

Blockchain based Resource Governance for Decentralized Web Environments

Davide Basile¹, Claudio Di Ciccio¹, Valerio Goretti¹ and Sabrina Kirrane²

¹*Sapienza University of Rome, Italy*

²*Vienna University of Economics and Business, Austria*

Correspondence*:

Valerio Goretti

valerio.goretti@uniroma1.it

2 ABSTRACT

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

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

1 INTRODUCTION

17 Since its development, the internet has steadily evolved into a ubiquitous ecosystem that is seen by many
18 as a public utility (Quail and Larabie, 2010). The development of centralized web-based platforms on top
19 of the internet has undoubtedly brought benefits from both an economic and a social perspective. However,
20 the web as we know it today, is dominated by a small number of stakeholders that have a disproportionate
21 influence on the content that the public can produce and consume. The scale of the phenomenon has
22 brought about the need for legal initiatives aimed at safeguarding content producer rights (Quintais, 2020).
23 In parallel, technical decentralization initiatives such as Solid¹, Digi.me², and ActivityPub³ aim to give
24 data owners more control over their data, while at the same time providing small companies as well as
25 individuals with access to data, which is usually monopolized by centralized platform providers, thus
26 stimulating innovation. To this end, the Solid community are developing tools, best practices, and web
27 standards that facilitate ease of data integration and support the development of decentralized social
28 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

29 that enable individuals to download their data from centralized platforms such that they can store it in an
30 encrypted personal data store and leverage a variety of applications that can process this data locally on
31 the data owners device. These client-side applications are developed by innovative app developers who
32 use the Digi.me software development kit to communicate with the encrypted personal data stores directly.
33 Following the same principles, ActivityPub is a decentralized social networking protocol, published by the
34 W3C Social Web Working Group that offers a client-server application programming interface (API) for
35 adding, modifying, and removing material as well as a federated server-server API for sending notifications
36 and subscribing to content. Social networks implementing ActivityPub can be easily integrated with each
37 other in order to form a larger ecosystem, commonly referred to as the Fediverse⁴. Some of the most
38 popular Fediverse initiatives include Mastodon⁵, PeerTube⁶, and PixelFed⁷.

39 In order to better cater for use case scenarios that involve data sharing across various distributed data
40 stores underpinning decentralized applications, there is a need for tools and technologies that are not only
41 capable of working with distributed data but are also able to manage data resources that come with a variety
42 of usage terms and conditions specified by data producers. However, the vast majority of decentralized web
43 initiatives, which aim to provide users with a greater degree of control over personal resources, manage
44 data access via simple access control mechanisms (Ouaddah et al., 2016; Toninelli et al., 2006; Tran et al.,
45 2005) that are not able to verify that usage conditions are adhered to after access has been granted (Akaichi
46 and Kirrane, 2022b). For example, access control rules can determine if users can retrieve data or not.
47 However, they cannot express conditions on the type of application that can process them, the geographical
48 area in which they can be treated, when the access grant would expire, or the number of times they can be
49 processed.

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

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

62 In terms of contributions, we extend the state of the art by: (i) proposing a generic resource governance
63 conceptual framework; (ii) demonstrating how blockchain technologies and trusted execution environments
64 can together be used to manage resource usage; and (iii) assessing the effectiveness of the proposed
65 framework via concrete quantitative and qualitative evaluation metrics derived from our data market
66 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.

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

2 BACKGROUND AND RELATED WORK

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

2.1 Background

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

2.1.1 Data Access and Usage Control

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

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

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

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

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

116 2.1.2 Trusted Execution Environments

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

146 Our work resorts to trusted execution environments to keep control of resources' utilization by enforcing
147 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.

148 2.1.3 Decentralized Applications and Blockchain Oracles

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

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

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

176 2.2 Related work

177 Several works strive to provide more control and transparency with respect to personal data processing
178 by leveraging blockchain distributed application platforms (Xu et al., 2019). For instance, Ayode et al.
179 (2018) defines an access control mechanism for IoT devices that stores a hash of the data in a blockchain
180 infrastructure and maintains the raw information in a secure storage platform using a TEE. In the proposed
181 framework, a blockchain based ledger is used in order to develop an audit trail of data access that provides
182 more transparency with respect to data processing. Xiao et al. (2020) propose a system, called PrivacyGuard,
183 which gives data owners control over personal data access and usage in a data market scenario.

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

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

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

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

3 MOTIVATING SCENARIO AND REQUIREMENTS

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

3.1 Motivating Scenario

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

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

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

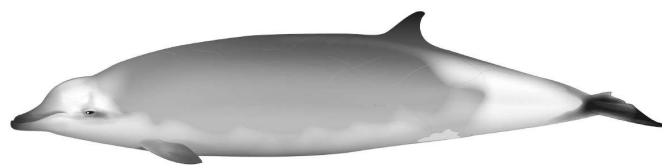


Figure 1. A photographic representation of a Mesoplodon eueu (Carroll et al., 2021). Image used under the Attribution 4.0 International (CC BY 4.0) license (<https://creativecommons.org/licenses/by/4.0/>). Cropped from original.

