frontiers

Blockchain based Resource Governance for Decentralized Web Environments

Davide Basile [1], **Claudio Di Ciccio** [1], **Valerio Goretti** [1] **and Sabrina Kirrane** [2]

[1] *Sapienza University of Rome, Italy*
[2] *Vienna University of Economics and Business, Austria*

Correspondence*:
Valerio Goretti
valerio.goretti@uniroma1.it

ABSTRACT

Decentralization initiatives such as Solid, Digi.me, and ActivityPub aim to give data owners more control over their data and to level the playing field by enabling small companies and individuals to gain access to data, thus stimulating innovation. However, these initiatives typically employ access control mechanisms that cannot verify compliance with usage conditions after access has been granted to others. In this paper, we extend the state of the art by proposing a resource governance conceptual framework, entitled ReGov, that facilitates usage control in decentralized web environments. We subsequently demonstrate how our framework can be instantiated by combining blockchain and trusted execution environments. Through blockchain technologies, we record policies expressing the usage conditions associated with resources and monitor their compliance. Our instantiation employs trusted execution environments to enforce said policies, inside data consumers' devices. We evaluate the framework instantiation through a detailed analysis of requirements derived from a data market motivating scenario, as well as an assessment of the security, privacy, and affordability aspects of our proposal.

Keywords: Decentralization; Usage Control; Governance; Blockchain; Trusted Execution Environment

1 INTRODUCTION

Since its development, the internet has steadily evolved into a ubiquitous ecosystem that is seen by many as a public utility (Quail and Larabie, 2010). The development of centralized web-based platforms on top of the internet has undoubtedly brought benefits from both an economic and a social perspective. However, the web as we know it today, is dominated by a small number of stakeholders that have a disproportionate influence on the content that the public can produce and consume. The scale of the phenomenon has brought about the need for legal initiatives aimed at safeguarding content producer rights (Quintais, 2020). In parallel, technical decentralization initiatives such as Solid[1], Digi.me[2], and ActivityPub[3] aim to give data owners more control over their data, while at the same time providing small companies as well as individuals with access to data, which is usually monopolized by centralized platform providers, thus stimulating innovation. To this end, the Solid community are developing tools, best practices, and web standards that facilitate ease of data integration and support the development of decentralized social applications based on Linked Data principles. In turn, Digi.me are developing tools and technologies

[1] https://solidproject.org/about. Accessed: Thursday 11th May, 2023.
[2] https://digi.me/what-is-digime/. Accessed: Thursday 11th May, 2023.
[3] https://activitypub.rocks/. Accessed: Thursday 11th May, 2023

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

that enable individuals to download their data from centralized platforms such that they can store it in an encrypted personal data store and leverage a variety of applications that can process this data locally on the data owners device. These client-side applications are developed by innovative app developers who use the Digi.me software development kit to communicate with the encrypted personal data stores directly. Following the same principles, ActivityPub is a decentralized social networking protocol, published by the W3C Social Web Working Group that offers a client-server application programming interface (API) for adding, modifying, and removing material as well as a federated server-server API for sending notifications and subscribing to content. Social networks implementing ActivityPub can be easily integrated with each other in order to form a larger ecosystem, commonly referred to as the Fediverse[4]. Some of the most popular Fediverse initiatives include Mastodon[5], PeerTube[6], and PixelFed[7].

In order to better cater for use case scenarios that involve data sharing across various distributed data stores underpinning decentralized applications, there is a need for tools and technologies that are not only capable of working with distributed data but are also able to manage data resources that come with a variety of usage terms and conditions specified by data producers. However, the vast majority of decentralized web initiatives, which aim to provide users with a greater degree of control over personal resources, manage data access via simple access control mechanisms (Ouaddah et al., 2016; Toninelli et al., 2006; Tran et al., 2005) that are not able to verify that usage conditions are adhered to after access has been granted (Akaichi and Kirrane, 2022b). For example, access control rules can determine if users can retrieve data or not. However, they cannot express conditions on the type of application that can process them, the geographical area in which they can be treated, when the access grant would expire, or the number of times they can be processed.

When it comes to the realization of usage control in decentralized web environments, Trusted Execution Environments (TEEs) and Distributed Ledger Technologies (DLTs) could serve as fundamental enablers. Trusted execution environments offer data and code integrity to enforce the conditions established by decentralized data providers, directly in consumers' devices. DLTs can store shared policies in a distributed ecosystem in which data usage is governed by smart contracts, while recording an immutable log of usage operations.

To this end, in this paper we propose a resource governance (ReGov) conceptual framework and an instantiation thereof. ReGov combines blockchain applications and trusted execution environments to facilitate usage control in decentralized web environments. The work is guided by a typical decentralized web scenario, according to which data are not stored in centralized servers but rather in decentralized data stores controlled by users. Throughout the paper, we refer to the component for managing the data stored locally on every user's device as a *data node* (or *node* for simplicity).

In terms of contributions, we extend the state of the art by: (i) proposing a generic resource governance conceptual framework; (ii) demonstrating how blockchain technologies and trusted execution environments can together be used to manage resource usage; and (iii) assessing the effectiveness of the proposed framework via concrete quantitative and qualitative evaluation metrics derived from our data market motivating use case scenario.

[4] https://fediverse.party/en/fediverse/. Accessed: Thursday 11th May, 2023.

[5] https://docs.joinmastodon.org. Accessed: Thursday 11th May, 2023.

[6] https://peertube.uno. Accessed: Thursday 11th May, 2023.

[7] https://pixelfed.uno/site/about. Accessed: Thursday 11th May, 2023.

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

The remainder of the paper is structured as follows: Section 2 presents the necessary background information regarding data access and usage control, trusted execution environments, decentralized applications, and blockchain oracles. In the same section we also provide an overview of related work. We introduce the motivating scenario used to guide our work in Section 3 and our ReGov conceptual framework in Section 4. Following on from this, we described our DLT and TEE-based instantiation in Section 5 and the results of our quantitative and qualitative in Section 6. Finally, we conclude and outline directions for future work in Section 7.

2 BACKGROUND AND RELATED WORK

This section sets the context for the work being presented, highlighting the significance and relevance of the study. It also gives credit to previous work in the field and identifies gaps in the current understanding that the study aims to fill.

2.1 Background

As we leverage blockchain technologies and trusted execution environments to manage resource usage control, in the following we provide the necessary background information from these fields.

2.1.1 Data Access and Usage Control

Access control is a technique used to determine who or what can access resources in a computing environment (Sandhu and Samarati, 1994). In system infrastructures, access control is dependent upon and coexists alongside other security services. Such technologies require the presence of a trusted reference entity that mediates any attempted access to confidential resources. In order to decide who has rights to specific resources, access control frameworks make use of authorization rules, typically stored inside the system (Koshutanski and Massacci, 2003). A set of rules constitutes a policy. A popular approach of implementing access policies is through Access Control Lists (ACLs) (Grünbacher, 2003). Each protected resource has an associated ACL file, which lists the rights each subject in the system is allowed to use to access objects.

With the evolution of the web and decentralized data ecosystems, there is the need to move beyond managing access to resources via authorizations (Akaichi and Kirrane, 2022b). Authorization predicates define limitations that consider the user and resource credentials and attributes. Usage control is an extension of access control whereby policies take into account obligations and conditions in addition to authorizations (Lazouski et al., 2010). Obligations are constraints that must be fulfilled by users before, during, or after resource usage. Conditions are environmental rules that need to be satisfied before or during usage.

One of the most highly cited usage control models is UCON$_{ABC}$ (Park and Sandhu, 2004). The model represents policy rules by defining specific rights (e.g., operations to be executed) related to sets of subjects (e.g., users who want to perform an operation), objects (e.g., the resource to operate), authorizations, obligations, and conditions. *Attributes* are properties associated with subjects or objects. UCON$_{ABC}$ improves conventional access control mainly through the following two concepts: (i) *attribute mutability*, namely the change of attributes as a consequence of usage actions, and (ii) *decision continuity*, i.e., the enforcing of policies not only as a check at access request time, but also during the subsequent resource usage. Systems implementing usage control through the UCON$_{ABC}$ model require dedicated infrastructure to guarantee policy enforcement and monitoring in order to detect misconduct and execute compensation actions (e.g., penalties and/or right revocations).

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

The literature offers several alternative approaches that could potentially be used to represent usage control policies. For instance, Hilty et al. (2007) propose a language named Obligation Specification Language (OSL) intended for distributed environments. Bonatti et al. (2020) introduce the SPECIAL usage control policy language, which considers a policy as the intersection of basic entities governing data, processing, purposes, location, and storage of personal data. A comprehensive overview of existing usage control frameworks and their respective languages is provided by Akaichi and Kirrane (2022b) and Esteves and Rodríguez-Doncel (2022).

The overarching goal of our work is to enable usage control in a decentralized environment. We provide a conceptual framework that serves as a blueprint for policy governance in a decentralized setting.

2.1.2 Trusted Execution Environments

A Trusted Execution Environment (TEE) is a tamper-proof processing environment that runs on a separation kernel (McGillion et al., 2015). Through the combination of both software and hardware features, it isolates the execution of code from the operating environment. The separation kernel technique ensures separate execution between two environments. TEEs were first introduced by Rushby (1981) and allow multiple systems requiring different levels of security to coexist on one platform. Thanks to kernel separation, the system is split into several partitions, guaranteeing strong isolation between them. TEEs guarantee the authenticity of the code it executes, the integrity of the runtime states, and the confidentiality of the code and data stored in persistent memory. The content generated by the TEE is not static, and data are updated and stored in a secure manner. Thus, TEEs are hardened against both software and hardware attacks, preventing the use of even backdoor security vulnerabilities (Sabt et al., 2015). There are many providers of TEE that differ in terms of the software system and, more specifically, the processor on which they are executed. In this work, we make use of the Intel Software Guard Extensions (Intel SGX)[8] TEE. Intel SGX is a set of CPU-level instructions that allow applications to create *enclaves*. An enclave is a protected area of the application that guarantees the confidentiality and integrity of the data and code within it. These guarantees are also effective against malware with administrative privileges (Zheng et al., 2021). The use of one or more enclaves within an application makes it possible to reduce the potential attack surfaces of an application. An enclave cannot be read or written to from outside. Only the enclave itself can change its secrets, independent of the Central Processing Unit (CPU) privileges used. Indeed, it is not possible to access the enclave by manipulating registers or the stack. Every call made to the enclave needs a new instruction that performs checks aimed at protecting the data that are only accessible through the enclave code. The data within the enclave, in addition to being difficult to access, is encrypted. Gaining access to the Dynamic Random Access Memory (DRAM) modules would result in encrypted data being obtained (Jauernig et al., 2020). The cryptographic key changes randomly each time the system is rebooted following a shutdown or hibernation (Costan and Devadas, 2016). An application using Intel SGX consists of a trusted and an untrusted component. We have seen that the trusted component is composed of one or more enclaves. The untrusted component is the remaining part of the application (Zhao et al., 2016). The trusted part of the application has no possibility of interacting with any other external components except the untrusted part. Nevertheless, the fewer interactions between the trusted and untrusted part, the greater the security guaranteed by the application.

Our work resorts to trusted execution environments to keep control of resources' utilization by enforcing the usage conditions set by data owners.

[8] https://www.intel.co.uk/content/www/uk/en/architecture-and-technology/software-guard-extensions.html. Accessed: Thursday 11th May, 2023.

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

2.1.3 Decentralized Applications and Blockchain Oracles

With second-generation blockchains, the technology evolved from being primarily an e-cash distributed management system to a distributed programming platform for decentralized applications (DApps) (Mohanty, 2018). Ethereum first enabled the deployment and execution of smart contracts (i.e., stateful software artifacts exposing variables and callable methods) in the blockchain environment through the Ethereum Virtual Machine (EVM) (Buterin et al., 2014). The inability of smart contracts to access data that is not stored on-chain restricts the functionality of many application scenarios, including multi-party processes. Oracles solve this issue (Xu et al., 2016).

Oracles act as a bridge for communication between the on-chain and off-chain worlds. This means that DApps should also be able to trust an oracle in the same way it trusts the blockchain. Reliability for oracles is key (Mammadzada et al., 2020; Al-Breiki et al., 2020a). Therefore, the designation and sharing of a well-defined protocol become fundamental for the proper functioning of the oracle's service, particularly when the oracles themselves are organized in the form of networks for the interaction with decentralized environments (Basile et al., 2021). As illustrated by Mühlberger et al. (2020), oracle patterns can be described according to two dimensions: the information direction (inbound or outbound) and the initiator of the information exchange (pull- or push-based). While outbound oracles send data from the blockchain to the outside, inbound oracles inject data into the blockchain from the outside. Pull-based oracles have the initiator as the recipient, oppositely to push-based oracles, where the initiator is the transmitter of the information. By combining the push-/pull-based and inbound/outbound categories, four oracle design patterns can be identified (Pasdar et al., 2022). A push-based inbound oracle (*push-in* oracle for simplicity) is employed by an off-chain component that sends data from the real world. The push-based outbound (*push-out*) oracle is used when an on-chain component starts the procedure and transmits data to off-chain components. The pull-based outbound (*pull-out*) oracle is operated by an off-chain component that wants to retrieve data from the blockchain. Finally, the pull-based inbound (*pull-in*) oracle enables on-chain components to retrieve information outside the blockchain.

We leverage the blockchain's tamper-proof infrastructure to record usage conditions associated with resources. We represent this information via smart contracts running in the blockchain and communicating with off-chain processes through oracles.

