SoK: Data Sovereignty

Jens Ernstberger [†▼], Jan Lauinger [†], Fatima Elsheimy [‡], Liyi Zhou [§▼],
Sebastian Steinhorst [†], Ran Canetti [¶], Andrew Miller [‖▲], Arthur Gervais [**▼], Dawn Song [*▼]
[†]*Technical University of Munich, Germany*
[‡]*Yale University, United States*
[§]*Imperial College London, United Kingdom*
[¶]*Boston University, United States*
[‖]*University of Illinois at Urbana-Champaign, United States*
[**]*University College London, United Kingdom*
[*]*University of California, Berkeley, United States*
[▲]*IC3,* [▼]*Berkeley Center for Responsible, Decentralized Intelligence (RDI)*

***Abstract*—Society appears to be on the verge of recognizing the need for control over sensitive data in modern web applications. Recently, many systems claim to give control to individuals, promising the preeminent goal of *data sovereignty*. However, despite recent attention, research and industry efforts are fragmented and lack a holistic system overview. In this paper, we provide the first transecting systematization of data sovereignty by drawing from a dispersed body of knowledge. We clarify the field by identifying its three main areas: *(i)* decentralized identity, *(ii)* decentralized access control and *(iii)* policy-compliant decentralized computation. We find that literature lacks a cohesive set of formal definitions. Each area is considered in isolation, and priorities in industry and academia are not aligned due to a lack of clarity regarding user control. To solve this issue, we propose formal definitions for each subarea. By highlighting that data sovereignty transcends the domain of decentralized identity, we aim to guide future works to embrace a broader perspective on user control. In each section, we augment our definition with security and privacy properties, discuss the state of the art and proceed to identify open challenges. We conclude by highlighting synergies between areas, emphasizing the real-world benefit obtained by further developing data sovereign systems.**

1. Introduction

Administering and protecting user-affiliated data is an undeniable burden. In the physical world, the importance of ownership over confidential documents is well understood — secrecy, shredding, and limiting access have been common practice for decades. Unfortunately, privacy is less practiced in the digital domain, and users are often unaware of privacy violations [1]. In the current version of the Web (Web 2.0), user data is managed by centralized entities through a wide range of web platforms. Service providers generate, store and analyze large amounts of user data. In essence, they remain in control over the personal data of their users. Simultaneously, implementing regulations to protect user data is labor-intensive and expensive [2], diminishing utility for users *and* service providers. To improve administration and protection of user-affiliated data, recent developments aim

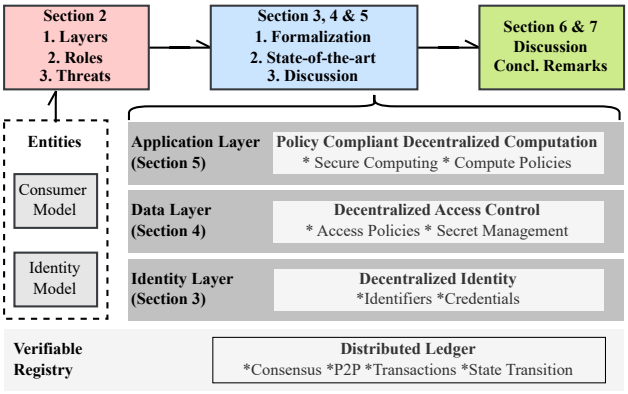

Figure 1: Overview of Data Sovereignty (§ 2). Throughout this work, we assume the existence of a verifiable registry. We set out to systematize decentralized identity (§ 3), decentralized access control (§ 4), and policy-compliant decentralized computation (§ 5). In each section, we give a formalization followed by insights & challenges.

to address two separate, yet intertwined issues — enabling sufficiently fine-grained control of users over the use of their data, which is pertinent (and lacking) even in a centralized setting, and decentralization, which guarantees the security and availability of the system without the need to trust any single entity, even when a significant fraction of the participants are misbehaving. These efforts contribute to the evolution of the Web (Web 3.0), where data is controlled through user-generated authority [3]. However, defining user control remains a challenging task, as the predominant emphasis in current research and development centers on *decentralized identity*. While a decentralized identity allows a user to disclose sensitive information through knowledge of a secret key, complementary solutions that enable control over how data is accessed and processed are still underdeveloped.