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

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

251 **3.2 Requirements**

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

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

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

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

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

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

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

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

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

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

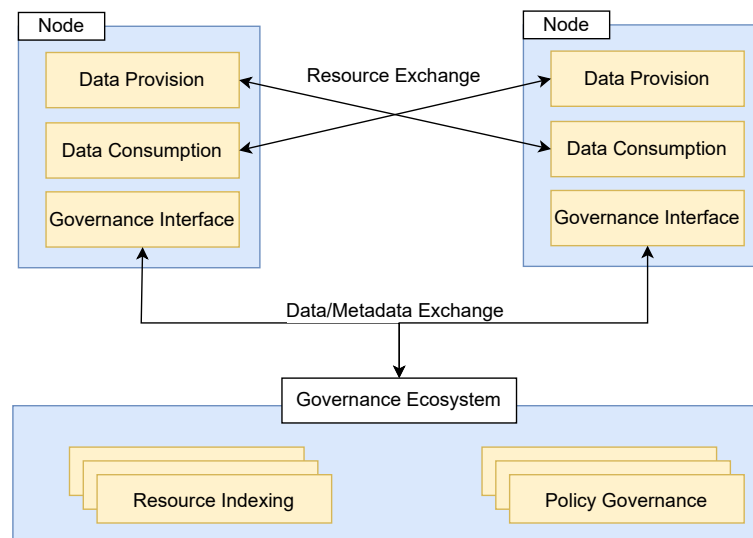


Figure 2. High-level overview of the proposed conceptual resource governance (ReGov) framework.

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

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

4 CONCEPTUAL RESOURCE GOVERNANCE FRAMEWORK

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

312 4.1 ReGov Framework Entities

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

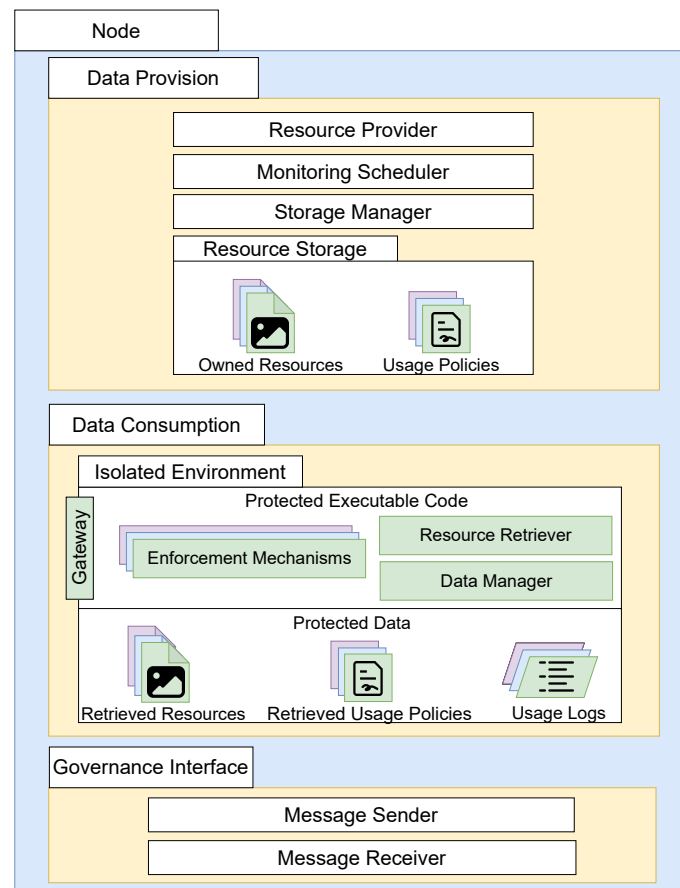


Figure 3. Content of the data provision, data consumption and governance interface components.

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

323 4.1.1 Components of a Node

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

326 **Data provision.** The Data Provision component encapsulates the functionality that enable node owners
 327 to manage the sharing of their resources with other nodes in the decentralized environment. Users can
 328 interact with the Storage Manager to manually upload their data to the Resource Storage that is
 329 encapsulated within the Data Provision component. The upload operation also facilitates the definition
 330 of usage rules that are collected in usage policies associated with resources. Usage policies are represented
 331 in a machine-readable format (e.g., SPECIAL⁹ and LUCON¹⁰ policy languages) and stored in the Data
 332 Provision component alongside the resources. Additionally, when a new resource is uploaded, the
 333 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.

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

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

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

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

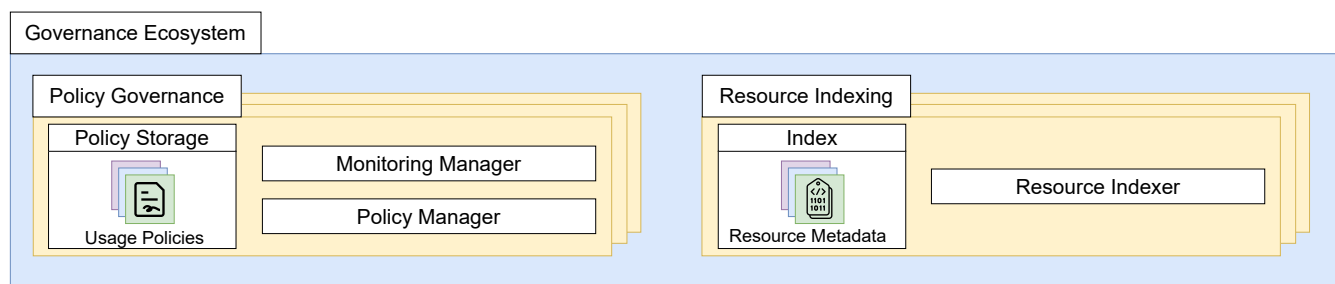


Figure 4. Content of policy governance and resource indexing components inside the governance ecosystem