2.2 Related work

Several works strive to provide more control and transparency with respect to personal data processing by leveraging blockchain distributed application platforms (Xu et al., 2019). For instance, Ayoade et al. (2018) defines an access control mechanism for IoT devices that stores a hash of the data in a blockchain infrastructure and maintains the raw information in a secure storage platform using a TEE. In the proposed framework, a blockchain based ledger is used in order to develop an audit trail of data access that provides more transparency with respect to data processing. Xiao et al. (2020) propose a system, called PrivacyGuard, which gives data owners control over personal data access and usage in a data market scenario.

The literature offers numerous study cases in which usage control frameworks have been instantiated to increase the degree of privacy and confidentiality of shared data. Neisse et al. (2011) propose a usage control framework in which a Policy Enforcement Point (PEP) keeps track of business operations and intercepts action requests while taking into consideration Policy Decision Point event subscriptions (PDP). Bai et al. (2014) addresses usage control in a Web Of Thing environment by adapting the UCON model for Smart Home ecosystems. Zhaofeng et al. (2020) introduce a secure usage control scheme for Internet

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

of things (IoT) data that is built upon a blockchain-based trust management approach. While, Khan et al. (2020) conceptualizes a distributed usage control model, named DistU, for industrial blockchain frameworks with monitoring procedures that are able to revoke permissions automatically.

Additionally, there are several papers that propose frameworks or architectures that combine blockchain platforms and decentralized web initiatives such as Solid web. Ramachandran et al. (2020) demonstrate how together Solid data stores (namely, *pods*) and blockchains can be used for trustless verification with confidentiality. Patel et al. (2019) propose a fully decentralized protocol named DAuth that leverages asymmetric encryption in order to implement authentication. Cai et al. (2020) introduce a secure Solid authentication mechanism, integrating Rivest–Shamir–Adleman (RSA) signatures into permissioned blockchain systems. In turn, Becker et al. (2021) demonstrate how data stored in Solid pods can be monetized by leveraging a blockchain based payment system. Whereas, Havur et al. (2020) discuss how solid could potentially leverage existing consent, transparency and compliance checking approaches.

Several studies have shown that blockchain and TEEs can profitably coexist. The state of the art proposes numerous cases where the combination of the two technologies leads to advantages in terms of data ownership, availability, and trust. One of these is the work of Liang et al. (2017), that propose a patient-centric personal health data management system with accountability and decentralization. The architecture of the framework employs TEEs to generate a fingerprint for each data access that are immutably maintained by a blockchain infrastructure. Whereas, Lind et al. (2017) designed and implemented a protocol named Teechain that integrates off-chain TEEs for secure and scalable payment procedures, built on top of the Bitcoin blockchain platform.

3 MOTIVATING SCENARIO AND REQUIREMENTS

The motivating use case scenario and the corresponding requirements, discussed in this section, are used not only to guide our work but also to contextualize theoretical notions introduced in the paper.

3.1 Motivating Scenario

A new decentralized data market called DecentralTrading aims to facilitate data access across decentralized data stores. Alice and Bob sign up for the DecentralTrading market, pay the subscription fee, and set up their data nodes. Alice is a research biologist in the area of marine science and is conducting studies on deep ocean animals. Such species are difficult to identify due to the adverse conditions of their ecosystem and the lack of good-quality images. Bob is a professional diver with a passion for photography. He has collected several photos from his last immersion and the most scientifically relevant of them portrays a recently discovered whale species named ‘Mesoplodon eueu’ showed in Fig. 1.

Bob shares his photos with the DecentralTrading market by uploading them to his data node. Once the images are shared, they can be retrieved by the other participants in the market. Moreover, he wants to establish rules regarding the usage of his images. Table 1 illustrates the constraints he exerts on the data utilization, along with the **rule type** they represent (inspired by the work of Akaichi and Kirrane, 2022a). Bob makes his images available only for applications belonging to the scientific domain (this constraint belongs to the type of **domain rules**). Moreover, he sets geographical restrictions by making the images usable only by devices located in European countries (**geographical rule**). Finally, Bob wants his photos to be deleted after a specific number of application accesses (**access counter rule**) or after a specific time interval (**temporal rule**). Therefore, he sets a maximum number of 100 local accesses and an expiry date of 20 days after the retrieval date. Bob gets remuneration from the DecentralTrading market, according to

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

Table 1. Schematization of the usage policy associated with Bob's 'Mesoplodon.jpg' image. Every rule belongs to a rule type and consists of a subject, an action, an object, and a constraint.

Rule components / Rule type	Subject	Action	Object	Constraint
Domain rule	market members	access the resource	Mesoplodon.jpg	The resource can be processed only by research apps
Geographical rule	market members	access the resource	Mesoplodon.jpg	The resource can be loaded only in European countries
Temporal rule	market members	access the resource	Mesoplodon.jpg	The resource can be stored for up to 20 days
Access counter rule	market members	access the resource	Mesoplodon.jpg	The resource can be opened up to 100 times

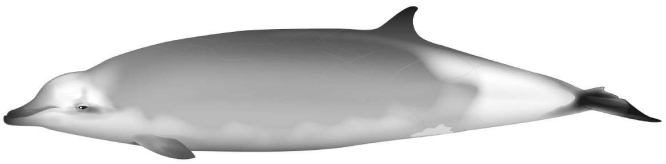

Figure 1. A photographic representation of a Mesoplodon eueu (Carroll et al., 2021). Image used under the Attribution 4.0 International (CC BY 4.0) license (https://creativecommons.org/licenses/by/4.0/). Cropped from original.

the number of requests for his resources. At any point in time, Bob can ask the DecentralTrading market to get evidence that the rules associated with his image are being adhered to and check if there were attempts to use his image outside the specified rules.

Bob's images of the Mesoplodon eueu species could be extremely useful for Alice's research, so she requests a specific picture of the gallery through her DecentralTrading node. Alice's node obtains a URL for Bob's node from the market and subsequently contacts Bob's node in order to retrieve a copy of the image, which is stored in a protected location of her device alongside the related usage rules. Data shared in DecentralTrading is used by Alice and Bob through a set of known applications approved by the market community. Alice opens the image through an app called 'ZooResearch', which is used for the analysis of zoological images. 'ZooResearch' belongs to the set of approved applications, and it disables some tasks for data duplication by the operating system (OS) such as screenshots to replicate the image once it is accessed. Since the domain of the application corresponds with the usage constraint set by Bob and her device is located in Ireland, the action is granted by Alice's node. Afterwards, Alice tries to share the image through a social network application named 'Socialgram', which also belongs to the set of supported applications. Then, Alice's node denies the action since it goes against the application domain constraint set by Bob. Alice opens the image through 'ZooResearch' 99 more times and, following the last attempt, the image is deleted from her node since the maximum number of local accesses of 100 has been reached. Therefore, Alice asks her DecentralTrading node to retrieve the image from Bob's node again. Since Alice starts working on a different research project, she stops using the Mesoplodon eueu's image. The image remains stored in the protected location of Alice's node until 20 days from the retrieval date have passed. Subsequently, Alice's node deletes the image from the protected location.

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

3.2 Requirements

The following concrete requirements are derived from our motivating scenario. The two top level requirements, which are inspired by the seminal work of Akaichi and Kirrane (2022b), are subdivided into more concrete sub-requirements.

(R1) Resource utilization and policy fulfillment must be managed by trusted entities. According to Akaichi and Kirrane (2022b), a usage control framework must provide an enforcement mechanism that ensures usage policies are adhered to both before and after data are accessed. Therefore, the data market must be able to able to handle the access control and additionally the nodes of a decentralized environment must be equipped with a dedicated component managing the utilization of resources owned by other nodes.

(R1.1) The trusted entity must be able to store resources obtained from other entities. Once resources are accessed, they must be kept in a trusted memory zone directly controlled by the trusted entity. This requirement drastically reduces the risks of data theft or misuse. Considering our running example, it allows Alice to not only store Bob's resources but also to protect them from unauthorized access.

(R1.2) The trusted entity must support the execution of programmable procedures that enforce constraints associated with resource usage. Specific procedures must be designed in order to cater for the various usage policy rules types. The trusted entity must execute these procedures in order to enforce policies and control resource utilization. This aspect enables the logic associated with usage control rules, such as those defined in Table 1, to be executed when Alice tries to use Bob's image.

(R1.3) Resources and procedures managed by the trusted entity must be protected against malicious manipulations. The trusted entity must guarantee the integrity of the resources it manages alongside the logic of the usage control procedures. Therefore, Alice should not be able to perform actions that directly manipulate Bob's image or corrupt the logic of the mechanisms that govern its use.

(R1.4) The trusted entity must be able to prove its trusted nature to other entities in a decentralized environment. Remote resource requests must be attributable to a trusted entity of the decentralized environment. Therefore, prior to Bob sending his image to Alice, it must be possible to verify that the data request has actually been generated by Alice's trusted node.

(R2) Policy compliance must be monitored via the entities of a governance ecosystem. According to Akaichi and Kirrane (2022b), usage control frameworks must incorporate a policy monitoring component. The monitoring, performed through one or more services, enables nodes to detect misconduct and unexpected or unpermitted usage. This is, e.g., the mechanism thanks to which Bob can verify that Alice has never tried to open the picture of the Mesoplodon eueu with Socialgram.

(R2.1) The governance ecosystem must provide transparency to all the nodes of the decentralized environment. In order to gain the trust of the various nodes that comprise a decentralized environment, a governance ecosystem must guarantee transparency with respect to its data and procedures. This feature enables Bob to verify at any time that the usage policy associated with his image is being adhered to.

(R2.2) Data and metadata maintained by the governance ecosystem must be tamper-resistant. Once policies and resource metadata are sent to the governance ecosystem, their integrity must be ensured. The inability to tamper with resources and their metadata is crucial for the effective functioning of the governance ecosystem. Therefore, when Bob publishes images and their respective usage policies in the market, his node should be the only entity capable of modifying this metadata.

(R2.3) The governance ecosystem and the entities that the form part of the ecosystem must be aligned with the decentralization principles. It is essential that the governance ecosystem itself respects the decentralization principles, as centralized solutions would establish a central authority in which data

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

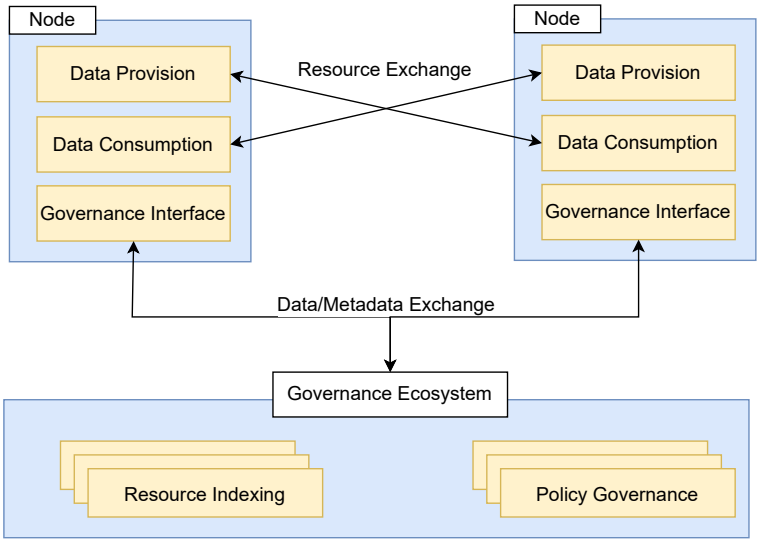

Figure 2. High-level overview of the proposed conceptual resource governance (ReGov) framework.

and decisional power are accumulated. Hence, the monitoring functionality provided by the previously mentioned market scenario should not rely on centralized platforms and data stores. Bob's policies for the usage of the Mesoplodon eueu's photo are not uploaded on, nor verified by, any third-party service running on a specific server.

(R2.4) The entities that form part of the governance ecosystem must be able to represent policies and verify their observance. In order to provide monitoring functionality, entities in the governance ecosystem should be capable of managing usage policies. These entities should enact procedures for retrieving policy observance information directly from nodes that consume market resources. This feature allows Bob to obtain evidence that Alice is using his image according to the rules stipulated in the usage policy and to detect any misbehavior.

4 CONCEPTUAL RESOURCE GOVERNANCE FRAMEWORK

To cater for our motivating scenario and to meet the derived requirements, we propose a conceptual framework, named ReGov, that enables the governance of usage policies in decentralized web environments. ReGov generalizes the principles of data ownership and control, which constitute the foundations of numerous decentralized web initiatives. The ReGov framework extends these aspects by not only controlling data access but also supporting the continuous monitoring of compliance with usage policies and enforcing the fulfillment of usage policy obligations. The degree of abstraction of the ReGov framework means that it could potentially be instantiated in numerous decentralized web contexts.

4.1 ReGov Framework Entities

According to the decentralization concept, the web is a peer-to-peer network with no central authority. In this scenario, data are no longer collected in application servers, but rather data are managed by nodes that are controlled by users (i.e., data owners determine who can access their data and in what context). Nodes communicate directly with other nodes in order to send and retrieve resources via the decentralized environment.