In this work, we further explore the space of data sovereignty beyond the currently discussed notion of decentralized identity. We set out to synthesize existing concepts, highlight areas of research that demand for greater attention, and proceed to lay out a general framework by building upon a vast number of existing works and

literature. We identify *decentralized access control* as an additional, complementary key control point for a user. Further, we observe that current technologies, and both decentralized identity and decentralized access control, are insufficient to provide users with control over how their data is *processed*. Therefore, we introduce *policy-compliant decentralized computation* as a contribution of independent interest with the aim to mitigate and control data non-rivalry. We are the first to propose formal definitions for the novel concepts of decentralized access control and policy-compliant decentralized computation, which we deem essential fragments of data sovereign systems.

The contributions made in this paper are fourfold:

- We disentangle the concepts of control over data and decentralization, providing a comprehensive understanding of their individual significance.
- We clarify the notion of decentralized identity, providing its first formalization by relying on upcoming community standards and state-of-the-art protocols.
- We introduce the first definition and formalization of decentralized access control and policy-compliant decentralized computation.
- We discuss the capabilities and limitations of each area, and the vastness of unexplored research possibilities, highlighting the potential for future advancements.

We follow up each formalization with a list of security & privacy properties that a protocol should possess. Further, we provide a Universal Composability (UC) analysis in the appendix, detailing the security guarantees of the proposed formalisms. We hope that our results will enable non-experts to better understand the field, while allowing domain specialists to identify relevant research questions.

Structure of the paper. The remainder of this paper is organized as follows (cf. Figure 1). Section 2 introduces the layers and entities that comprise the data sovereignty ecosystem. In Sections 3-5, we systematize *(i)* decentralized identity and propose the novel concepts of *(ii)* decentralized access control and *(iii)* policy-compliant decentralized computation. We begin each section by contrasting solutions for Web 2.0 and Web 3.0. Successively, we provide a generic and formal definition. We proceed to discuss the state-of-the-art, highlight recent achievements through concise insights, and uncover potential synergies and challenges by taking a birds-eye view on the overall ecosystem. Finally, in Section 6 and 7, we emphasize the interdependence between investigated sub-areas and conclude with recommendations for the Web 3.0 community.
Scope & Methodology. We aim to provide a holistic overview, presenting the most relevant lines of work, whilst highlighting the differences between a platform-centric Web 2.0 and a user-centric Web 3.0. Since our goal is to systematize a large body of work, instead of providing a complete survey, we focus on crucial aspects and key papers as a foundation for our systematization. Therein, we focus on works that discuss how a user can *(i)* retain unshared control over and *(ii)* audit any actions on its personal data. We consider distributed key management, distributed storage solutions and calculations for reputation and trust to be out of scope.

2. Overview - Data Sovereignty

Currently, Web 2.0 solutions remain fallible due to their reliance on regulatory frameworks and centralized trust [4], [5]. The overarching objective of data sovereign tools and frameworks is to allow users to control *how*, *when*, and *where* other peers in the network may utilize their personal data. The user is able to track and successively audit operations performed on its personal data to either detect or prevent privacy violations. Ultimately, fulfilling these goals is motivated by the desire for a decentralized society, solely governed by users [6]. To explicitly define data sovereignty, we develop a system model which introduces *layers* upon which current systems operate, *roles* that participants can take, and *threats* that are relevant in subsequent sections (cf. Figure 1).

2.1. Layers

Our system model consists of four layers (cf. Figure 1). The *(i)* Verifiable Registry acts as a trustless third party which ensures availability of public values and auditability. It ensures a consistent view of registered values, such as public identifiers and user-specified policies. The *(ii)* Identity Layer enables identification of participants, whereas the *(iii)* Data Layer holds the personal data of a user and provides authorization accordingly. The *(iv)* Application Layer refers to programs used to process confidential data according to user-specified policies.

Verifiable Registry A verifiable registry is an append-only data structure, which is collaboratively maintained by a network of mutually distrustful parties. The verifiable registry represents a *digital bulletin board*, whose current state is publicly verifiable and agreed upon by a set of validator nodes through a consensus algorithm. In the remainder of this work, we consider the verifiable registry to be a distributed ledger $\mathcal{L}$, as it constitutes the most commonly used data structure that fulfills the needs of a verifiable registry [7]. Unless otherwise stated, we consider a *permissionless* distributed ledger where users can join and leave at any given point in time.

Identity Layer. Each user in a system is associated with an *identifier*. To join, a user has to generate a key pair (sk, pk), where the identifier is derived from the generated public key. In a permissionless distributed ledger, each user can obtain multiple pseudonymous identifiers. It depends on the underlying distributed ledger whether a user can derive multiple identifiers from the same key pair, or needs to generate a new key pair per identifier. Identifiers and key pairs are collectively maintained through wallets.