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

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

389 4.1.2 Components of the Governance Ecosystem

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

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

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

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

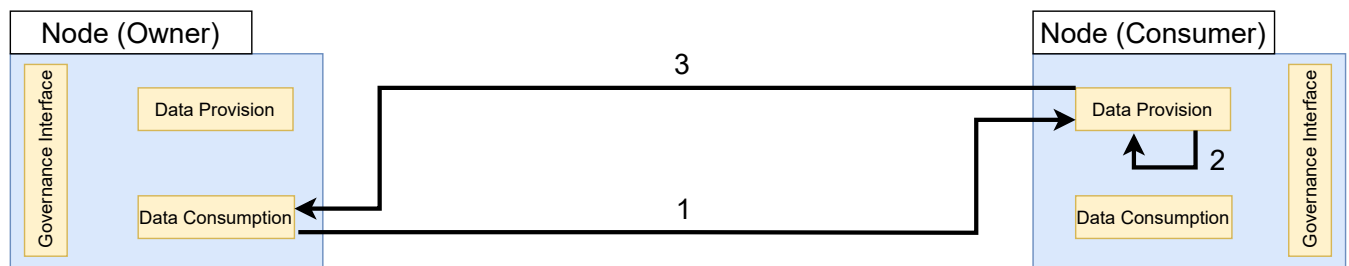


Figure 5. Visualization of the ReGov framework data retrieval process.

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

413 4.2 Predominant ReGov Framework Operations

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

418 4.2.1 Data Retrieval

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

437 4.2.2 Monitoring

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

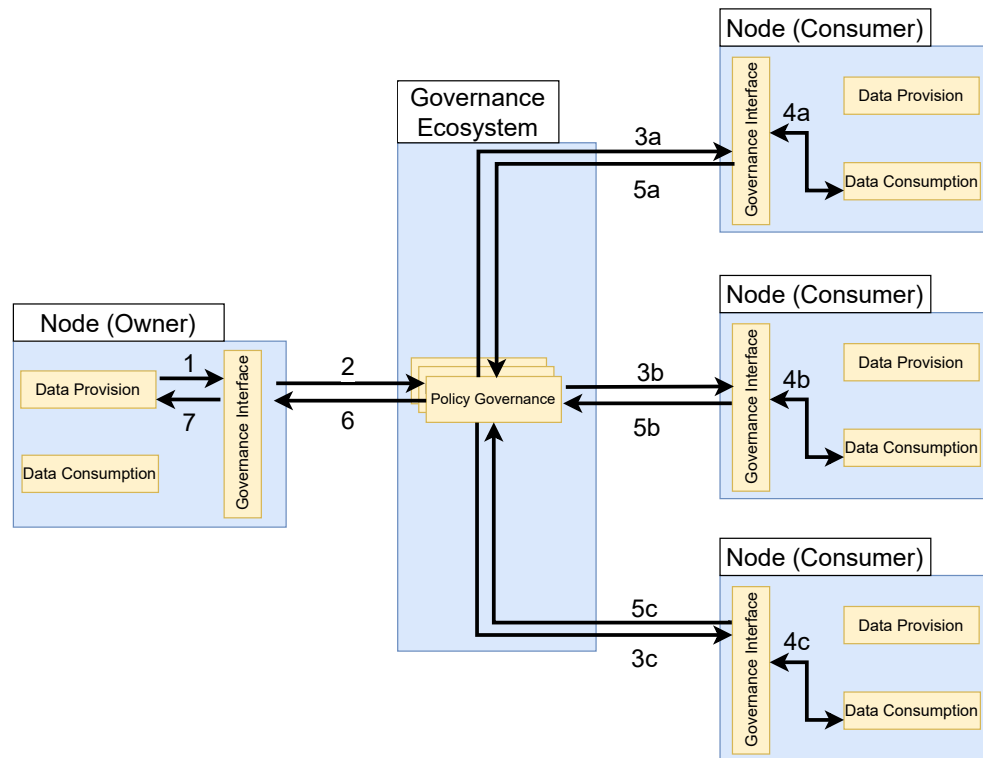


Figure 6. Visualization of the ReGov framework data monitoring routine.

443 Governance component forwards the request to provide evidence of utilization to each consumer Node that
 444 has a copy of the resource (**3a, 3b, 3c**). In the depicted monitoring routine, we assume the resource whose
 445 usage must be monitored is held by three consumer nodes. In each of these nodes, the monitoring request
 446 is received by the Message Receiver of the Governance Interface that forwards, in turn, the request
 447 to the Data Manager running in the Isolated Environment inside the Data Consumption component
 448 (**4a, 4b, 4c**). The latter retrieves the usage log from the protected data storage and employs the Message
 449 Sender of the Governance Interface to forward the information to the Governance Ecosystem,
 450 which in turn ensures that all the consumer node responses are collected (**5a, 5b, 5c**). Finally, the evidence
 451 are returned to the Message Receiver (**6**) of the initiator Node, which delivers the information to the
 452 Monitoring Scheduler (**7**).

5 BLOCKCHAIN AND TRUSTED EXECUTION ENVIRONMENT INSTANTIATION

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

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

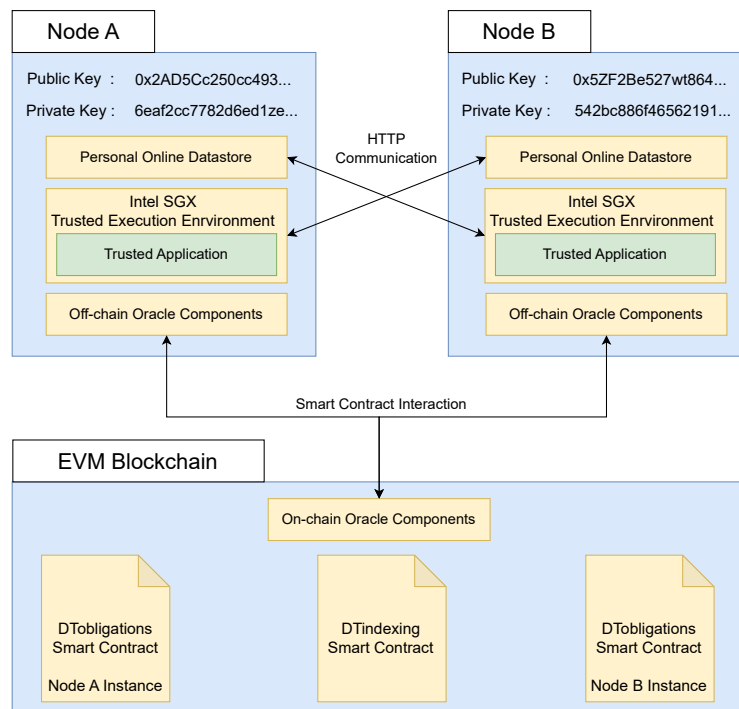


Figure 7. High-level architectural overview of our ReGov framework instantiation.