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

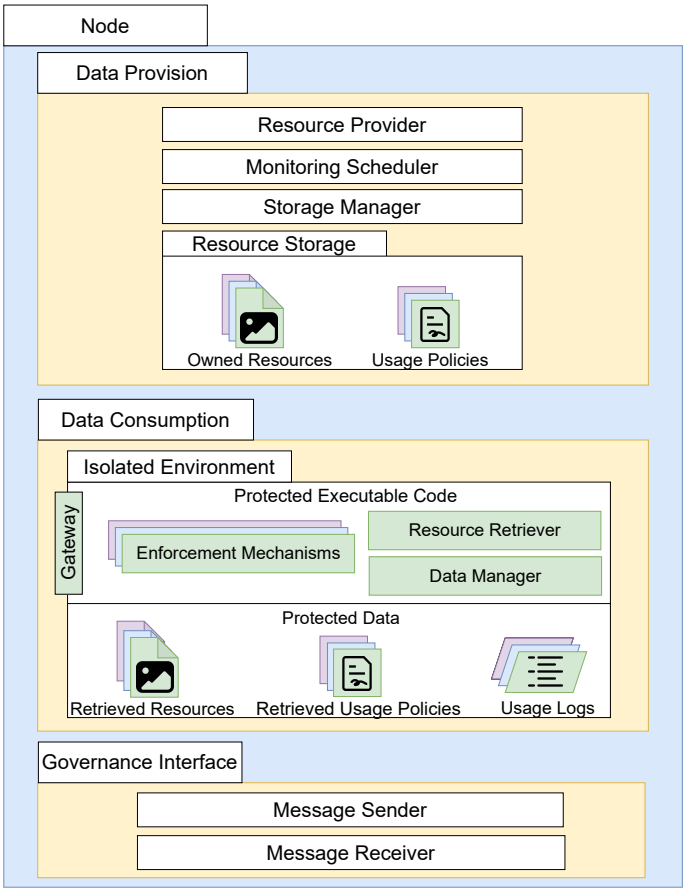

Figure 3. Content of the data provision, data consumption and governance interface components.

Figure 2 depicts a high-level overview diagram of the ReGov framework. Nodes are characterized by the `Data Provision`, `Data Consumption`, and `Governance Interface` components. Governance ecosystems are responsible for indexing web resources, facilitating node and resource discovery, and monitoring resource usage. Thus, in our architecture, a `Governance Ecosystem` is constituted by the `Resource Indexing` and `Policy Governance` components.

4.1.1 Components of a Node

A `Node` is a combination of hardware and software technologies, running on user devices. As shown in Fig. 3, each `Node` comprises the following components.

Data provision. The `Data Provision` component encapsulates the functionality that enable node owners to manage the sharing of their resources with other nodes in the decentralized environment. Users can interact with the `Storage Manager` to manually upload their data to the `Resource Storage` that is encapsulated within the `Data Provision` component. The upload operation also facilitates the definition of usage rules that are collected in usage policies associated with resources. Usage policies are represented in a machine-readable format (e.g., SPECIAL[9] and LUCON[10] policy languages) and stored in the `Data Provision` component alongside the resources. Additionally, when a new resource is uploaded, the `Storage Manager` forwards these rules and resource references to the `Governance Ecosystem`. In

[9] https://ai.wu.ac.at/policies/policylanguage/. Accessed: Thursday 11th May, 2023.

[10] https://industrial-data-space.github.io/trusted-connector-documentation/docs/usage_control/. Accessed: Thursday 11th May, 2023.

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

order to deliver the stored resources, the Data Provision component offers the logic for a Resource Provider that is capable of processing requests that allow other nodes to retrieve data. A data request must contain the necessary information to perform the authentication of the sender node. Therefore, the Resource Provider is able to authenticate resource requests to decide whether to grant or deny access to the requested resource based on the identity of the sender. Several web service protocols could potentially be used to implement the functionality offered by the Resource Provider (e.g., HTTP, FTP, Gopher). Once data are delivered, node owners can plan sessions to monitor the utilization of provisioned resources through the Monitoring Scheduler, which periodically forwards monitoring requests to the Governance Ecosystem.

Referring to our running example, Bob uses the functionality of the Storage Manager inside the Data Provision component to upload the images to his Node. During the upload, he specifies the location where the images must be stored and the rules composing the images' Usage Policy (i.e. the image must be deleted 20 days after the retrieval date, the image can only be used in European countries). Therefore, these pieces of information are delivered to the Governance Ecosystem. The HTTP web service implementing the Resource Provider of Bob's Node enables him to make his resource available to the other participants of the DecentralTrading market. The web service authenticates the requests for his images to determine whether the sender has the rights to access the resource. Finally, Bob can schedule monitoring sessions through the Monitoring Scheduler, in order to get evidence of the usage of his images by other nodes.

Data consumption. The Data Consumption component groups the functionalities that enable nodes to retrieve and use data in the network. Data Consumption is built upon both hardware and software techniques that ensure the protection of sensitive data through an Isolated Environment that guarantees the integrity and confidentiality of protected data and executable code. The Isolated Environment contains the logic of a Resource Retriever that creates authenticable requests for data residing in other nodes. The Resource Retriever supports multiple web protocols (e.g., HTTP, FTP, Gopher) according to the implementation of the Resource Provider inside the Data Provision component. Therefore, if the Resource Provider is implemented as an FTP web service, the Request Retriever must be able to generate authenticable FTP requests. Once resources are retrieved alongside the related usage policies, they are controlled by the Data Manager that stores them in the Isolated Environment. To get access to a protected resource, local applications running in the Node must interact with the Data Manager via the Gateway, which acts as a bridge to the processes running in the Isolated Environment. The Gateway is similarly employed when the Resource Retriever demands new resources from other nodes. In turn, Enforcement Mechanisms governing data utilization are necessary to apply the rules of the usage policies. While controlling resources, the Data Manager cooperates with these mechanisms enabling the rules contained in the usage policies to be enforced. Each operation involving the protected resources is recorded in dedicated usage logs whose administration is entrusted by the Data Manager too. Usage logs facilitate policy monitoring procedures that employ these registers to detect potential misconduct.

As shown in the motivating scenario, Alice uses the Data Consumption component to get Bob's images, which she keeps in her own Node. During the resource retrieval process, the Resource Retriever of Alice's Data Consumption component directly communicates with the Data Provision component of Bob's Node through the Gateway. After the retrieval, the image and the associated policy are maintained in the Isolated Environment and governed by the Data Manager. Considering the geographical rule, when Alice tries to open Bob's image with a local application, the app interacts with the Gateway, which in turn, creates a communication channel with the Data Manager. The latter generates the execution of

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

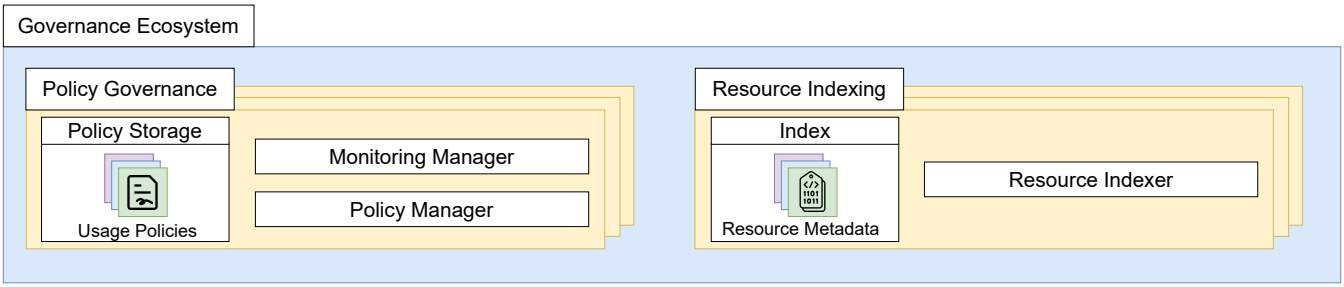

Figure 4. Content of policy governance and resource indexing components inside the governance ecosystem

the `Enforcement Mechanism` of the geographical constraint. This mechanism consults the image's usage policy, retrieves the current geographical position of the `Node`, and decides whether to grant the action.

Governance interface. Nodes facilitate communication with the `Governance Ecosystem` via the `Governance Interface`. As we will see in Section 4.2.2, messages flowing through the `Governance Interface` are crucial for resource usage monitoring. Indeed, the `Governance Ecosystem` can forward the interface messages such as requests for usage logs by remotely interacting with the `Message Receiver`. When a new message is received, the `Governance Interface` interacts with the other components of the `Node` in order to deliver the information. Similarly, the `Data Provision` and `Data Consumption Components` make use of the `Message Sender` to transmit data to the `Governance Ecosystem`. In order to provide continuous communication, the `Governance Interface` must constantly be active and listening for new messages.

4.1.2 Components of the Governance Ecosystem

We extend the typical decentralized model by including the `Governance Ecosystem`, illustrated in Fig. 4. The ecosystem hosts the `Resource Indexing` and `Policy Governance` components, whose multiple instances are able to immutably store data and metadata, execute procedures, and communicate with all the nodes of the decentralized environment.

Policy governance. `Policy Governance` components provide shared `Policy Storage` in which data owners publish applicable usage policies associated with resources. Policies are uploaded and modified through the `Policy Manager` of the component. In addition to their storage capabilities, `Policy Governance` components are able to execute procedures for policy monitoring. This function is supported by the `Monitoring Manager` of the component, containing the logic to verify the compliance of the policies stored inside the `Policy Storage`. Therefore, nodes forward monitoring requests to the `Monitoring Manager` which keeps track of resource usage and detects any illicit behavior.

Resource indexing. Policies are associated with resources through `Resource Indexing` components. They contain metadata about the resources shared in the decentralized environment (e.g., identifiers, web references, owner node). When data owners upload new resources in their node, it interacts with the `Resource Indexer` of these components, in order to serialize the information of the shared data.

Referring to our running example, when Bob uploads his image to his `Node` and specifies the corresponding usage rules in its policy, his `Node` shares the image metadata (e.g., the HTTP reference `https://BobNode.com/images/Mesoplodon.jpg`) and the usage policy with respectively the `Resource Indexing` and `Policy Governance` components running in the `Governance Ecosystem`. After Bob has delivered his 'Mesoplodon.jpg' image to Alice's `Node`, he can demand the verification of the image's

Pre-print copy of the manuscript published in **journal**
identified by DOI: `10.3389/fbloc.2023.1141909`

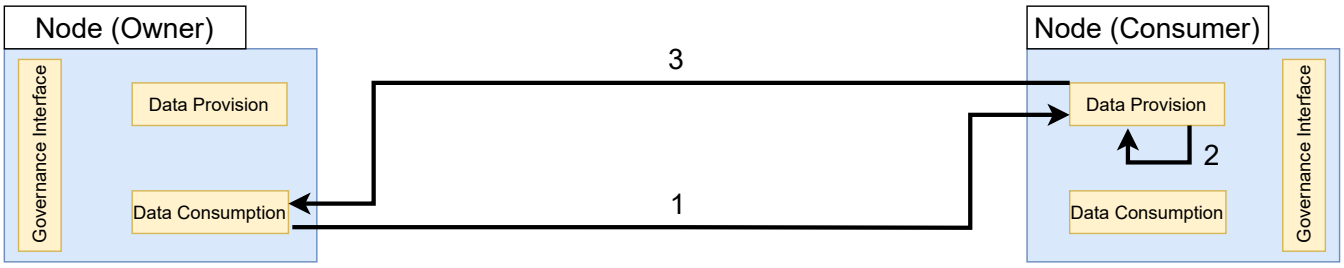

Figure 5. Visualization of the ReGov framework data retrieval process.

utilization to the `Policy Governance` component holding the image's policy. The `Policy Governance` component retrieves the usage log of the image from Alice's device, by interacting with her `Node`. Finally, Alice's usage can be verified based on the content of the usage log.

4.2 Predominant ReGov Framework Operations

Now that we have introduced the entities of our ReGov framework, we detail the predominant framework operations: data retrieval and monitoring. In the following, we simplify the processes by distinguishing owner nodes (i.e., nodes that are assuming the role of data providers) from data consumer nodes (i.e., nodes that are requesting access to and using resources), however, in practice, all nodes are dual purpose.

4.2.1 Data Retrieval

The data retrieval process allows consumer nodes to retrieve a resource from the decentralized environment. Figure 5 depicts a diagram representing the process. In order to obtain a specific resource, the data consumer `Node` generates a new request and sends it to the owner `Node`. We assume the consumer `Node` already has the information needed to contact the owner node (e.g., IP address or web reference). This information can be obtained by reading resource metadata maintained by `Resource Indexing` components running in the governance ecosystem. The process starts when the `Resource Retriever` inside the `Data Consumption` component of the consumer `Node` formats the request specifying the resource to be accessed and additional parameters intended for verification purposes. Subsequently, the request leaves the `Isolated Environment` through the `Gateway` and is received by the `Resource Provider` inside the `Data Provision` component of the owner node (**1**). The latter uses the parameters of the request to verify the identity of the sender `Node` (**2**). At this stage, the `Resource Provider` also verifies that the request has been generated in the `Isolated Environment` of a `Data Consumption` technology. Requests generated by alternative technologies are rejected. Once verified, the `Resource Provider` decides whether to grant access to the resource, according to the identity of the sender `Node`. If access is granted, the resource provider interacts with the `Storage Manager` inside the `Data Provision` component in order to construct the response, which includes both the requested resource and its usage policy. Finally, the `Resource Retriever` of the consumer `Node` obtains the resource, stores it in the `Isolated Environment` and registers it with the local `Data Manager` (**3**), as described in Section 4.1.1.

