiv.1901.07302.

[40] Simson Garfinkel. 2003. *Pretty Good Privacy (PGP)*. John Wiley and Sons Ltd., pp. 1421–1422. ISBN: 0470864125.

[41] Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2018. Stake-bleeding attacks on proof-of-stake blockchains. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, Zug, Switzerland, pp. 85–92. DOI: 10.1109/CVCBT.2018.00015.

[42] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. ACM, Vienna, Austria, pp. 3–16. DOI: 10.1145/2976749.2978341.

[43] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles* (*SOSP '17*). ACM, Shanghai, China, pp. 51–68. DOI: 10.1145/3132747.3132757.

[44] Derric Gilling. 2020. Top 10 API Security Threats Every API Team Should Know. (2020). Retrieved 12/22/2023 from https://dzone.com/articles/top-10-api-security-threats-every-api-team-should.

[45] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. 2018. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, Lyon, France, pp. 309–318. DOI: 10.1145/3178876.3186097.

[46] Josef Gustafsson, Gustaf Overier, Martin Arlitt, and Niklas Carlsson. 2017. A first look at the CT landscape: Certificate Transparency logs in practice. In *Passive and Active Measurement* (*LNCS*, volume 10176). Mohamed Ali Kaafar, Steve Uhlig, and Johanna Amann, (Eds.) Springer, Cham, pp. 87–99. DOI: 10.1007/978-3-319-54328-4_7.

[47] Mike Hearn and Richard Gendal Brown. 2019. Corda: A distributed ledger. *Corda Technical White Paper*, 2019. Retrieved 12/22/2023 from https://www.r3.com/wp-content/uploads/2019/08/corda-technical-whitepaper-August-29-2019.pdf.

[48] Benjamin Hof and Georg Carle. 2017. Software distribution transparency and auditability. *arXiv preprint arXiv:1711.07278*, (November 2017). DOI: 10.48550/arXiv.1711.07278.

[49] Tobias Höller. 2019. Towards establishing the link between a person's real-world interactions and their decentralized, self-managed digital identity in the Digidow architecture. In *IDIMT-2019: Innovation and Transformation in a Digital World*. Trauner Verlag, Kutná Hora, Czech Republic, (September 2019), pp. 327–332. ISBN: 978-3-99062-590-3.

[50] Tobias Höller, Michael Roland, and René Mayrhofer. 2021. Analyzing inconsistencies in the Tor consensus. In *The 23rd International Conference on Information Integration and Web Intelligence*. ACM, pp. 485–494. DOI: 10.1145/3487664.3487793.

[51] Yuncong Hu, Kian Hooshmand, Harika Kalidhindi, Seung J Yang, and Raluca A Popa. 2021. Merkle 2: A Low-Latency Transparency Log System. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, San Francisco, CA, USA, pp. 285–303. DOI: 10.1109/SP40001.2021.00088.

[52] IPFS. 2023. How IPFS works. (2023). Retrieved 12/22/2023 from https://docs.ipfs.io/concepts/how-ipfs-works.

[53] Jan. 2014. Answer to "What is a package index file". (2014). Retrieved 12/22/2023 from https://askubuntu.com/a/550397.

[54] Rob Jansen, Kevin S Bauer, Nicholas Hopper, and Roger Dingledine. 2012. Methodically Modeling the Tor Network. In *Proceedings of the 5th USENIX Conference on Cyber Security Experimentation and Test*. USENIX Association, Bellevue, WA, pp. 1–8. Retrieved 12/22/2023 from https://www.usenix.org/conference/cset12/workshop-program/presentation/Jansen.

[55] Sara Jelen. 2023. What Are Certificate Transparency Logs? (2023). Retrieved 12/22/2023 from https://securitytrails.com/blog/what-are-certificate-transparency-logs.

[56] Krish K. 2023. Query pruning. (2023). Retrieved 12/22/2023 from https://answers.sap.com/questions/3047281/query-pruning.html.

[57] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil Gligor. 2013. Accountable key infrastructure (AKI) a proposal for a public-key validation infrastructure. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, Rio de Janeiro, Brazil, pp. 679–690. DOI: 10.1145/2488388.2488448.

[58] Sunny King and Scott Nadal. 2012. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake, (August 2012). Retrieved 12/22/2023 from https://bitcoin.peryaudo.org/vendor/peercoin-paper.pdf.

[59] Ramakrishna Kotla and Michael Dahlin. 2004. High throughput Byzantine fault tolerance. In *International Conference on Dependable Systems and Networks, 2004*. IEEE, Florence, Italy, pp. 575–584. DOI: 10.1109/DSN.2004.1311928.

[60] Hyun Kwon, Yongchul Kim, Hyunsoo Yoon, and Daeseon Choi. 2017. Optimal Cluster Expansion-Based Intrusion Tolerant System to Prevent Denial of Service Attacks. *Applied Sciences*, 7, 11. DOI: 10.3390/app7111186.