463 a Trusted Application containing the logic for Data Consumption. The Governance Ecosystem is
 464 realized by developing blockchain smart contracts that store information and execute distributed procedures.
 465 Our implementation involves an EVM Blockchain¹¹ (i.e., a blockchain based on the Ethereum Virtual
 466 Machine) which hosts the DTindexing and DTobligations smart contracts. They fulfill the functions of
 467 the Resource Indexing and Policy Governance components of the general framework, respectively.
 468 DTindexing is characterized by a unique instance managing the resource metadata of the decentralized
 469 environment. Instead, DTobligations is designed to be deployed multiple times. Therefore, each Node is
 470 associated with a specific instance of this smart contract that stores the rules for its resources. The
 471 tasks performed by the Governance Interface are executed by blockchain oracles that provide a
 472 communication channel between the blockchain and the nodes of the decentralized environment. Oracles
 473 consist of On-Chain components, running in the EVM Blockchain, and Off-Chain components, operating
 474 within each Node. We built the resource retrieval process between nodes using the HTTP communication
 475 standard. By interacting with smart contracts, nodes exchange metadata necessary for resource indexing
 476 and monitoring procedures.

477 Our implementation employs the asymmetric encryption methodology that underlies the EVM
 478 Blockchain, in order to provide an authentication mechanism for the environment's nodes. Each Node
 479 is uniquely related to a public and private key pair that is used to sign authenticable data requests and
 480 transactions that transmit information to the blockchain and execute smart contract functions. A private key
 481 is a 256-bit number generated through a secure random number generator. The corresponding public key is
 482 derived from the private key through the Elliptic Curve Digital Signature Algorithm (Johnson et al., 2001).
 483 The public key is connected to a unique account address on the EVM Blockchain derived as a 160-bit
 484 segment of the hash digest of the public key. In our setting, Nodes store their private key in an encrypted
 485 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.

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

490 5.1 Usage Policy Instantiation

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

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

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

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

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

508 5.2 Personal Online Data Stores for Data Provision

509 We develop the Personal Online Datastore prototype using the Python language. Python's support
510 for the Web3.py library¹² enables the creation of communication protocols with the blockchain platform
511 acting as the Governance Ecosystem of the decentralized environment. Our implementation also includes
512 a graphical user interface developed with the Tkinter library¹³. As shown in Fig. 8, our Personal Online
513 Datastore implementation is composed of three main parts: the Application, the Web Service and
514 the Resource Storage. The app module contains the executable code implementing the graphical user
515 interface.

516 5.2.1 Resource Storage

517 The resource storage contains the resources of the Personal Online Datastore. The storage location
518 is characterized by two meta-files named DTconfig.json and DTobligations.json. They contain
519 descriptive and confidential information about the Personal Online Datastore and its resources.
520 DTconfig.json includes various attributes of a Personal Online Datastore, such as its unique
521 identifier, its node's public and private keys, the web reference to access data, and a list of the initialized
522 resources. DTobligations.json holds rules that apply to the resources of the storage. The user can
523 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.

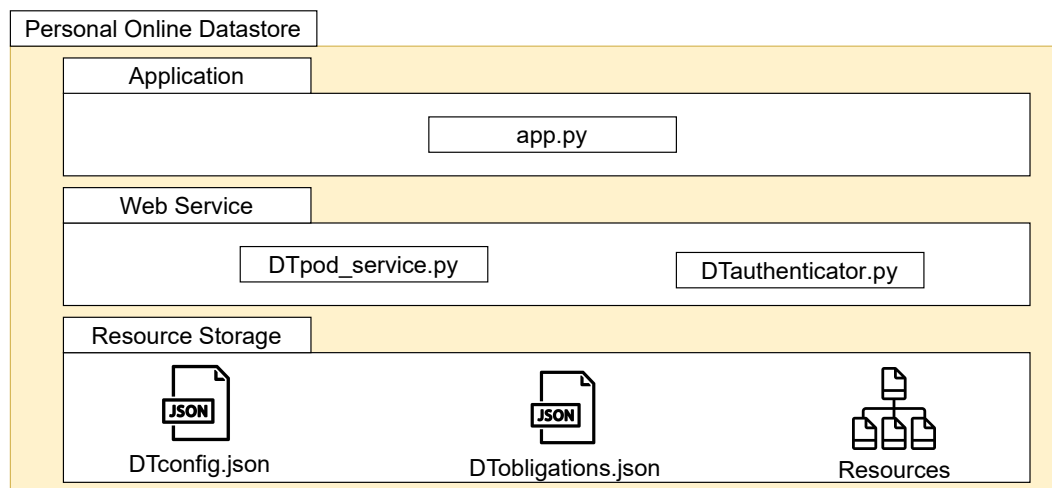


Figure 8. Schematization of the personal online datastore implementation.