4.2.2 Monitoring

The policy monitoring process is used to continuously check if usage policies are being adhered to. In Fig. 6, we schematize the monitoring procedure. The owner node initiates the process via a scheduled job. Therefore the `Monitoring Scheduler` in the `Data Provision` component employs the `Message Sender` of the `Governance Interface` (**1**) to send a monitoring request, regarding a specific resource, to a `Policy Governance` component running in the `Governance Ecosystem` (**2**). Subsequently, the `Policy`

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

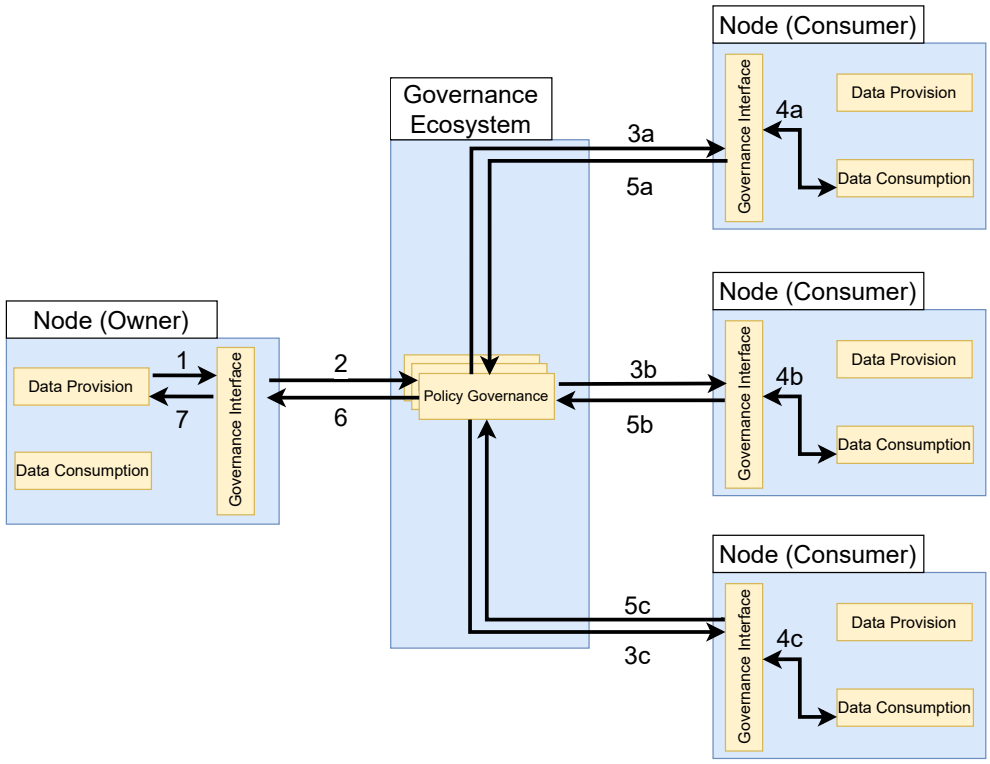

Figure 6. Visualization of the ReGov framework data monitoring routine.

`Governance` component forwards the request to provide evidence of utilization to each consumer `Node` that has a copy of the resource (**3a, 3b, 3c**). In the depicted monitoring routine, we assume the resource whose usage must be monitored is held by three consumer nodes. In each of these nodes, the monitoring request is received by the `Message Receiver` of the `Governance Interface` that forwards, in turn, the request to the `Data Manager` running in the `Isolated Environment` inside the `Data Consumption` component (**4a, 4b, 4c**). The latter retrieves the usage log from the protected data storage and employs the `Message Sender` of the `Governance Interface` to forward the information to the `Governance Ecosystem`, which in turn ensures that all the consumer node responses are collected (**5a, 5b, 5c**). Finally, the evidence are returned to the `Message Receiver` (**6**) of the initiator `Node`, which delivers the information to the `Monitoring Scheduler` (**7**).

5 BLOCKCHAIN AND TRUSTED EXECUTION ENVIRONMENT INSTANTIATION

In this section, we describe an instantiation of the ReGov framework. To this end, we propose a prototype implementation of the DecentralTrading data market illustrated in the motivating scenario. The implementation integrates a trusted application running in a trusted execution environment and blockchain technologies to address usage control needs. The code is openly available at the following address: https://github.com/ValerioGoretti/UsageControl-DecentralTrading.

In Fig. 7, we visualize the architecture of our ReGov framework instantiation. As shown in Section 4, the general framework assumes nodes of the decentralized environment are characterized by separate components dealing with `Data Provision` and `Data Consumption`. The `Data Provision` functionality is implemented in a software component we refer to as a `Personal Online Datastore`. We leverage security guarantees offered by the `Intel SGX Trusted Execution Environment` in order to implement

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

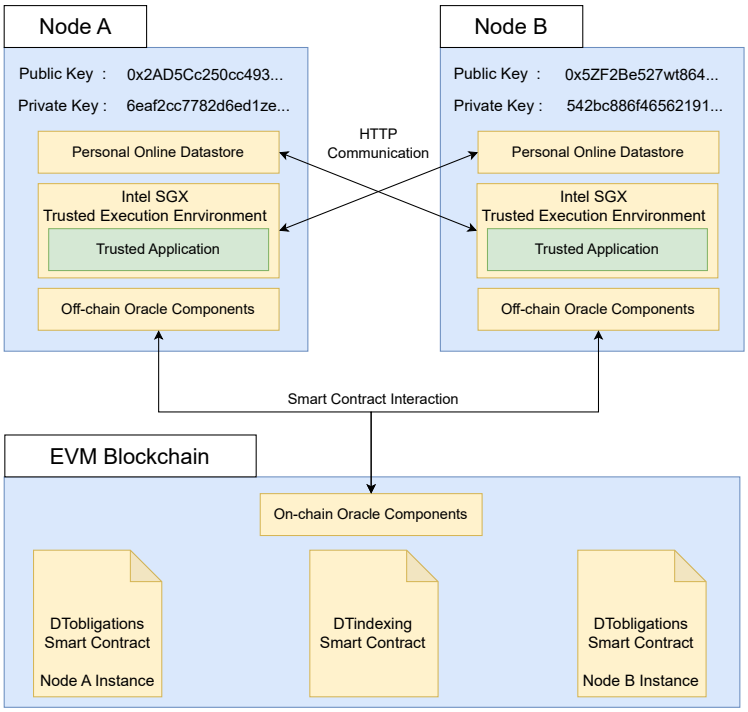

Figure 7. High-level architectural overview of our ReGov framework instantiation.

a `Trusted Application` containing the logic for `Data Consumption`. The `Governance Ecosystem` is realized by developing blockchain smart contracts that store information and execute distributed procedures. Our implementation involves an `EVM Blockchain`[11] (i.e., a blockchain based on the Ethereum Virtual Machine) which hosts the `DTindexing` and `DTobligations` smart contracts. They fulfill the functions of the `Resource Indexing` and `Policy Governance` components of the general framework, respectively. `DTindexing` is characterized by a unique instance managing the resource metadata of the decentralized environment. Instead, `DTobligations` is designed to be deployed multiple times. Therefore, each `Node` is associated with a specific instance of this smart contract that stores the rules for its resources. The tasks performed by the `Governance Interface` are executed by blockchain oracles that provide a communication channel between the blockchain and the nodes of the decentralized environment. Oracles consist of `On-Chain` components, running in the `EVM Blockchain`, and `Off-Chain` components, operating within each `Node`. We built the resource retrieval process between nodes using the HTTP communication standard. By interacting with smart contracts, nodes exchange metadata necessary for resource indexing and monitoring procedures.

Our implementation employs the asymmetric encryption methodology that underlies the `EVM Blockchain`, in order to provide an authentication mechanism for the environment's nodes. Each `Node` is uniquely related to a public and private key pair that is used to sign authenticable data requests and transactions that transmit information to the blockchain and execute smart contract functions. A private key is a 256-bit number generated through a secure random number generator. The corresponding public key is derived from the private key through the Elliptic Curve Digital Signature Algorithm (Johnson et al., 2001). The public key is connected to a unique account address on the `EVM Blockchain` derived as a 160-bit segment of the hash digest of the public key. In our setting, `Nodes` store their private key in an encrypted format to increase the degree of confidentiality of this information.

[11] Ethereum Virtual Machine (EVM): https://ethereum.org/en/developers/docs/evm/. Accessed: Thursday 11th May, 2023.

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

In the following, we describe the technical details of the individual aspects of our implementation. In particular, we focus on features inherent to resource governance (data retrieval, enforcement, and monitoring) and avoid the implementation details related to the data market logic (e.g., subscription payments and remuneration mechanisms).

5.1 Usage Policy Instantiation

The first step of the instantiation process involves the definition of rule types that are used to stipulate usage policies. While our approach allows for a wide range of rules, we establish a specific subset of rules to demonstrate the capabilities of our ReGov framework. In particular, we propose four types of rules inspired by the work of Akaichi and Kirrane (2022a). Each rule assumes that the target resource has already been retrieved and stored on the consumer device. In the following, we explain the various rule types that have already been introduced in the motivating scenario detailed in Section 3.1.

Temporal rules. Through a temporal rule, data owners establish the maximum time a resource can be maintained within a consumer device. The rule is parameterized through an integer value representing the duration in seconds. Once the term expires, the rule stipulates that the resource must be deleted.

Access counter rules. An access counter rule specifies a maximum number of local accesses that can be executed for a specific resource, after which, the resource must be deleted. The rule is parameterized with an integer value that defines the maximum number of accesses.

Domain rules. The domain rule represents the purpose for which a resource can be opened. It is characterized by an integer value that identifies groups of applications that share the same domain. Known applications that are part of the domain group can execute local access to the resource.

Geographical rules. A geographical constraint is a limitation on where a resource can be used. It is indicated by an integer code that specifies the territory in which the resource can be utilized.

5.2 Personal Online Data Stores for Data Provision

We develop the `Personal Online Datastore` prototype using the Python language. Python's support for the Web3.py library[12] enables the creation of communication protocols with the blockchain platform acting as the `Governance Ecosystem` of the decentralized environment. Our implementation also includes a graphical user interface developed with the Tkinter library[13]. As shown in Fig. 8, our `Personal Online Datastore` implementation is composed of three main parts: the `Application`, the `Web Service` and the `Resource Storage`. The `app` module contains the executable code implementing the graphical user interface.

5.2.1 Resource Storage

The resource storage contains the resources of the `Personal Online Datastore`. The storage location is characterized by two meta-files named `DTconfig.json` and `DTobligations.json`. They contain descriptive and confidential information about the `Personal Online Datastore` and its resources. `DTconfig.json` includes various attributes of a `Personal Online Datastore`, such as its unique identifier, its node's public and private keys, the web reference to access data, and a list of the initialized resources. `DTobligations.json` holds rules that apply to the resources of the storage. The user can establish a default policy inherited by all resources in the `Personal Online Datastore`, except those

[12] https://web3js.readthedocs.io/en/v1.8.1/. Accessed: Thursday 11th May, 2023.

[13] https://docs.python.org/3/library/tk.html. Accessed: Thursday 11th May, 2023.

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

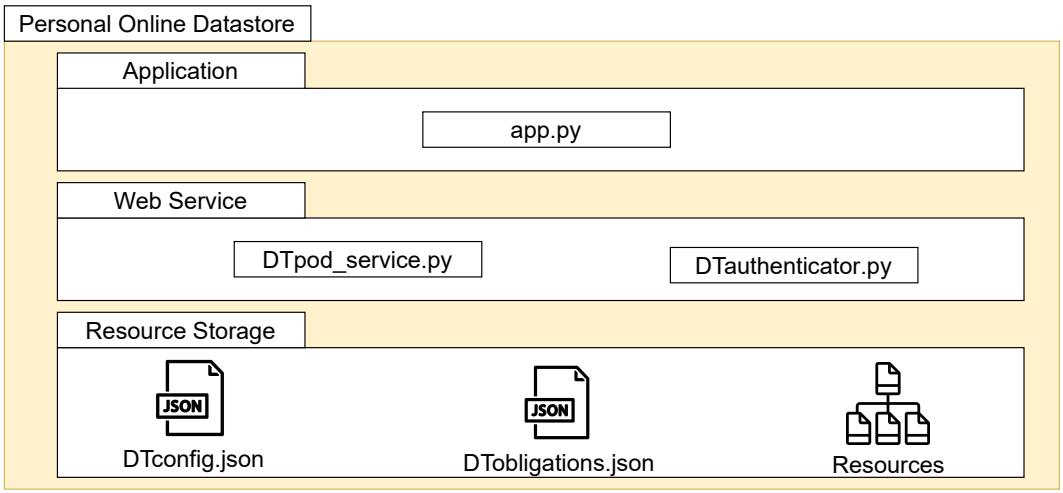

Figure 8. Schematization of the personal online datastore implementation.

with specific policies. Mentioning our running example, Bob interacts with the `Personal Online Datastore` application to upload the 'Mesoplodon.jpg' resource in the '/images' location inside the storage. During this process, Bob can establish the rules associated with the image. The initialization of the image generates the metadata to be held in the `DTconfig.json` and `DTobligations.json` metafiles.

5.2.2 Web Service

The implementation of the data provision process is built upon the HTTP web standard. Our `Personal Online Datastore` prototype implements a `Web Service` that listens for HTTP requests, verifies the authenticity of the sender `Node`, and delivers the requested data through HTTP responses. This approach enables the efficient and on-demand provision of initialized data. In Fig. 9, we summarize the main stages of the data provision process, taking place in our `Web Service` implementation. The `DTpod_service` Python class contains the core functionality for resource delivery. The class extends `BaseHTTPRequetsHandler` that enables the processing of GET and POST requests. Due to confidentiality reasons, the `Web Service` of the `Personal Online Datastore` only responds to `POST Requests` and ignores GET ones. The data provision process starts with the `Parameter Extraction`, which takes place when a new `POST Request` is received by the `Web Service`. The parameters inside the body of the `POST Request` are crucial for the authentication and remote attestation procedures. In order to correctly demand a resource, requests must specify a URL composed of the web domain name of the service followed by the relative path of the requested resource inside storage. In the case of the motivating scenario, to retrieve Bob's image, Alice's node must generate an authenticable `POST Request`, whose URL is 'https://BobNode/images/Mesoplodon.jpg'.