Data Layer. The Data Layer holds the personal data that is associated with an identifier. We consider personal data to exist in the form of data items $d_i^{\mathcal{U}} \in \mathcal{D}^{\mathcal{U}}$, where each data item is controlled by a user $\mathcal{U}$. Unless otherwise specified, we make no assumptions about the underlying structure of a data item. Data items may either be stored locally or in an outsourced, distributed data storage [8], e.g., if the size of the data item prohibits local storage in a wallet. If a data item is attested by a third party, we refer to it as *credential*. A credential consists of a set of *claims* made about the *subject* of the respective data item. Throughout this work, we focus on how a data item can remain confidential through user-specified policies.

Application Layer. Applications may process personal data if they fulfill the policies specified by the user. We assume applications in the form of programs, which are executed *locally*, on a user's machine, or *externally*, on an dedicated compute node. The correctness of program execution remains auditable via the verifiable registry.

2.2. System Roles

Roles of participants differ in the three surveyed sub-areas. A user either presents statements on its personal data in the case of decentralized identity, or gives access to its personal data for decentralized access control and policy-compliant decentralized computation. For clarity, we define system roles in the *(a)* decentralized identity model for decentralized identity and *(b)* data consumer model for decentralized access control and policy-compliant decentralized computation. The user is a common role between both models, and hence, defined first.

User. A user can be identified by *(i)* a unique identifier, or *(ii)* the semantics of its personal data. In general, we consider the user to be a person or an organization. For decentralized identity, the user is the holder of a credential. A credential, in this context, refers to a digital document containing verifiable claims about the user's identity attributes, signed by an issuing authority.

2.2.1. Decentralized Identity Model

In our model of decentralized identity, we assume that an entity can either act as an issuer, a verifier, or a user. Any entity may take more than one of the introduced roles.

Credential Issuer. The issuer asserts claims about a user 's attributes in the form of a credential. Therein, an issuer vouches for the validity of claims about user-specific attributes. For example, issuers can be corporations, governmental authorities, and individuals.

Verifier. The verifier determines whether or not to provide a service to the user based on the validity of the presented information. For confidentiality, the verifier needs to verify a proof as presented by the user. For example, verifiers can be websites, security personnel, and data consumers.

2.2.2. Data Consumer Model

In the data consumer model, we assume that an entity can either act as a user, an access controller, a compute node or a data consumer. The user intends to limit access to its personal data. Any entity may take more than one of the introduced roles.

Access Controllers. An access controller maintains the secret key encrypting the user's personal data. The access controller can either be represented as a distributed committee or as a centralized entity, which e.g. runs trusted hardware. An access controller disseminates the decryption secrets of an owner's confidential data to data consumers based on the user-specified access policy.

Compute Nodes. A compute node executes a computation on a user's personal data, according to a program supplied by the data consumer. The execution of the program needs to abide with the privacy requirements as specified by the data owner. The compute node further updates policies upon successful program execution.

Data Consumers. A data consumer intends to either *(i)* access encrypted personal data or *(ii)* execute a program on a user provided data item. To successfully access or process user data, the data consumer needs to ensure that its actions comply with the user's policy.

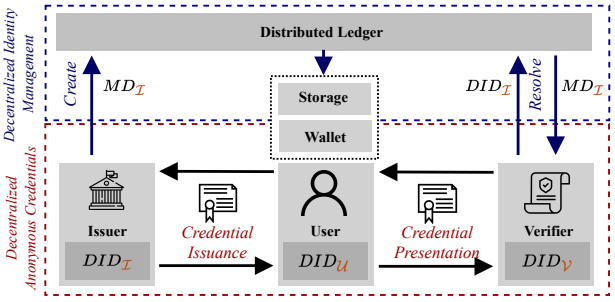

Figure 2: Overview of the Decentralized Identity Model. Each entity is identified by a pseudonymous DID. The issuer $\mathcal{I}$ issues a credential to the user $\mathcal{U}$. Upon presentation of the credential, the verifier $\mathcal{V}$ can verify its validity without interacting with $\mathcal{I}$ by resolving the issuer DID to obtain the issuer metadata $MD_{\mathcal{I}}$.

2.3. Threat Model & Assumptions

We consider a variety of adversarial objectives and capabilities for our definitions and properties. We emphasize that the core motivation behind decentralization is the assumption that all entities are potentially corruptible, while still requiring security and availability in the system.