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

528 5.2.2 Web Service

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

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

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

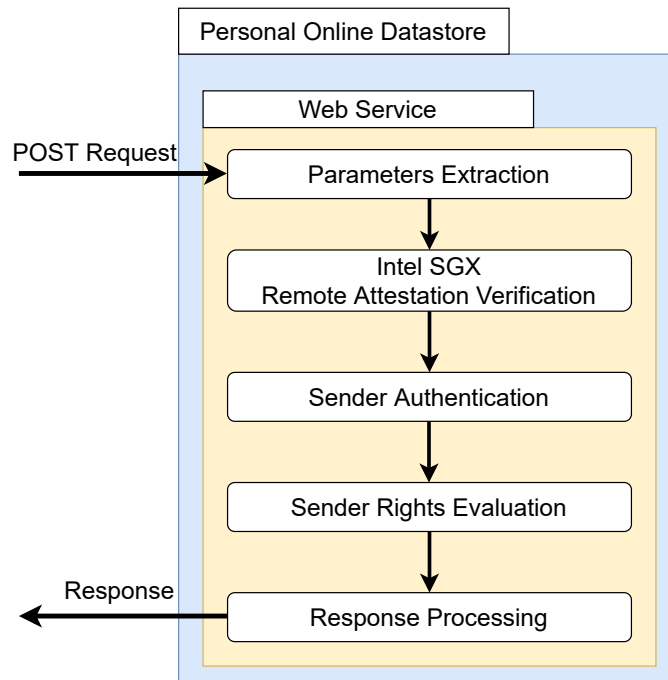


Figure 9. Main stages of the ReGov data provision instantiation process.

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

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

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

570 5.3 Trusted Execution Environment for Data Consumption

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

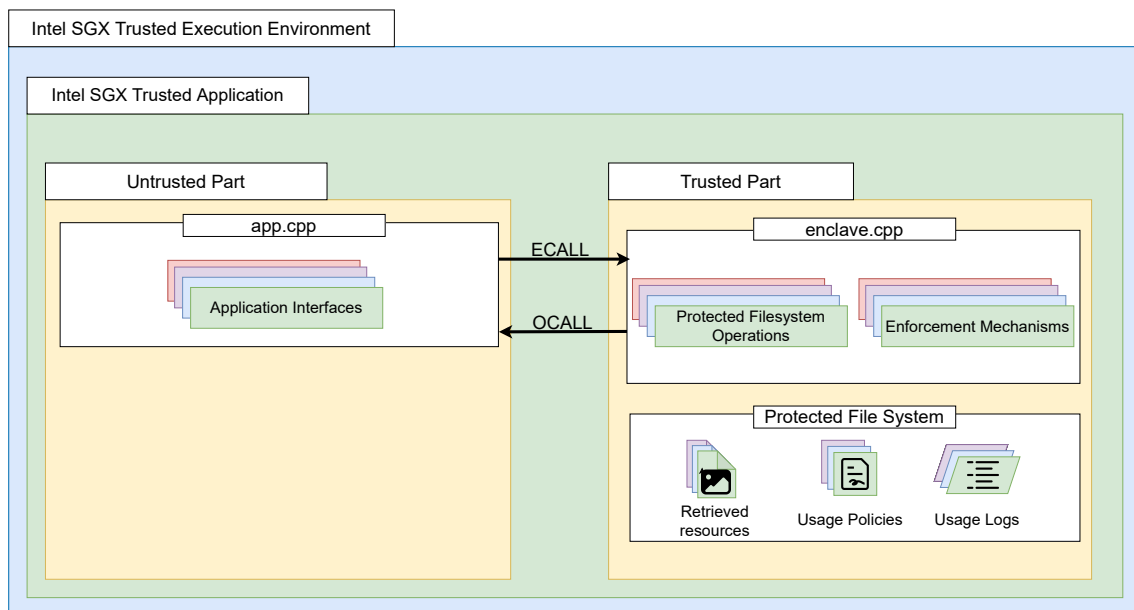


Figure 10. Schematization of our trusted application composed of both trusted and untrusted elements.

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

583 5.3.1 Data Protection

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

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

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

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

622 5.3.2 Implementation of the Enforcement Mechanisms

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

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

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

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

645 **Geographical rule enforcement.** The `enforce_geographical` function is invoked and passed the policy
646 for the requested resource. The `get_geo_location` function (`0call`) is then used to retrieve the geographic
647 location of the device from which the resource is being accessed. In the end, the geographic data set by
648 the user and the current location are compared. If the position is correct, a positive result is returned to
649 the `access_protected_resource` function, otherwise access is denied. Referring to the scenario, the
650 Trusted Application uses Alice's location to check if it meets the location stipulated by Bob in his
651 usage policy.

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

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

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

682 5.4 Blockchain as a Governance Ecosystem

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

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

690 We implemented the smart contracts using the Solidity programming language¹⁴. The smart contracts
691 have been deployed in a local environment powered by the Ganache tool¹⁵ which enables the execution
692 of a local blockchain replicating the Ethereum protocol and supporting the generation of transactions
693 for testing purposes. In the following, we present the implementation details regarding the DTindexing
694 and DTobligations smart contracts that fulfill the functionality of the Resource Indexing and Policy
695 Governance components respectively.