Through remote attestation, the `Web Service` can verify that the resource request has been legitimately generated by a `Trusted Application` running a `Intel SGX Trusted Execution Environment` of a `Node`. Therefore, we leverage the `Intel SGX Remote Attestation Verification` to establish a trusted communication channel between the consumer and the owner nodes. Once the attestation procedure ends successfully, the `Web Service` can be assured that the content of its response is managed by a `Data Consumption` technology inside the decentralized environment.

`Sender Authentication` takes place after the successful outcome of the remote attestation verification. The logic of our authentication mechanism is implemented through the `DTauthenticator` class, whose

Pre-print copy of the manuscript published in **journal**
identified by DOI: `10.3389/fbloc.2023.1141909`

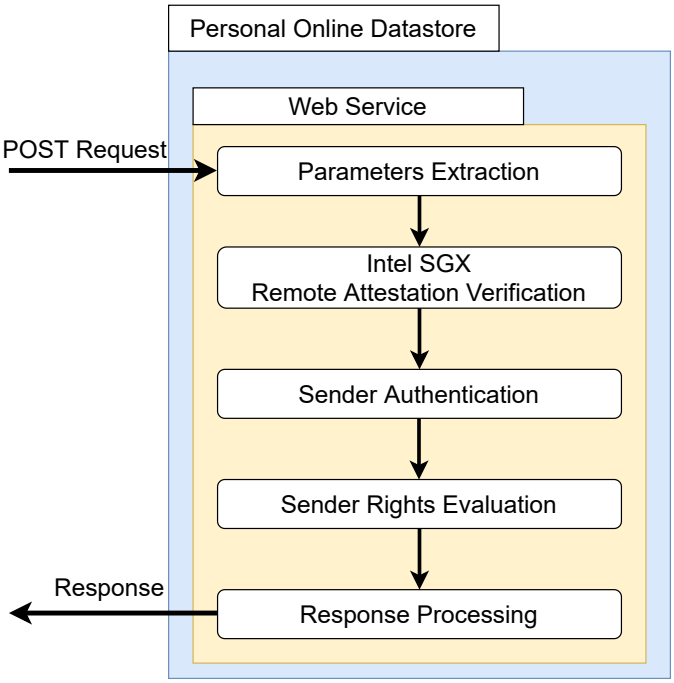

Figure 9. Main stages of the ReGov data provision instantiation process.

purpose is to use the `auth_token` (a message signed with the sender's credentials) and `claim` (the public key of the sender) parameters inside the `POST Request` to determine the sender `Node`'s identity. Specifically, `auth_token` refers to the URL of the resource to be accessed, encrypted with a private key. `DTauthenticator` is able to extract a public key from the `auth_token` parameter when the request is received. If the extracted public key is equal to the `claim` parameter, the identity of the sender `Node` is confirmed. At the end of the authentication procedure, Bob's `Web Service` identifies the sender of the request as Alice's `Node`.

The determined identity is subsequently evaluated by the `Web Service` during the `Sender Rights Evaluation` to determine whether the consumer `Node` can access the resource. Because our instantiation considers the decentralized environment related to the DecentralTrading data market (mentioned in Section 3), this step establishes whether the sender `Node` is associated with an active subscription (e.g., if Alice has an active subscription). However, the evaluation of alternative criteria, such as organization membership, can be freely integrated depending on the specific use case. In all cases, it is crucial to keep track of the consumer nodes that have accessed the `Personal Online Datastore`'s resources by establishing their identity.

Once the `POST Request` has passed the necessary checks, the `Response Processing` takes place. Therefore, the `Web Service` then interacts with the local storage to retrieve the requested resource, which, along with the associated policy, are inserted into the `Response`.

5.3 Trusted Execution Environment for Data Consumption

The `Trusted Execution Environment` manages the resources recovered within the consumer node. In Fig. 10, we propose a schematization of our `Trusted Application` implementation. The trusted application consists of two fundamental components: the `Trusted Part` and the `Untrusted Part`. The `Trusted Part` comprises one or more enclaves. The `Enclave`'s code is in the `enclave.cpp` file. It

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

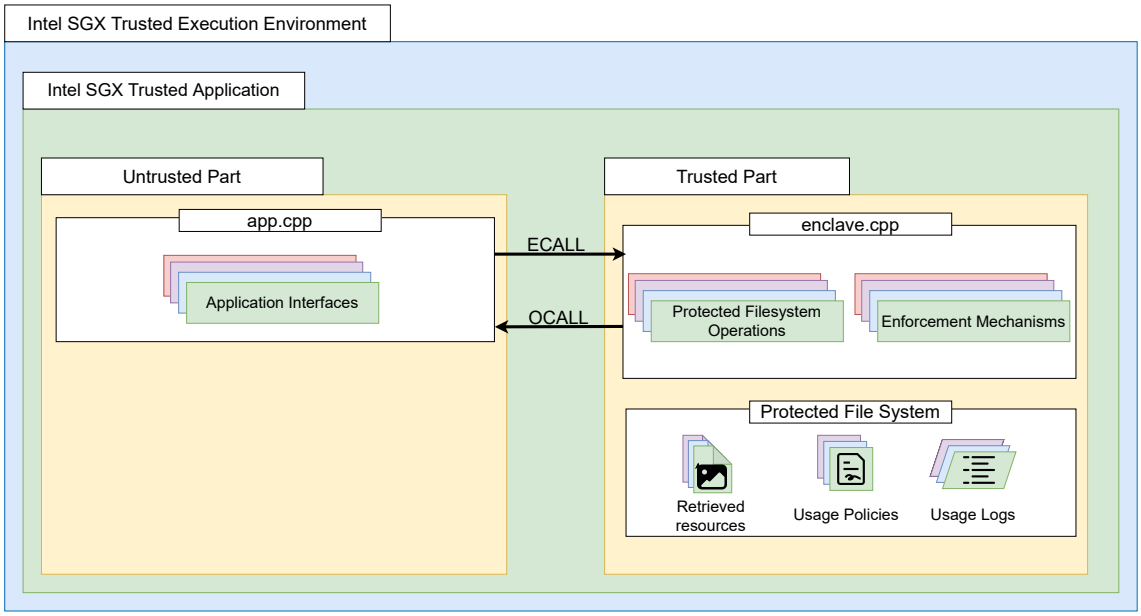

Figure 10. Schematization of our trusted application composed of both trusted and untrusted elements.

includes all the implementations of the `Enforcement Mechanisms` and a set of `Protected File System Operations` to handle the resources stored in it. The `Trusted Part` cannot communicate directly with the outside world. Any pieces of information that enter or leave the `Trusted Part` pass through the `Untrusted Part`. The `Untrusted Part`'s code is in the `app.cpp` file. This application has multiple `Application Interfaces` that are used to expose the application to the outside world. In order to communicate, the two parts use dedicated functions called `Ecall` and `Ocall`. 'Ecall' stands for Enclave Call and represents an invocation made by a function in the `Untrusted Part` to the `Enclave` (`Trusted Part`). The term 'Ocall' (Out Call) refers to a call from the `Enclave` to the `Untrusted Part`.

5.3.1 Data Protection

The main purpose of using the `Trusted Application` is to manage and protect the data of other users obtained from the market. The `Retrieved Resources` are stored within the `Enclave`, more specifically in its `Protected File System`, because in this way they are decrypted only within the processor and only the enclave itself can access the processor in order to decrypt it. Within the enclave, both the `Resources Retrieved` by the user and the `Usage Policies` set by the owner are stored. Storing the `Retrieved Resource` within the `Trusted Part` is essential both from a data protection and a usage control perspective. In addition, the `Usage Policy` chosen by the data owner must also be saved in a secure space, as it could be tampered with by malicious code in order to be bypassed.

Protection of usage data. When a user requests a piece of data, the request is received by the dedicated `Application Interface` in the `Untrusted Part`, and it is retrieved from the market. For instance, when Alice requests a photo of a Mesoplodon eueu from Bob, an identifier is assigned to this data before it is stored in the `Enclave`. The identifier associated with the resource is used to index the retrieved resources and store them within the trusted part. A copy of the policies set by the owner, the rules set by Bob for the photo, is associated with it in order to store all the necessary resource information in the enclave. More specifically, when Alice wants to retrieve a piece of data from Bob, she interacts with the `Untrusted Part` and sends a post HTTP request to Bob's node. Within the request parameters, the resource in which the consumer is interested is specified, and an identifier is provided with which the consumer gets authenticated

Pre-print copy of the manuscript published in **journal**
identified by DOI: `10.3389/fbloc.2023.1141909`

(as described in Section 5.2.2). Finally, a certificate provided by Intel SGX Remote Attestation is added to the request, providing evidence that the request comes from a `Trusted Application`. Once the `Personal Online Datastore` ensures that the other party involved in the communication is trusted, it sends the resource and policy information via an HTTP reply. Since the `Trusted Part` cannot communicate with the outside world, the response reaches the `Untrusted Part` who forwards it via an `Ecall` to the `Trusted Part`. Once the resource arrives at the `Trusted Part`, it stores the data sent from the `Personal Online Datastore` in the `Enclave` using the `Protected File System Operations` that allow the `Enclave` to manage the `Protected File System`. Based on the example scenario, at this point the photo of the Mesoplodon eueu and the related `Usage Policies` set by Bob, the owner, are stored within Alice's `Enclave`.

Protection of log data. To keep track of the correct use of resources, all actions performed on them within the `Trusted Part` are stored in a `usage log file`. In short, all actions concerning the retrieved resources are stored. The objective is to let the data owner initiate a monitoring procedure through an oracle, to check whether resources are used in accordance with usage conditions. When the `Untrusted Part` receives a monitoring request from the blockchain, it performs an `Ecall` to request a copy of the `Usage Log` file stored in the `Enclave` and returns it to the blockchain through an oracle to perform the monitoring. Referring to the example, all actions performed by Alice are recorded in a `Usage Log` file, and when Bob wants to check that everyone is using their resource correctly, he starts a monitoring procedure that aims to check all the `Usage Log` files of consumers who have retrieved the Mesoplodon eueu photos. When the `Usage Log` file is requested to be monitored, before sending a copy, the `Trusted Part` enters an entry to keep track of the monitoring request.

5.3.2 Implementation of the Enforcement Mechanisms

In order to guarantee that data are accessed and used according to usage policies when a resource from the `Trusted Part` of a `Trusted Application` is requested by an external application, `enforcement mechanisms` must be implemented. These mechanisms are implemented within the `Enclave` to ensure they are executed within a `Trusted Environment`.

Receiving a request for access to a resource stored in the trusted application. Before proceeding with the `Enforcement Mechanisms`, when the external application makes a request to the `Trusted Application`, the latter asks the external application to identify itself in order to check whether the sender is who it declares to be. More specifically, the `Untrusted Part` receives a request for access to a resource via the `Application Interfaces` and forwards it to the `Trusted Part` through an `Ecall` by invoking the `access_protected_resource` function, which verifies the identity of the claimant. Referring to the example, when Alice uses the 'Zooresearch' or 'Socialgram' applications, they have to authenticate themselves.

Retrieval of the requested resource and its usage policy. Once the external application has been authenticated, the `Trusted Application` gathers all the necessary information about it and accepts the request for the data that the external application is interested in and starts checking whether it is possible to access and use the resource. First, the `access_protected_resource` function retrieves the requested data and the associated policies, using the `get_policy` function, set by the owner. Then, the `access_protected_resource` function invokes the different enforcement modules, passing the retrieved policies to it, in order to ensure that the rules are satisfied. In our implementation, four different enforcement modules have been developed. The proposed approach is highly flexible, thus catering for the extension

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

of the existing rule types. The first mechanism in the enforcement process is checking the geographical position of the device.

Geographical rule enforcement. The `enforce_geographical` function is invoked and passed the policy for the requested resource. The `get_geo_location` function (`Ocall`) is then used to retrieve the geographic location of the device from which the resource is being accessed. In the end, the geographic data set by the user and the current location are compared. If the position is correct, a positive result is returned to the `access_protected_resource` function, otherwise access is denied. Referring to the scenario, the `Trusted Application` uses Alice's location to check if it meets the location stipulated by Bob in his usage policy.

Domain rule enforcement. The `access_protected_resource` function invokes the `enforce_domain` function by passing it the policy of the requested resource and information about the requesting application. Following a comparison between the application's domain and the domain set by the resource owner, if the domains are equal, the `enforce_domain` function returns a positive result to the `access_protected_resource` function, which proceeds to the next check. Otherwise, access to the resource is denied. Looking at the example scenario, the domain of the application used by Alice is checked to determine if it satisfies the usage domain set by Bob. If Alice's application domain is correct, a positive result is returned.