[61] Chris Lamb and Stefano Zacchiroli. 2021. Reproducible builds: Increasing the integrity of software supply chains. *IEEE Software*, 39, 2, 62–70, 2. DOI: 10.1109/MS.2021.3073045.

[62] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2020. Browser fingerprinting: A survey. *ACM Transactions on the Web (TWEB)*, 14, Article 8, 1–33, 2. DOI: 10.1145/3386040.

[63] Felix Lau, Stuart H Rubin, Michael H Smith, and Ljiljana Trajkovic. 2000. Distributed denial of service attacks. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*. Volume 3. IEEE, Nashville, TN, USA, (March 2000), pp. 2275–2280. DOI: 10.1109/ICSMC.2000.886455.

[64] Dimitrios Lekkas and Dimitris Gritzalis. 2007. Long-term verifiability of the electronic healthcare records' authenticity. *International Journal of Medical Informatics*, 76, 5, 442–448. DOI: 10.1016/j.ijmedinf.2006.09.010.

[65] Colin LeMahieu. 2018. Nano: A feeless distributed cryptocurrency network. 16, 17. Retrieved 12/22/2023 from https://content.nano.org/whitepaper/Nano_Whitepaper_en.pdf.

[66] Jinyuan Li, Maxwell N Krohn, David Mazieres, and Dennis E Shasha. 2004. Secure Untrusted Data Repository (SUNDR). In *6th Symposium on Operating Systems Design & Implementation (OSDI 04)*. USENIX Association, San Francisco, CA, pp. 121–136. https://www.usenix.org/conference/osdi-04/secure-untrusted-data-repository-sundr.

[67] Lindner Mark. 2023. cbase – A C Foundation Library. (2023). Retrieved 12/22/2023 from https://www.hyperrealm.com/oss_cbase.shtml.

[68] Ueli Maurer. 1996. Modelling a public-key infrastructure. In *Computer Security — ESORICS 96* (*LNCS*, volume 1146). Springer, Berlin, Heidelberg, pp. 325–350. DOI: 10.1007/3-540-61770-1_45.

[69] MDN Contributors. 2023. Making PWAs work offline with Service workers. (2023). Retrieved 12/22/2023 from https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Tutorials/js13kGames/Offline_Service_workers.

[70] Apeksha Mehta. 2022. The top API security risks and how to mitigate them. (2022). Retrieved 12/22/2023 from https://appinventiv.com/blog/how-to-mitigate-api-security-risks/.

[71] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. 2015. CONIKS: Bringing Key Transparency to End Users. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., (August 2015), pp. 383–398. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/melara.

[72] Paul Mockapetris and Kevin J Dunlap. 1988. Development of the domain name system. In *Symposium Proceedings on Communications Architectures and Protocols*. ACM, Stanford, California, USA, pp. 123–133. DOI: 10.1145/52324.52338.

[73] Roger M Needham. 1994. Denial of service: an example. *Communications of the ACM*, 37, 11, (November 1994), 42–46. DOI: 10.1145/188280.188294.

[74] Sullivan Nick. 2018. Introducing Certificate Transparency and Nimbus. (2018). Retrieved 12/22/2023 from https://blog.cloudflare.com/introducing-certificate-transparency-and-nimbus/.

[75] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. 2017. CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, (August 2017), pp. 1271–1287. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/nikitin.

[76] Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. 2017. Blockchain. *Business & Information Systems Engineering*, 59, 183–187, 3. DOI: 10.1007/s12599-017-0467-3.

[77] Institute of Networks and Security at Johannes Kepler University Linz. 2023. Private Digital Authentication in the Physical World. (2023). Retrieved 12/22/2023 from https://www.digidow.eu/.

[78] Maxwell Ogden, Karissa McKelvey, and Mathias Buus Madsen. 2017. Dat-distributed dataset synchronization and versioning. *Open Science Framework*, (January 2017). DOI: 10.31219/osf.io/nsv2c.

[79] Diego Ongaro and John Ousterhout. 2013. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*. USENIX Association, Philadelphia, PA, pp. 305–320. Retrieved 12/22/2023 from https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

[80] The Tor Project. 2023. Relay Search. (2023). Retrieved 12/22/2023 from https://metrics.torproject.org/rs.html#search/flag:authority.

[81] Benedikt Putz and Günther Pernul. 2019. Trust factors and insider threats in permissioned distributed ledgers. In *Transactions on Large-Scale Data- and Knowledge-Centered Systems XLII*. LNCS. Volume 11860. Springer, Berlin, Heidelberg, pp. 25–50. DOI: 10.1007/978-3-662-60531-8_2.

[82] Radix Publishing Ltd. 2018. Primer on Merkle Trees. (2018). Retrieved 12/22/2023 from https://www.radixdlt.com/post/primer-on-merkle-trees.

[83] Roya. 2014. Answer to "How many directory servers are there in the Tor networks?" (2014). Retrieved 12/22/2023 from https://tor.stackexchange.com/a/4853.

[84] Cornelius Schätz. 2018. How IOTA solves Blockchains scalability problem. (2018). Retrieved 12/22/2023 from https://hackernoon.com/how-iota-solves-blockchains-scalability-problem-12e5cae05531.