696 5.4.1 DTindexing Smart Contract

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

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

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

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

708 The Pod and Resource structs are stored in the podList and resourceList array variables, respectively.
709 The contract includes several methods for interacting with online datastores and resources, including
710 the ability to register new ones, deactivate existing ones, and to search for them based on various
711 criteria. For example, the registerPod method allows nodes to initialize new personal online datastores
712 in the network. It takes as input a web reference for the online datastore service and the public
713 key of the owner Node. The function creates a new Pod struct and stores it in the podList. It
714 also deploys a DTobligations smart contract (discussed next in detail), as every Personal Online
715 Datastore is related to one of these contracts. Finally, the function emits a NewPod event containing
716 the identifier and the address of the DTobligations smart contract for the new online datastore.
717 In our running example, Bob's node invokes this function to initialize his new Personal Online
718 Datastore providing the web reference <https://BobNode.com/> among the arguments. The function,
719 in turn, generates a new Pod struct. The registerResource method works similarly, generating a
720 new Resource object and storing it in the resourceList state variable. In this case, Bob's Personal
721 Online Datastore employs this function to initialize the 'Mesoplodon.jpg' image providing metadata
722 such as the <https://BobNode.com/images/Mesoplodon.jpg> url. The deactivateResource and
723 deactivatePod methods ensure that personal online datastores and resources are no longer accessible.
724 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.

Table 2. Class diagram of the DTindexing smart contract.

DTindexing
<pre> private podsCounter: int private resourceCounter: int private dtSubscription: int private podList: Pod[] private resourceList: Resource[] </pre>
<pre> private searchByType(tp: PodType): Pod[] <<event>> NewPod(idPod: int, obligationAddress: 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>> </pre>

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

731 5.4.2 DTobligations Smart Contract

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

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

Table 3. Class diagram of the DTobligations smart contract.

DTobligations <<extends >> Ownable
<pre> 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>> </pre>

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

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

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

6 EVALUATION

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

6.1 Requirement Verification

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

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

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

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

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

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

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

820 6.1.2 (R2) Policy compliance must be monitored via the entities of a governance ecosystem

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

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

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

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

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

852 6.2 Architecture Discussion

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

856 6.2.1 Security

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

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

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

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

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

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

897 6.2.2 Privacy

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

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

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

920 6.2.3 Affordability

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

927 The deployment cost of DTindexing is 3,255,000 Gas units. The registerPod method is the most
928 expensive DTindexing's function (2,082,494 Gas units) as it involves the deployment of a new contract
929 instance, too. The Gas consumption of registerResource turns out to be significantly lower, requiring
930 143,004 Gas units. The least expensive function of the smart contract is deactivateResource with an
931 expenditure of 21,465 Gas units.

932 DTobligations is deployed during the registration of a new personal online datastore at the cost of
933 2,057,988 Gas units. DTobligations offers methods and functions to modify the obligation rules related
934 to the resources contained in personal online datastore. Among the functions for adding rules, the most
935 expensive one is addAccessCounterObligation with a value of 138,768 Gas units. However, the adding
936 of a domain restriction through addDefaultDomainObligation costs significantly less with 44,219 Gas
937 units per invocation. Methods for rule deactivation determine a lower expense than the previous ones. The

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

DTobligations		DTindexing	
Function	Cost	Function	Cost
deployment	2,057,988	deployment	3,255,000
addDefaultAccessCounterObligation(...)	62,627	registerPod(...)	2,082,494
addDefaultTemporalObligation(...)	62,638	registerResource(...)	143,004
addDefaultDomainObligation(...)	44,219	deactivateResource(...)	21,465
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		

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

940 As expected, operations involving new smart contract deployments are the most expensive ones. However,
 941 these costs are associated with one-time operations performed at setup time (at the bootstrapping of
 942 the platform, or every time a new pod is registered). On the other hand, functions intended for more
 943 frequent invocations (e.g., to monitor compliance or update rules) are characterized by significantly lower
 944 costs. Costs in fiat money are subject to high variability, as they depend on multiple factors including the
 945 network capacity utilization, the price in cryptocurrency per Gas unit, and the market exchange rate of the
 946 cryptocurrency. Also, these values change depending on the EVM blockchain in use (e.g., Ethereum¹⁶,
 947 Avalanche¹⁷, Polygon¹⁸, and more). At the time of writing, we empirically found variations of four orders
 948 of magnitude¹⁹. However, we remark that our implementation costs align with ERC721 implementations²⁰.
 949 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.

950 neighboring area of personal information indexing, amounts to 2,443,978 Gas units²². The market scenario
951 can support the structural expenses associated with the proposed implementation and provides an incentive
952 system that allows users to earn money by sharing their data. However, cost reduction practices are
953 necessary to increase usability. These include design improvements to the implementation's architecture as
954 well as the adoption of side-chains and layer-2 networks.

7 CONCLUSION

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

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

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

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

981 Acknowledgments.

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

²² <https://etherscan.io/tx/0xff3ee18523c9ec20e62d31d3d3ce3e8bf25f5ffcd4c32cd43ed0a786cc8640>. Accessed: Thursday 11th May, 2023.