Access counter rule enforcement. The `enforce_access_counter` function is called by the `access_protected_resource` function with the policy for the requested resource. If the number of remaining accesses is greater than 1, the function decrements the maximum number of remaining accesses for that resource and returns with success to the `access_protected_resource` function. If the number of remaining accesses is equal to 1, the function removes the resource and related policies from the `Enclave` before returning a positive value, as the resource can no longer be accessed. In the motivating scenario, Bob set 100 as the maximum number of accesses to the resource. Each time Alice makes a request and logs in, the maximum number of hits left decreases. When the counter becomes 1, Alice is allowed a last access to the Mesoplodon eueu's photo, and then the resource is deleted from her `Trusted Application`. Then, having successfully completed all the enforcement, the `access_protected_resource` function forwards the contents of the file to the `Untrusted Part`, which forwards it to the external requesting application. As already mentioned, all actions performed on the resources in the trusted application are saved on a `Usage Log` file, which keeps information and accesses made on the resources from when it is retrieved until it is deleted, maintaining an overview of the use of the resource. This `Usage Log` file makes it possible to prove and check that all resources have been used correctly within the trusted application.

Temporal rule enforcement. When it comes to temporal rules, the `Untrusted Part` periodically invokes the `Ecall` function called `enforce_temporal` to verify that all resources within the trusted part have not expired. The `enforce_temporal` function uses the `get_trusted_time` function to retrieve the current day. It then reads all resource policies stored within the `Trusted Part` and checks whether the date set on the policy is later than the current date. If a resource has expired, the `enforce_temporal` function removes it. Each time this type of check is performed, it is written to the `Usage Log` file, and all deletions are also saved.

5.4 Blockchain as a Governance Ecosystem

In our instantiation, we leverage blockchain smart contracts in order to realize the `Governance Ecosystem`. Transparency, distribution, and immutability are the key features that make this technology highly suitable for our needs. The DecentralTrading implementation leverages the `EVM Blockchain`

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

platform hosting several interconnected smart contracts. Nodes of the decentralized environment that are equipped with confidential blockchain public and private keys, sign authenticate transactions that generate the execution of smart contract functions. Processes that involve data exchange between Nodes and smart contracts are supported by blockchain oracles.

We implemented the smart contracts using the Solidity programming language[14]. The smart contracts have been deployed in a local environment powered by the Ganache tool[15] which enables the execution of a local blockchain replicating the Ethereum protocol and supporting the generation of transactions for testing purposes. In the following, we present the implementation details regarding the DTindexing and DTobligations smart contracts that fulfill the functionality of the Resource Indexing and Policy Governance components respectively.

5.4.1 DTindexing Smart Contract

The DTindexing smart contract caters for the initialization of shared resources in the decentralized environment. The main goal of this component is to keep track of the decentralized environment's data. Owner nodes interact with the smart contract to index their Personal Online Datastore, sharing the necessary metadata for data retrieval. Consumer nodes make use of the smart contract to find references for registered resources through search functionality. Table 2 represents the class diagram of the smart contract. The smart contract saves the following variables in the Pod struct in order to keep track of the information about personal online datastores:

```
struct Pod { int id; address owner; bytes baseUrl; bool isActive; }
```

Similarly, the contract stores information about resources in a Resource struct, which consists of the following:

```
struct Resource{ int id; address owner; int podId; bytes url; bool isActive; }
```

The Pod and Resource structs are stored in the podList and resourceList array variables, respectively. The contract includes several methods for interacting with online datastores and resources, including the ability to register new ones, deactivate existing ones, and to search for them based on various criteria. For example, the registerPod method allows nodes to initialize new personal online datastores in the network. It takes as input a web reference for the online datastore service and the public key of the owner Node. The function creates a new Pod struct and stores it in the podList. It also deploys a DTobligations smart contract (discussed next in detail), as every Personal Online Datastore is related to one of these contracts. Finally, the function emits a NewPod event containing the identifier and the address of the DTobligations smart contract for the new online datastore. In our running example, Bob's node invokes this function to initialize his new Personal Online Datastore providing the web reference https://BobNode.com/ among the arguments. The function, in turn, generates a new Pod struct. The registerResource method works similarly, generating a new Resource object and storing it in the resourceList state variable. In this case, Bob's Personal Online Datastore employs this function to initialize the 'Mesoplodon.jpg' image providing metadata such as the https://BobNode.com/images/Mesoplodon.jpg url. The deactivateResource and deactivatePod methods ensure that personal online datastores and resources are no longer accessible. Nodes submit metadata referring to new datastores and resources by using push-in oracles, that enable

[14] https://docs.soliditylang.org/en/v0.8.17/. Accessed: Thursday 11th May, 2023.

[15] https://trufflesuite.com/ganache/. Accessed: Thursday 11th May, 2023.

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

Table 2. Class diagram of the DTindexing smart contract.

DTindexing
private podsCounter: int private resourceCounter: int private dtSubscription: int private podList: Pod[] private resourceList: Resource[]
private searchByType(tp: PodType): Pod[] <<event>> NewPod(idPod: int, obgliationAddress: address) <<event>> NewResource(idResource: int) <<modifier>> validPodId(id: uint, owner: address) public getMedicalPods(idSubscription: uint): Pod[] public getSocialPods(idSubscription: uint): Pod[] public getFinancialPods(idSubscription: uint): Pod[] public registerPod(newReferene: bytes, podType: PodType, podAddress: address): int public registerResource(podId: int, newReferene: bytes, idSubscription: uint): int <<validPodId>> public getPodResources(podId: int, idSubscription: int): Resource[] public deactivateResource(idResource: int): Resource <<validResourceId>>

sending information to the blockchain. The smart contract also offers various search functions that can be useful for consumer nodes. The `getPodResources` method allows users to obtain a list of `Resource` structs stored in a specific datastore, identified by its integer identifier. The `getResource` method accepts an integer identifier as input and returns the `Resource` struct with that identifier. Referring to our use case scenario, Alice uses `getPodResources` to read the image's identifier that is given as a parameter to `getResource`, thanks to which the associated web reference is retrieved.

5.4.2 DTobligations Smart Contract

We use the `DTobligations` smart contract to model usage policies inside the blockchain environment and execute their monitoring. The architecture of the implementation assumes the deployment of multiple instances of the smart contract, one for each `Personal Online Datastore` in the network. Each `DTobligations` smart contract is associated with a specific `Personal Online Datastore` that is the only entity allowed to establish and manage the rules associated with the stored resources. As we showed in our motivating scenario, the architecture of our implementation assumes the deployment of a dedicated `DTobligations` instance containing the rules for Bob's `Personal Online Datastore`. In Table 3, we propose the class diagram of the `DTobligations` smart contract.

The `DTobligations` smart contract includes four structs, each of which, models a specific rule: `AccessCounterObligation`, which restricts the number of resource accesses on a client device; `CountryObligation`, which imposes restrictions on the countries in which a resource can be used; `DomainObligation`, which specifies the purposes for which resources can be used; and `TemporalObligation`, which imposes a maximum duration for resource storage. These are stored in an `ObligationRules` struct, which can apply to a specific resource or to the entire `Personal Online Datastore`. The smart contract includes functions that allow nodes to set default rules for their `Personal Online Datastore` and related resources. For instance, the `addDefaultAccessCounterObligation` and `addDefaultTemporalObligation` are used to set rules that are inherited by all the resources of the `Personal Online Datastore`. Similarly, functions such as `addAccessCounterObligation` and `addTemporalObligation` establish rules that are applied to a specific resource of the datastore. Referring to our running example, Bob's `Personal Online Datastore` invokes the `addTemporalObligation` giving as input the 'Mesoplodon.jpg' identifier and the integer value that describes the time duration of 20 days. The `onlyOwner` modifier ensures that certain functions can only be invoked by using the blockchain

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

Table 3. Class diagram of the DTobligations smart contract.

DTobligations <<extends >> Ownable
dtIndexing: DTindexing defaultPodObligation: ObligationRules resourcesObligation: mapping(int=>ObligationRules)
<<modifier>>hasSpecificRules(resourceId: int) <<modifier>>isValidTemporal(deadline: uint) <<modifier>>isTheResourceCovered(idResource: int) public constructor(dtInd: address, podAddress: address) public getObligationRules(idResource: int): ObligationRules <<isTheResourceCovered>> public getDefaultObligationRules(): ObligationRules public addDefaultAccessCounterObligation(accessCounter: uint) public addDefaultTemporalObligation(temporalObligation: uint) <<isValidTemporal, onlyOwner>> public addDefaultCountryObligation(country: uint) <<onlyOwner>> public addDefaultDomainObligation(domain: DomainType) <<onlyOwner>> public addAccessCounterObligation(idResource: int, accessCounter: uint): ObligationRules <<isTheResourceCovered, onlyOwner>> public addDomainObligation(idResource: int, domain: DomainType): ObligationRules <<onlyOwner, isTheResourceCovered>> public addCountryObligation(idResource: int, country: uint): ObligationRules <<onlyOwner, isTheResourceCovered>> public addTemporalObligation(idResource: int, deadline: uint): ObligationRules <<onlyOwner, isTheResourceCovered, isValidTemporal>> public removeAccessCounterObligation(idResource: int) <<onlyOwner, isTheResourceCovered, hasSpecificRules>> public removeTemporalObligation(idResource: int) <<isTheResourceCovered, onlyOwner, hasSpecificRules>> public removeDomainObligation(idResource: int) <<isTheResourceCovered, onlyOwner, hasSpecificRules>> public removeCountryObligation(idResource: int) <<isTheResourceCovered, onlyOwner, hasSpecificRules>> public removeDefaultTemporalObligation() <<onlyOwner>> public removeDefaultAccessCounterObligation() <<onlyOwner>> public removeDefaultCountryObligation() <<onlyOwner>> public removeDefaultDomainObligation() <<onlyOwner>> public withSpecificRules(idResource: int): bool public monitorCompliance() <<onlyOwner>>

credentials associated with the smart contract's owner. It is applied to the functions for rule modification, which can be invoked only by the owner `Node`. In this way, Bob is sure that modification of the rules can only be executed by his `Personal Online Datastore`.

The main goal of the monitoring procedure is to retrieve evidence from consumer nodes attesting to the utilization of resources, whose policies are represented by the `DTobligations` instance. The smart contract implements the `monitorCompliance` function, solely invocable by the contract owner, to initiate the monitoring procedure. When the function is used, it interacts with a pull-in oracle, that is able to retrieve external information outside the blockchain. Therefore, the `DTobligations` smart contract communicates with the on-chain component of the oracle (i.e. smart contract named `PullInOracle`) by invoking its `initializeMonitoring` function. The oracle generates a new `MonitoringSession` struct instance that contains information about the current state of the session and aggregates the external responses. The same function emits a `NewMonitoring` event. The emission of the event is caught by the off-chain components of the oracle, running in consumer nodes, that forward to the `SGX Intel Trusted Application` the command to provide the usage log of the resources involved. Once the usage log is retrieved, the information contained within it are sent to the on-chain component of the oracle through its `_callback` method. The function aggregates the responses from consumer nodes and updates the involved `MonitoringSession` instance each time it is called. Once all the responses are collected, they are returned to the `DTobligations` smart contract at the end of the process. In our running example, the procedure is started by Bob's `Personal Online Datastore` using the `monitorCompliance` function. Subsequently, Alice's `SGX Trusted Application` is contacted by the pull-in oracle and it is asked to provide the usage log of the 'Mesoplodon.jpg' resource. Alice's response contains information such as the number of local accesses to the image or the time from its retrieval. The evidence provided by Alice's

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

`SGX Trusted Application` is collected, together with pieces of evidence provided by other nodes in the network, by the pull-in oracle. Finally, the oracle forwards the logs to Bob's instance of `DTindexing`.

6 EVALUATION

We evaluate the implementation of the ReGov framework by taking two distinct approaches. In the first part of this section we revisit the specific requirements usage control requirements that were derived from out motivating scenario. While, in the second part, we examine the security, privacy, and affordability of our implementation.

6.1 Requirement Verification

In this section, we discuss how the previously established requirements are satisfied by our ReGov instantiation, following the methodology described in the study of Terry Bahill and Henderson (2005). Through the discussion of the requirements, we contextualize the use of the trusted execution environment and the blockchain respectively in our architecture. Both requirements are composed of several sub-requirements that express various environmental and technological functions.

6.1.1 (R1) Resource utilization and policy fulfillment must be managed by trusted entities

The first requirement (**R1**) stipulates that **resource utilization and policy fulfillment must be managed by trusted entities**. We employ a trusted execution environment in order to develop a trusted application executable inside our nodes. We implemented it using Intel SGX, as explained in Section 5.3. Our design and implementation choice allows us to satisfy the following sub-requirements:

(R1.1) The trusted entity must be able to store resources obtained from other entities. In the proposed ReGov framework instantiation, all resources retrieved from the data market by the untrusted part of a node are passed to the trusted part of a node in order to store them within the enclave. For storage, we use an Intel SGX function, called Protected File System Library, which allows the management of files containing the resources retrieved within the enclave. We chose to store the data in the enclave because any information stored in it is encrypted and decrypted solely by the enclave.

(R1.2) The trusted entity must support the execution of programmable procedures that enforce constraints associated with resource usage. When a resource stored within the enclave is requested, before retrieving it, the enclave we have implemented executes all the application procedures provided by the resource policy, invoking the necessary enforcement functions. The proposed enclave only allows access to the resource if at the end of the execution of all enforcement procedures, all of them have given a positive result. Otherwise, the resource is not returned and access is denied. It is worth noting that the enforcement mechanism within the trusted application is implemented in a modular way. Although our current implementation is limited to four rule types, this feature allows developers to easily extend our implementation with additional rule types based on their specific needs.

(R1.3) Resources and procedures managed by the trusted entity must be protected against malicious manipulations. In the proposed ReGov implementation, we store resources within the enclave, because it is secure and protected from unauthorized access. The trusted part cannot communicate directly with the outside world and thus avoids interacting with malicious software. In addition, all code included and executed in the trusted part is, in turn, trusted, as it is not possible to use third-party libraries. The data stored within the enclave are encrypted. Therefore, a direct attack on the memory by malicious software would not be able to read the data.