[85] Patrick Schöppl. 2019. *Personal Agent Prototype in Rust*. Master's thesis. Johannes Kepler University Linz, Institute of Networks and Security, Linz, Austria, (November 2019), 88 pages.

[86] SmallStep Labs. 2023. ACME Basics. (2023). Retrieved 12/22/2023 from https://smallstep.com/docs/step-ca/acme-basics/.

[87] Jordi Subira-Nieto. 2021. Security of Proof-of-Personhood: Idena. Retrieved 12/22/2023 from https://www.epfl.ch/labs/dedis/wp-content/uploads/2021/07/report-2021-1-jordi-idena_report.pdf.

[88] Michael Szydlo. 2004. Merkle Tree Traversal in Log Space and Time. In *Advances in Cryptology - EUROCRYPT 2004* (*LNCS*, volume 3027). Springer, Berlin, Heidelberg, pp. 541–554. DOI: 10.1007/978-3-540-24676-3_32.

[89] tdjsnelling. 2019. dat-keyserver. (2019). Retrieved 12/22/2023 from https://github.com/tdjsnelling/dat-keyserver.

[90] The DNS Institute. 2023. Proof of Non-Existence (NSEC and NSEC3). (2023). Retrieved 12/22/2023 from https://dnsinstitute.com/documentation/dnssec-guide/ch06s02.html.

[91] Alin Tomescu, Vivek Bhupatiraju, Dimitrios Papadopoulos, Charalampos Papamanthou, Nikos Triandopoulos, and Srinivas Devadas. 2019. Transparency logs via append-only authenticated dictionaries. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, London, United Kingdom, pp. 1299–1316. DOI: 10.1145/3319535.3345652.

[92] Jon Truby. 2018. Decarbonizing Bitcoin: Law and policy choices for reducing the energy consumption of Blockchain technologies and digital currencies. *Energy research & social science*, 44, 399–410. DOI: 10.1016/j.erss.2018.06.009.

[93] Martin Valenta and Philipp Sandner. 2017. Comparison of ethereum, hyperledger fabric and corda. FSBC Working Paper. Frankfurt School, Blockchain Center, (July 2017), pp. 1–8. Retrieved 12/22/2023 from https://www.smallake.kr/wp-content/uploads/2017/07/2017_Comparison-of-Ethereum-Hyperledger-Corda.pdf.

[94] Gene Vayngrib. 2020. Hypercore universe FAQ. (2020). Retrieved 12/22/2023 from https://github.com/tradle/why-hypercore/blob/master/FAQ.md#how-is-hypercore-different-from-ipfs.

[95] Martín A G Vigil, Cristian T Moecke, Ricardo F Custódio, and Melanie Volkamer. 2013. The Notary Based PKI. In *Public Key Infrastructures, Services and Applications* (*LNCS*, volume 7868). Springer, Berlin, Heidelberg, pp. 85–97. DOI: 10.1007/978-3-642-40012-4_6.

[96] Marko Vukolic. 2015. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In *Open Problems in Network Security* (*LNCS*, volume 9591). Springer, pp. 112–125. DOI: 10.1007/978-3-319-39028-4_9.

[97] Tao Wang and Ian Goldberg. 2013. Improved Website Fingerprinting on Tor. In *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*. ACM, Berlin, Germany, pp. 201–212. DOI: 10.1145/2517840.2517851.

[98] Tao Wang and Ian Goldberg. 2016. On Realistically Attacking Tor with Website Fingerprinting. *Proceedings on Privacy Enhancing Technologies*, 2016, 4, (February 2016), 21–36. DOI: 10.1515/popets-2016-0027.

[99] WIN-ACME. 2023. TLS-ALPN validation. (2023). Retrieved 12/22/2023 from https://www.win-acme.com/reference/plugins/validation/tls-alpn/.

[100] Erik Wittern, Alan Cha, James C Davis, Guillaume Baudart, and Louis Mandel. 2019. An Empirical Study of GraphQL Schemas. In *Service-Oriented Computing* (*LNCS*, volume 11895). Springer, Cham, pp. 3–19. DOI: 10.1007/978-3-030-33702-5_1.

[101] Anthony D Wood and John A Stankovic. 2002. Denial of service in sensor networks. *Computer*, 35, 10, 54–62. DOI: 10.1109/MC.2002.1039518.

[102] Alexander Yakubov, Wazen Shbair, and Radu State. 2018. BlockPGP: A blockchain-based framework for PGP key servers. In *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*. IEEE, Takayama, Japan, pp. 316–322. DOI: 10.1109/CANDARW.2018.00065.

[103] Yury Zhauniarovich, Issa Khalil, Ting Yu, and Marc Dacier. 2018. A Survey on Malicious Domains Detection through DNS Data Analysis. *ACM Computing Surveys (CSUR)*, 51, 4, (July 2018), 1–36. DOI: 10.1145/3191329.

[104] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. 2018. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14, 4, (October 2018), 352–375. DOI: 10.1504/IJWGS.2018.095647.