REFERENCES

- 988 Akaichi, I. and Kirrane, S. (2022a). A semantic policy language for usage control. In *SEMANTiCS (Posters*
989 *& Demos)* (CEUR-WS.org), 10:1–10:5
- 990 Akaichi, I. and Kirrane, S. (2022b). Usage control specification, enforcement, and robustness: A survey.
991 *arXiv preprint arXiv:2203.04800*
- 992 Al-Breiki, H., Rehman, M. H. U., Salah, K., and Svetinovic, D. (2020a). Trustworthy blockchain oracles:
993 Review, comparison, and open research challenges. *IEEE Access* 8, 85675–85685
- 994 Al-Breiki, H., Rehman, M. H. U., Salah, K., and Svetinovic, D. (2020b). Trustworthy blockchain oracles:
995 Review, comparison, and open research challenges. *IEEE Access* 8, 85675–85685
- 996 Alabdulwahhab, F. A. (2018). Web 3.0: The decentralized web blockchain networks and protocol innovation.
997 In *2018 1st International Conference on Computer Applications & Information Security (ICCAIS)*. 1–4.
998 doi:10.1109/CAIS.2018.8441990
- 999 Ayoade, G., Karande, V., Khan, L., and Hamlen, K. (2018). Decentralized IoT data management using
1000 blockchain and trusted execution environment. In *2018 IEEE International Conference on Information*
1001 *Reuse and Integration (IRI)*. 15–22. doi:10.1109/IRI.2018.00011
- 1002 Bai, G., Yan, L., Gu, L., Guo, Y., and Chen, X. (2014). Context-aware usage control for web of things.
1003 *Security and Communication Networks* 7, 2696–2712
- 1004 Basile, D., Goretti, V., Di Ciccio, C., and Kirrane, S. (2021). Enhancing blockchain-based processes with
1005 decentralized oracles. In *BPM (Blockchain and RPA Forum)*. 102–118
- 1006 Becker, H., Vu, H., Katzenbach, A., Braun, C. H., and Käfer, T. (2021). Monetising resources on a solid
1007 pod using blockchain transactions. In *The Semantic Web: ESWC 2021 Satellite Events*. 49–53
- 1008 Bonatti, P. A., Kirrane, S., Petrova, I. M., and Sauro, L. (2020). Machine understandable policies and
1009 GDPR compliance checking. *KI-Künstliche Intelligenz* 34, 303–315
- 1010 Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform. *white*
1011 *paper* 3, 2–1
- 1012 Cai, T., Yang, Z., Chen, W., Zheng, Z., and Yu, Y. (2020). A blockchain-assisted trust access authentication
1013 system for solid. *IEEE Access*
- 1014 Carroll, E. L., McGowen, M. R., McCarthy, M. L., Marx, F. G., Aguilar, N., Dalebout, M. L., et al. (2021).
1015 Speciation in the deep: genomics and morphology reveal a new species of beaked whale mesoplodon
1016 eueu. *Proceedings of the Royal Society B* 288, 20211213
- 1017 Costan, V. and Devadas, S. (2016). Intel sgx explained. *Cryptology ePrint Archive*
- 1018 Esteves, B. and Rodríguez-Doncel, V. (2022). Analysis of ontologies and policy languages to represent
1019 information flows in GDPR. *Semantic Web* , 1–35
- 1020 Ferrag, M. A. and Shu, L. (2021). The performance evaluation of blockchain-based security and privacy
1021 systems for the internet of things: A tutorial. *IEEE Internet of Things Journal* 8, 17236–17260.
1022 doi:10.1109/JIOT.2021.3078072
- 1023 Grünbacher, A. (2003). POSIX access control lists on linux. In *Proceedings of the FREENIX Track: 2003*
1024 *USENIX Annual Technical Conference*. 259–272
- 1025 Havur, G., Vander Sande, M., and Kirrane, S. (2020). Greater control and transparency in personal data
1026 processing. In *International Conference on Information Systems Security and Privacy (ICSSP)*. 655–662.
1027 doi:10.5220/0009143206550662
- 1028 Hilty, M., Pretschner, A., Basin, D., Schaefer, C., and Walter, T. (2007). A policy language for distributed
1029 usage control. In *European Symposium on Research in Computer Security* (Springer), 531–546
- 1030 Jauernig, P., Sadeghi, A.-R., and Stapf, E. (2020). Trusted execution environments: properties, applications,
1031 and challenges. *IEEE Security & Privacy* 18, 56–60

- 1032 Johnson, D., Menezes, A., and Vanstone, S. (2001). The elliptic curve digital signature algorithm (ecdsa).
1033 *International journal of information security* 1, 36–63
- 1034 Khan, M. Y., Zuhairi, M. F., Syed, T. A., Alghamdi, T. G., and Marmolejo-Saucedo, J. A. (2020). An
1035 extended access control model for permissioned blockchain frameworks. *Wirel. Networks* 26, 4943–4954
- 1036 Kirrane, S. and Di Ciccio, C. (2020). BlockConfess: Towards an architecture for blockchain constraints
1037 and forensics. In *AICChain@Blockchain* (IEEE), 539–544. doi:10.1109/Blockchain50366.2020.00078
- 1038 Koshutanski, H. and Massacci, F. (2003). An access control framework for business processes for web
1039 services. In *Proceedings of the 2003 ACM workshop on XML security*. 15–24
- 1040 Lazouski, A., Martinelli, F., and Mori, P. (2010). Usage control in computer security: A survey. *Computer*
1041 *Science Review* 4, 81–99
- 1042 Liang, X., Shetty, S., Zhao, J., Bowden, D., Li, D., and Liu, J. (2017). Towards decentralized
1043 accountability and self-sovereignty in healthcare systems. In *International conference on information*
1044 *and communications security* (Springer), 387–398
- 1045 Lind, J., Eyal, I., Kelbert, F., Naor, O., Pietzuch, P., and Sirer, E. G. (2017). Teechain: Scalable blockchain
1046 payments using trusted execution environments. *arXiv preprint arXiv:1707.05454*
- 1047 Mammadzada, K., Iqbal, M., Milani, F., García-Bañuelos, L., and Matulevicius, R. (2020). Blockchain
1048 oracles: A framework for blockchain-based applications. In *BPM (Blockchain and RPA Forum)*
1049 (Springer), 19–34
- 1050 Marangone, E., Di Ciccio, C., and Weber, I. (2022). Fine-grained data access control for collaborative
1051 process execution on blockchain. *arXiv preprint arXiv:2207.08484*
- 1052 McGillion, B., Dettenborn, T., Nyman, T., and Asokan, N. (2015). Open-tee—an open virtual trusted
1053 execution environment. In *2015 IEEE Trustcom/BigDataSE/ISPA* (IEEE), vol. 1, 400–407
- 1054 Mohanty, D. (2018). Ethereum for architects and developers. *Apress Media LLC, California* , 14–15
- 1055 Mühlberger, R., Bachhofner, S., Ferrer, E. C., Di Ciccio, C., Weber, I., Wöhrer, M., et al. (2020).
1056 Foundational oracle patterns: Connecting blockchain to the off-chain world. In *BPM (Blockchain and*
1057 *RPA Forum)* (Springer), 35–51
- 1058 Neisse, R., Pretschner, A., and Di Giacomo, V. (2011). A trustworthy usage control enforcement
1059 framework. In *2011 Sixth International Conference on Availability, Reliability and Security*. 230–235.
1060 doi:10.1109/ARES.2011.40
- 1061 Ouaddah, A., Abou Elkalam, A., and Ait Ouahman, A. (2016). Fairaccess: a new blockchain-based access
1062 control framework for the internet of things. *Security and communication networks* 9, 5943–5964
- 1063 Pan, J., Paul, S., and Jain, R. (2011). A survey of the research on future internet architectures. *IEEE*
1064 *Communications Magazine* 49, 26–36
- 1065 Park, J. and Sandhu, R. (2004). The uconabc usage control model. *ACM transactions on information and*
1066 *system security (TISSEC)* 7, 128–174
- 1067 Pasdar, A., Lee, Y. C., and Dong, Z. (2022). Connect API with blockchain: A survey on blockchain oracle
1068 implementation. *ACM Comput. Surv.* doi:10.1145/3567582
- 1069 Patel, S., Sahoo, A., Mohanta, B. K., Panda, S. S., and Jena, D. (2019). Dauth: A decentralized web
1070 authentication system using ethereum based blockchain. In *2019 International Conference on Vision*
1071 *Towards Emerging Trends in Communication and Networking (ViTECoN)* (IEEE), 1–5
- 1072 Quail, C. and Larabie, C. (2010). Net neutrality: Media discourses and public perception. *Global Media*
1073 *Journal* 3, 31
- 1074 Quintais, J. (2020). The new copyright in the digital single market directive: a critical look. *European*
1075 *Intellectual Property Review*