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

(R1.4) The trusted entity must be able to prove its trusted nature to other entities in a decentralized environment. When it comes to interaction between nodes, in order to prove a node's trustworthiness, we employ the Intel SGX remote attestation within our trusted application. This advanced feature allows a node to gain the trust of a remote node. The provided attestation ensures that the node is interacting with a trusted application using an updated Intel SGX enclave.

6.1.2 (R2) Policy compliance must be monitored via the entities of a governance ecosystem

The second requirement (**R2**) stipulates that **policy compliance must be monitored through entities running in a governance ecosystem**. In our ReGov framework, we propose the adoption of a governance ecosystem that we instantiate on top of blockchain technology. In the following, we show the suitability of blockchain for this role by addressing each sub-requirement.

(R2.1) The governance ecosystem must provide transparency to all the nodes of the decentralized environment. By allowing all nodes to view the complete transaction history of the blockchain technology, we are able to ensure that each participant of the decentralized environment has equal access to information and is able to independently verify the accuracy and integrity of governance data. Additionally, we implement the policy management tasks via smart contracts, the code for which is made publicly available within the blockchain infrastructure. This enables nodes in the decentralized environment to be aware of the governance processes that are being executed.

(R2.2) Data and metadata maintained by the governance ecosystem must be tamper-resistant. Our solution involves the storage of resource metadata and usage policies in data structures that are part of smart contracts. Through smart contracts functions, we implement functionality that can be used to upload and modify stored data. We leverage the asymmetric key encryption mechanism of the blockchain environment to verify that data modifications are performed by authorized users. Once data and metadata of ReGov are validated in a blockchain block, we rely on the cryptographic structure underlying the blockchain to guarantee the integrity of published smart contracts and the information contained therein.

(R2.3) The governance ecosystem and the entities that the form part of the ecosystem must be aligned with the decentralization principles. We fulfill the decentralization principles by proposing a blockchain-based architecture that is inherently decentralized. In our implementation, we publish data and metadata through a network of validators rather than a central authority. This ensures that no single entity has control over shared data and smart contracts that are distributed in the blockchain ecosystem. Through decentralization, we secure the fairness and integrity of policy management and prevent any single authority of the decentralized environment from having too much control or disproportionate decision-making power.

(R2.4) The entities that form part of the governance ecosystem must be able to represent policies and verify their observance. The majority of smart contract technologies are characterized by Turing-complete programming languages. We use the expressive power of smart contracts to implement data structures that can be used to represent usage policies and automate their monitoring. We facilitate the communication between smart contracts and off-chain nodes by integrating oracle technologies that implement the protocols for data-exchange processes.

6.2 Architecture Discussion

In this section, we broaden our discussion on the effectiveness of the proposed decentralized usage control architecture with a particular focus on privacy, security, and affordability. The criteria the discussion is based on have been inspired by the work of Ferrag and Shu (2021).

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

6.2.1 Security

Several works already show how the decentralized model makes it more difficult for attackers to compromise data, as they would need to gain access to multiple nodes rather than just one central server (Raman et al., 2019; Alabdulwahhab, 2018). As per the vast majority of decentralized web initiatives, our implementation preserves the security of data residing in nodes through the `Personal Online Datastore` component, which performs authentication and rights evaluation procedures to prevent unauthorized access to sensitive information or resources.

Our solution introduces new components into the decentralized model whose security should be discussed. The metadata stored in smart contracts (usage policies and resource indexes) are protected from unauthorized updates through the consensus mechanism of the blockchain platform and its distributed nature, which makes this information immutable. Moreover, the state of distributed applications running in this environment can only be changed by transactions marked by a digital signature. This feature guarantees that usage policy modifications can only be executed by authorized entities.

The `Intel SGX Trusted Execution Environment` provides a separate ecosystem for the execution of a `Trusted Application` that manages resource utilization. It has already shown its effectiveness in terms of preventing the injection of malicious code coming from the operating system of the client's machine (Sabt et al., 2015), which could jeopardize the integrity of the stored resources and the local representation of usage policies. Moreover, we also leverage the security guarantees offered by this technology to establish a protected environment in which the enforcement of the usage policies is ensured, inside the consumer's node.

The monitoring process, thanks to which nodes get evidence of the utilization of their resources, involves the interaction between the `EVM Blockchain` and consumer nodes. The procedure involves the exchange of confidential information, the integrity of which must be secured. Interactions between the involved components are managed via blockchain oracles that are capable of ensuring the legitimacy operations (Al-Breiki et al., 2020b). By definition, oracles establish secure communication protocols that enable on-chain and off-chain computations to send and receive data safely.

Security and verification of data consumption are enforced by the ensemble of smart contracts, trusted execution environments, and remote attestations. Through the latter, data providers are able to remotely verify the integrity of a node's data consumption component and thwart attempts to instantiate malicious consumer nodes in the decentralized environment. Nevertheless, data provision of inappropriate information through published data is a practice that requires automated ex-post checking and whistleblowing (Kirrane and Di Ciccio, 2020).

We remark that ReGov cannot supervise users' actions outside the digital context of the decentralized environment. For example, it is unable to prevent users from taking a picture of a protected image resource using a separate camera, or copying reserved information displayed on the screen. The framework is intended to operate at the digital level. Therefore, ReGov monitors and controls data access, processing, and distribution, ensuring that it is utilized in compliance with the associated policy. Our motivating scenario resorts to a list of approved applications that guarantee fair data elaboration and facilitate misconduct uncovering. Considering the running example, applications like "Socialgram" put in place procedures that counteract OS screen recording actions. In addition, unfair activities that break the enforcement mechanism can be detected by the presented monitoring routines, enabling data owners to indict malicious users.

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

6.2.2 Privacy

Privacy is key for decentralized web environments trying to take personal data out of the control of single organizations. With usage control, users can benefit from a greater level of privacy, as they have a way to determine how their resources are being used. However, enforcement and monitoring mechanisms that characterize usage control involve the exchange of data and metadata whose confidentiality should constantly be guaranteed.

One of the most critical issues of our solution regarding confidentiality relates to the blockchain metadata, which are publicly exposed in smart contracts. Public blockchains, such as Ethereum, provide public ledgers, thus allowing every node of the decentralized environment to get access to usage policy and resource locations. Despite the possibility of specifying private variables in smart contracts, the method invocations thanks to which those variables are set are recorded in publicly readable transactions. Therefore, blockchain users can freely deduce the state of a private variable by inspecting the public transactions associated with the invocation of the setter methods. In some use cases, it may be desirable to keep this data public. However, there may also be a need to encrypt data stored in the blockchain, so that only authorized parties (those that have access to the decryption key) can read this metadata (Pan et al., 2011; Marangone et al., 2022).

The confidentiality of the shared resources must be regulated after their retrieval inside consumer nodes, in order to apply the constraints associated with their policy rules. Our implementation leverage the `Intel SGX Trusted Execution Environment` that manages retrieved resources through the `SGX Protected File System (PFS)`. One of the key features of SGX-PFS is that it allows for files to be stored in a secure, encrypted format, even when the operating system is not running. This makes it difficult for attackers to access the resources, as they would need to have physical access to the machine and be able to bypass the SGX hardware security features in order to read the contents of the files.

6.2.3 Affordability

The affordability of our solution is strongly related to the costs associated with the smart contracts running in the blockchain ecosystem. `EVM Blockchains` associate the execution of smart contracts with a fee charged to the invoking user, according to the complexity of the code to be executed. This fee is measured in (units of) Gas. In Table 4, we collect the Gas expenses associated with the functions of the `DTobligations` and `DTindexing` smart contracts. The table omits their read functions, for which no transactions need to be sent to the network.

The deployment cost of `DTindexing` is $3{,}255{,}000$ Gas units. The `registerPod` method is the most expensive `DTindexing`'s function ($2{,}082{,}494$ Gas units) as it involves the deployment of a new contract instance, too. The Gas consumption of `registerResource` turns out to be significantly lower, requiring $143{,}004$ Gas units. The least expensive function of the smart contract is `deactivateResource` with an expenditure of $21{,}465$ Gas units.

`DTobligations` is deployed during the registration of a new personal online datastore at the cost of $2{,}057{,}988$ Gas units. `DTobligations` offers methods and functions to modify the obligation rules related to the resources contained in personal online datastore. Among the functions for adding rules, the most expensive one is `addAccessCounterObligation` with a value of $138{,}768$ Gas units. However, the adding of a domain restriction through `addDefaultDomainObligation` costs significantly less with $44{,}219$ Gas units per invocation. Methods for rule deactivation determine a lower expense than the previous ones. The

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

Table 4. Gas expenditure of the DTobligations and DTindexing smart contracts. Costs are expressed in Gas units.

DTobligations	
Function	**Cost**
deployment	2,057,988
`addDefaultAccessCounterObligation(···)`	62,627
`addDefaultTemporalObligation(···)`	62,638
`addDefaultDomainObligation(···)`	44,219
`addDefaultCountryObligation(···)`	62,561
`addAccessCounterObligation(···)`	138,768
`addTemporalObligation(···)`	97,737
`addCountryObligation(···)`	97,728
`addDomainObligation(···)`	79,452
`removeDefaultAccessCounterObligation(···)`	23,780
`removeDefaultTemporalObligation(···)`	16,079
`removeDefaultDomainObligation(···)`	24,747
`removeDefaultCountryObligation(···)`	23,758
`removeAccessCounterObligation(···)`	28,184
`removeTemporalObligation(···)`	28,151
`removeCountryObligation(···)`	28,173
`removeDomainObligation(···)`	38,111
`monitorCompliance(···)`	42,000

DTindexing	
Function	**Cost**
deployment	3,255,000
`registerPod(···)`	2,082,494
`registerResource(···)`	143,004
`deactivateResource(···)`	21,465

cheapest among them is `removeDomainObligation` (16,079 Gas units). The cost required to initialize a monitoring process through the `monitorCompliance` function is 42,000 units of Gas.

As expected, operations involving new smart contract deployments are the most expensive ones. However, these costs are associated with one-time operations performed at setup time (at the bootstrapping of the platform, or every time a new pod is registered). On the other hand, functions intended for more frequent invocations (e.g., to monitor compliance or update rules) are characterized by significantly lower costs. Costs in fiat money are subject to high variability, as they depend on multiple factors including the network capacity utilization, the price in cryptocurrency per Gas unit, and the market exchange rate of the cryptocurrency. Also, these values change depending on the EVM blockchain in use (e.g., Ethereum[16], Avalanche[17], Polygon[18], and more). At the time of writing, we empirically found variations of four orders of magnitude[19]. However, we remark that our implementation costs align with ERC721 implementations[20]. For example, the deployment fees of the Ethereum Name Service (ENS)[21], a non-fungible token in the

[16] https://ethereum.org/. Accessed: Thursday 11th May, 2023.

[17] https://www.avax.com/. Accessed: Thursday 11th May, 2023.

[18] https://polygon.technology/. Accessed: Thursday 11th May, 2023.

[19] The amount of gas needed for the deployment of the `DTindexing` smart contract, e.g., is 3,255,000. During our experiments, the price per Gas unit in the Ethereum public network amounted to 36.15 Gwei (one GWei is worth 10^{-9} ETH). The ETH/EUR exchange rate was 1/1590 EUR. The total gas cost price was thus 187.09 EUR. Other EVM blockchains exhibited lower Gas prices or exchange rates, decreasing the overall cost in fiat money. Considering the Avalanche and Polygon platforms, their Gas price was 42.56 and 168.65 Gwei, respectively. The AVAX/EUR exchange rate was 1/15.67, and the MATIC/EUR exchange rate was 1/1.19. As a result, the total expenses amounted to 2.17 and 0.65 EUR, respectively. Data collected: 14 March 2023, 11:30 pm. Our smart contract deployments can be found on the Görli Ethereum test network at https://goerli.etherscan.io/address/0xb0fe7d07947d9dd7cda47825e61ec14b98ef271a, on the Fuji Avalanche test network at https://testnet.snowtrace.io/address/0x0082698263ccc5765c97404af39023daefe20096, and on the Mumbai Polygon test network at https://mumbai.polygonscan.com/address/0x9ee2cb5ef7b1449d615d9fd0f9b167543e0d28eb.

[20] https://eips.ethereum.org/EIPS/eip-721. Accessed: Thursday 11th May, 2023.

[21] https://etherscan.io/token/0xc18360217d8f7ab5e7c516566761ea12ce7f9d72. Accessed: Thursday 11th May, 2023.

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

neighboring area of personal information indexing, amounts to 2,443,978 Gas units[22]. The market scenario can support the structural expenses associated with the proposed implementation and provides an incentive system that allows users to earn money by sharing their data. However, cost reduction practices are necessary to increase usability. These include design improvements to the implementation's architecture as well as the adoption of side-chains and layer-2 networks.

7 CONCLUSION

Since its inception, the web has evolved from a read-only medium for information dissemination to a ubiquitous information and communication platform that supports interaction and collaboration globally. Although the web is by design decentralized and thus is not controlled by any single entity or organization, the web as we know it today is dominated by a small number of centralized platforms. Consequently, the decentralized web initiative aims to promote research into tools and technologies that give data owners more control over their data and enable smaller players to gain access to data, thus enabling innovation.