- 1076 Ramachandran, M., Chowdhury, N., Third, A., Domingue, J., Quick, K., and Bachler, M. (2020). Towards
1077 complete decentralised verification of data with confidentiality: Different ways to connect solid pods and
1078 blockchain. In *Companion Proceedings of the Web Conference 2020*. 645–649
- 1079 Raman, A., Joglekar, S., Cristofaro, E. D., Sastry, N., and Tyson, G. (2019). Challenges in the decentralised
1080 web: The mastodon case. In *Proceedings of the Internet Measurement Conference*. 217–229
- 1081 Rushby, J. M. (1981). Design and verification of secure systems. *ACM SIGOPS Operating Systems Review*
1082 15, 12–21
- 1083 Sabt, M., Achemlal, M., and Bouabdallah, A. (2015). Trusted execution environment: What it is, and what
1084 it is not. In *2015 IEEE TrustCom/BigDataSE/ISPA*. 57–64
- 1085 Sandhu, R. S. and Samarati, P. (1994). Access control: principle and practice. *IEEE communications*
1086 *magazine* 32, 40–48
- 1087 Terry Bahill, A. and Henderson, S. J. (2005). Requirements development, verification, and validation
1088 exhibited in famous failures. *Systems engineering* 8, 1–14
- 1089 Toninelli, A., Montanari, R., Kagal, L., and Lassila, O. (2006). A semantic context-aware access control
1090 framework for secure collaborations in pervasive computing environments. In *International semantic*
1091 *web conference* (Springer), 473–486
- 1092 Tran, H., Hitchens, M., Varadharajan, V., and Watters, P. (2005). A trust based access control framework
1093 for P2P file-sharing systems. In *Proceedings of the 38th Annual Hawaii International Conference on*
1094 *System Sciences* (IEEE), 302c–302c
- 1095 Xiao, Y., Zhang, N., Li, J., Lou, W., and Hou, Y. T. (2020). Privacyguard: Enforcing private data usage
1096 control with blockchain and attested off-chain contract execution. In *Computer Security – ESORICS*
1097 *2020*, eds. L. Chen, N. Li, K. Liang, and S. Schneider. 610–629
- 1098 Xu, X., Pautasso, C., Zhu, L., Gramoli, V., Ponomarev, A., Tran, A. B., et al. (2016). The blockchain as a
1099 software connector. In *WICSA* (IEEE Computer Society), 182–191
- 1100 Xu, X., Weber, I., and Staples, M. (2019). *Architecture for Blockchain Applications* (Springer)
- 1101 Zhao, C., Saifuding, D., Tian, H., Zhang, Y., and Xing, C. (2016). On the performance of intel sgx. In
1102 *2016 13Th web information systems and applications conference (WISA)* (IEEE), 184–187
- 1103 Zhaofeng, M., Lingyun, W., Xiaochang, W., Zhen, W., and Weizhe, Z. (2020). Blockchain-enabled
1104 decentralized trust management and secure usage control of IoT big data. *IEEE Internet of Things*
1105 *Journal* 7, 4000–4015
- 1106 Zheng, W., Wu, Y., Wu, X., Feng, C., Sui, Y., Luo, X., et al. (2021). A survey of intel sgx and its
1107 applications. *Frontiers of Computer Science* 15, 1–15

This document is a pre-print copy of the manuscript
([Basile et al. 2023](#))
published by Frontiers.

The final version of the paper is identified by DOI: [10.3389/fbloc.2023.1141909](https://doi.org/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](https://doi.org/10.3389/fbloc.2023.1141909).

BibTeX

```
@Article{
  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)},
  publisher   = {Frontiers}
}
```