In this paper, we focus specifically on resource governance in a decentralized web setting. We extend the state of the art by proposing a conceptual resource governance framework, entitled ReGov, that facilitates usage control in a decentralized setting, with a particular focus on policy respecting resource utilization and resource indexing and continuous monitoring. In order to demonstrate the potential of our ReGov framework, we propose a concrete instantiation that employs a trusted execution environment to cater for the former, and blockchain technologies to facilitate the latter. The effectiveness of the ReGov framework and our particular instantiation is assessed via a detailed analysis of concrete requirements derived from a data market motivating scenario and an assessment of the security, privacy, and affordability aspects of our proposal.

Future work includes extending our primitive rule syntax to encompass more expressive usage control policies that are based on standard policy languages. Additionally, we plan to explore strategies for reducing the costs associated with the smart contracts running in the blockchain ecosystem. Studying incentivization mechanisms to encourage users to use the platform and possibly gain rewards for sharing information also paves the path for future endeavors.

The community-based categorization of applications interfaced with ReGov is a challenging aspect, the solution to which potentially involves the adoption of dedicated smart contracts for voting and arbitrage mechanisms. Also, erroneous or malicious misuse of ReGov such as the publication and disclosure of otherwise private information is beyond the reach of ReGov and would entail ex-post patrolling of the system. Studying these integrations with our framework is a task we envision for future work. Finally, we aim to conduct case studies with users to evaluate our approach in real-world settings.

Acknowledgments.

The work of D. Basile, C. Di Ciccio, and V. Goretti was partially funded by the Italian Ministry of University and Research under grant "Dipartimenti di eccellenza 2018-2022" of the Department of Computer Science at Sapienza, by the EU-NGEU NRRP MUR under grant PE00000014 (SERICS), by the Cyber 4.0 project BRIE, and by the Sapienza project "Drones as a Service for First Emergency Response". The work of S. Kirrane was funded by the FWF Austrian Science Fund and the Internet Foundation Austria under the FWF Elise Richter and netidee SCIENCE programmes as project number V 759-N.

[22] https://etherscan.io/tx/0xff3ee18523c9ec20e62d31d3d3ce3e8bf25f5ffcdfc4c32cd43ed0a786cc8640. Accessed: Thursday 11th May, 2023.

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

REFERENCES

Akaichi, I. and Kirrane, S. (2022a). A semantic policy language for usage control. In *SEMANTiCS (Posters & Demos)* (CEUR-WS.org), 10:1–10:5

Akaichi, I. and Kirrane, S. (2022b). Usage control specification, enforcement, and robustness: A survey. *arXiv preprint arXiv:2203.04800*

Al-Breiki, H., Rehman, M. H. U., Salah, K., and Svetinovic, D. (2020a). Trustworthy blockchain oracles: Review, comparison, and open research challenges. *IEEE Access* 8, 85675–85685

Al-Breiki, H., Rehman, M. H. U., Salah, K., and Svetinovic, D. (2020b). Trustworthy blockchain oracles: Review, comparison, and open research challenges. *IEEE Access* 8, 85675–85685

Alabdulwahhab, F. A. (2018). Web 3.0: The decentralized web blockchain networks and protocol innovation. In *2018 1st International Conference on Computer Applications & Information Security (ICCAIS)*. 1–4. doi:10.1109/CAIS.2018.8441990

Ayoade, G., Karande, V., Khan, L., and Hamlen, K. (2018). Decentralized IoT data management using blockchain and trusted execution environment. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. 15–22. doi:10.1109/IRI.2018.00011

Bai, G., Yan, L., Gu, L., Guo, Y., and Chen, X. (2014). Context-aware usage control for web of things. *Security and Communication Networks* 7, 2696–2712

Basile, D., Goretti, V., Di Ciccio, C., and Kirrane, S. (2021). Enhancing blockchain-based processes with decentralized oracles. In *BPM (Blockchain and RPA Forum)*. 102–118

Becker, H., Vu, H., Katzenbach, A., Braun, C. H., and Käfer, T. (2021). Monetising resources on a solid pod using blockchain transactions. In *The Semantic Web: ESWC 2021 Satellite Events*. 49–53

Bonatti, P. A., Kirrane, S., Petrova, I. M., and Sauro, L. (2020). Machine understandable policies and GDPR compliance checking. *KI-Künstliche Intelligenz* 34, 303–315

Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform. *white paper* 3, 2–1

Cai, T., Yang, Z., Chen, W., Zheng, Z., and Yu, Y. (2020). A blockchain-assisted trust access authentication system for solid. *IEEE Access*

Carroll, E. L., McGowen, M. R., McCarthy, M. L., Marx, F. G., Aguilar, N., Dalebout, M. L., et al. (2021). Speciation in the deep: genomics and morphology reveal a new species of beaked whale mesoplodon eueu. *Proceedings of the Royal Society B* 288, 20211213

Costan, V. and Devadas, S. (2016). Intel sgx explained. *Cryptology ePrint Archive*

Esteves, B. and Rodríguez-Doncel, V. (2022). Analysis of ontologies and policy languages to represent information flows in GDPR. *Semantic Web* , 1–35

Ferrag, M. A. and Shu, L. (2021). The performance evaluation of blockchain-based security and privacy systems for the internet of things: A tutorial. *IEEE Internet of Things Journal* 8, 17236–17260. doi:10.1109/JIOT.2021.3078072

Grünbacher, A. (2003). POSIX access control lists on linux. In *Proceedings of the FREENIX Track: 2003 USENIX Annual Technical Conference*. 259–272

Havur, G., Vander Sande, M., and Kirrane, S. (2020). Greater control and transparency in personal data processing. In *International Conference on Information Systems Security and Privacy (ICSSP)*. 655–662. doi:10.5220/0009143206550662

Hilty, M., Pretschner, A., Basin, D., Schaefer, C., and Walter, T. (2007). A policy language for distributed usage control. In *European Symposium on Research in Computer Security* (Springer), 531–546

Jauernig, P., Sadeghi, A.-R., and Stapf, E. (2020). Trusted execution environments: properties, applications, and challenges. *IEEE Security & Privacy* 18, 56–60

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

Johnson, D., Menezes, A., and Vanstone, S. (2001). The elliptic curve digital signature algorithm (ecdsa). *International journal of information security* 1, 36–63

Khan, M. Y., Zuhairi, M. F., Syed, T. A., Alghamdi, T. G., and Marmolejo-Saucedo, J. A. (2020). An extended access control model for permissioned blockchain frameworks. *Wirel. Networks* 26, 4943–4954

Kirrane, S. and Di Ciccio, C. (2020). BlockConfess: Towards an architecture for blockchain constraints and forensics. In *AIChain@Blockchain* (IEEE), 539–544. doi:10.1109/Blockchain50366.2020.00078

Koshutanski, H. and Massacci, F. (2003). An access control framework for business processes for web services. In *Proceedings of the 2003 ACM workshop on XML security*. 15–24

Lazouski, A., Martinelli, F., and Mori, P. (2010). Usage control in computer security: A survey. *Computer Science Review* 4, 81–99

Liang, X., Shetty, S., Zhao, J., Bowden, D., Li, D., and Liu, J. (2017). Towards decentralized accountability and self-sovereignty in healthcare systems. In *International conference on information and communications security* (Springer), 387–398

Lind, J., Eyal, I., Kelbert, F., Naor, O., Pietzuch, P., and Sirer, E. G. (2017). Teechain: Scalable blockchain payments using trusted execution environments. *arXiv preprint arXiv:1707.05454*

Mammadzada, K., Iqbal, M., Milani, F., García-Bañuelos, L., and Matulevicius, R. (2020). Blockchain oracles: A framework for blockchain-based applications. In *BPM (Blockchain and RPA Forum)* (Springer), 19–34

Marangone, E., Di Ciccio, C., and Weber, I. (2022). Fine-grained data access control for collaborative process execution on blockchain. *arXiv preprint arXiv:2207.08484*

McGillion, B., Dettenborn, T., Nyman, T., and Asokan, N. (2015). Open-tee–an open virtual trusted execution environment. In *2015 IEEE Trustcom/BigDataSE/ISPA* (IEEE), vol. 1, 400–407

Mohanty, D. (2018). Ethereum for architects and developers. *Apress Media LLC, California* , 14–15

Mühlberger, R., Bachhofner, S., Ferrer, E. C., Di Ciccio, C., Weber, I., Wöhrer, M., et al. (2020). Foundational oracle patterns: Connecting blockchain to the off-chain world. In *BPM (Blockchain and RPA Forum)* (Springer), 35–51

Neisse, R., Pretschner, A., and Di Giacomo, V. (2011). A trustworthy usage control enforcement framework. In *2011 Sixth International Conference on Availability, Reliability and Security*. 230–235. doi:10.1109/ARES.2011.40

Ouaddah, A., Abou Elkalam, A., and Ait Ouahman, A. (2016). Fairaccess: a new blockchain-based access control framework for the internet of things. *Security and communication networks* 9, 5943–5964

Pan, J., Paul, S., and Jain, R. (2011). A survey of the research on future internet architectures. *IEEE Communications Magazine* 49, 26–36

Park, J. and Sandhu, R. (2004). The uconabc usage control model. *ACM transactions on information and system security (TISSEC)* 7, 128–174

Pasdar, A., Lee, Y. C., and Dong, Z. (2022). Connect API with blockchain: A survey on blockchain oracle implementation. *ACM Comput. Surv.* doi:10.1145/3567582

Patel, S., Sahoo, A., Mohanta, B. K., Panda, S. S., and Jena, D. (2019). Dauth: A decentralized web authentication system using ethereum based blockchain. In *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)* (IEEE), 1–5

Quail, C. and Larabie, C. (2010). Net neutrality: Media discourses and public perception. *Global Media Journal* 3, 31

Quintais, J. (2020). The new copyright in the digital single market directive: a critical look. *European Intellectual Property Review*

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

Ramachandran, M., Chowdhury, N., Third, A., Domingue, J., Quick, K., and Bachler, M. (2020). Towards complete decentralised verification of data with confidentiality: Different ways to connect solid pods and blockchain. In *Companion Proceedings of the Web Conference 2020*. 645–649

Raman, A., Joglekar, S., Cristofaro, E. D., Sastry, N., and Tyson, G. (2019). Challenges in the decentralised web: The mastodon case. In *Proceedings of the Internet Measurement Conference*. 217–229

Rushby, J. M. (1981). Design and verification of secure systems. *ACM SIGOPS Operating Systems Review* 15, 12–21

Sabt, M., Achemlal, M., and Bouabdallah, A. (2015). Trusted execution environment: What it is, and what it is not. In *2015 IEEE TrustCom/BigDataSE/ISPA*. 57–64

Sandhu, R. S. and Samarati, P. (1994). Access control: principle and practice. *IEEE communications magazine* 32, 40–48

Terry Bahill, A. and Henderson, S. J. (2005). Requirements development, verification, and validation exhibited in famous failures. *Systems engineering* 8, 1–14

Toninelli, A., Montanari, R., Kagal, L., and Lassila, O. (2006). A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In *International semantic web conference* (Springer), 473–486

Tran, H., Hitchens, M., Varadharajan, V., and Watters, P. (2005). A trust based access control framework for P2P file-sharing systems. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences* (IEEE), 302c–302c

Xiao, Y., Zhang, N., Li, J., Lou, W., and Hou, Y. T. (2020). Privacyguard: Enforcing private data usage control with blockchain and attested off-chain contract execution. In *Computer Security – ESORICS 2020*, eds. L. Chen, N. Li, K. Liang, and S. Schneider. 610–629

Xu, X., Pautasso, C., Zhu, L., Gramoli, V., Ponomarev, A., Tran, A. B., et al. (2016). The blockchain as a software connector. In *WICSA* (IEEE Computer Society), 182–191

Xu, X., Weber, I., and Staples, M. (2019). *Architecture for Blockchain Applications* (Springer)

Zhao, C., Saifuding, D., Tian, H., Zhang, Y., and Xing, C. (2016). On the performance of intel sgx. In *2016 13Th web information systems and applications conference (WISA)* (IEEE), 184–187

Zhaofeng, M., Lingyun, W., Xiaochang, W., Zhen, W., and Weizhe, Z. (2020). Blockchain-enabled decentralized trust management and secure usage control of IoT big data. *IEEE Internet of Things Journal* 7, 4000–4015

Zheng, W., Wu, Y., Wu, X., Feng, C., Sui, Y., Luo, X., et al. (2021). A survey of intel sgx and its applications. *Frontiers of Computer Science* 15, 1–15

Pre-print copy of the manuscript published in **journal**
identified by DOI: 10.3389/fbloc.2023.1141909

This document is a pre-print copy of the manuscript
(Basile et al. 2023)
published in **journal**.

The final version of the paper is identified by DOI: 10.3389/fbloc.2023.1141909

References

Basile, Davide, Claudio Di Ciccio, Valerio Goretti, and Sabrina Kirrane (2023). "Blockchain based resource governance for decentralized web environments". In: *Frontiers in Blockchain* 6, p. 1141909. ISSN: 2624-7852. DOI: 10.3389/fbloc.2023.1141909.

BibTeX

```
@Article{         Basile.etal/FBloc2023:BlockchainResourceGovernanceDecentralizedWeb,
  author        = {Basile, Davide and Di Ciccio, Claudio and Goretti, Valerio
                   and Kirrane, Sabrina},
  journal       = {Frontiers in Blockchain},
  title         = {Blockchain based resource governance for decentralized web
                   environments},
  year          = {2023},
  issn          = {2624-7852},
  pages         = {1141909},
  volume        = {6},
  doi           = {10.3389/fbloc.2023.1141909},
  keywords      = {Decentralization; Usage control; Governance; Blockchain;
                   Trusted Execution Environment (TEE)}
}
```