- *Arbitrarily deviating adversary:* In the remainder of this work, we assume that any entity may deviate from the protocol to maximize its own benefit. Participants may act maliciously or abort at any time. In the decentralized identity model, we assume that an adversarial issuer may collude with a user to issue a forged credential, in an attempt to impersonate another user. Further, a user may attempt to convince the issuer to issue invalid credentials. We assume that verifiers and users do not collude, such that only valid credential presentations are treated as such. In the data consumer model, we assume that an access controller may attempt to evaluate a program without regarding the user's policy. A data consumer may attempt to access a resource without adhering to the user-specified policy. And, an adversarial user may intend to withhold resources after successful authentication of the data consumer. Compute Nodes may attempt to execute an arbitrary program that is not validated by the access controllers.
- *Distributed Ledger:* We assume that the distributed ledger is entrusted for correctness, but not privacy, and includes transactions with a bounded delay. The current state is publicly exposed to all participants.
- *Network & Primitives:* We assume the security of standard cryptographic primitives and secure channels for message transmission. An adversary can observe the network traffic between any of the above-mentioned parties and may arbitrarily delay messages.

Following definitions and formalizations, as in Appendix B, account for the described threat model.

3. Decentralized Identity

Web 2.0 — Centralized Identity. In conventional identity systems, users can authenticate to service providers by recalling their combination of username and password [9]. To increase usability for the user, web *service providers* rely on *identity providers* [10]. The service provider trusts the identity provider with the authentication of the user. As a result, the identity provider serves as an intermediary, maintaining user identities for numerous online services. In conventional systems [11], [12], a user's personal data is stored with either the identity provider or the service provider, leaving little protection to internal misuse and targeted attacks [1]. Note, that when a service provider does not employ an identity provider, the service provider takes over user authentication itself. As a consequence, a user's identity is controlled by third parties in Web 2.0.

Web 3.0 — Decentralized Identity. Decentralized Identity aims to provide the technical foundations to eliminate the shortcomings of centrally managed digital identities. A system for decentralized identity ensures that *(i)* digital identities can be maintained without a centralized entity, and that *(ii)* a user can authenticate without relying on a third party. A decentralized identity provides a solution to both challenges through decentralized identifiers [13] and publicly verifiable credentials [14]. With publicly verifiable credentials, users can decide what information is shared to requesting third parties. However, the decentralized identity ecosystem can be hard to grasp due to a lack of formal and concise definitions. To establish clarity, we divide approaches to decentralized identity into its two main areas [13], [14] — *(i)* decentralized identity management, which decouples identifiers from centralized registries, and *(ii)* decentralized anonymous credentials, which expand on *(i)* by introducing *publicly verifiable credentials*. Note that that standardization of the World-Wide-Web Consortium (W3C) for credentials mainly considers non-private credentials (i.e., Web Tokens [14], [15]). In this work, we explicitly focus on decentralized anonymous credentials.

3.1. Decentralized Identity Management

In this section, we discuss the creation of, and critical operations on, decentralized identifiers. In the following, we provide the *first* formal definition that specifies the relevant components for decentralized identity management.

3.1.1. Formalization

We assume the existence of a public, permissionless distributed ledger $\mathcal{L}$ and data items $d_i^{\mathcal{U}}$ as specified in Section 2. Therein, we consider data items in the form of attribute value pairs (a, v) (e.g., $(age, 18)$). Each Decentralized Identifier (DID) is associated with metadata, $MD_{DID} = \{(a_1, v_1), (a_2, v_2), ..., (a_i, v_i)\}$. The metadata contains cryptographic material (e.g., to verify credentials), and specific service endpoints (e.g., URL). By maintaining metadata on-chain, anybody may inspect an identifier's information, ensuring that user statements are publicly verifiable.

Definition 1. *(*Decentralized Identity Management.*) A system for decentralized identity management employs the following algorithms:*

1) **SETUP**: $1^\lambda \rightarrow pp$: *Given a security parameter λ the system outputs a set of public parameters pp.*
2) **GENERATE**: $KeyGen(pp) \rightarrow (pk, sk)$, *A pair of keys is generated using a randomized algorithm. The keypair consists of a public key pk and a private key sk, which is known only to the owner. For* **GENERATE**, *different key generation protocols can be applied.*
3) **CREATE**: $(sk, pk) \rightarrow (DID, MD_{DID})$. *Given pk, the protocol checks the sk for valid ownership. It outputs a globally unique persistent identifier DID and the associated metadata $MD_{DID} \in \mathcal{L}$ (widely known as DID document). If the identifier is public, the metadata is registered on-chain, otherwise it remains local.*
4) **RESOLVE**: $(DID) \rightarrow MD_{DID}$. *Given a DID, the resolution protocol outputs MD_{DID} for DID.*
5) **UPDATE**:$(DID, MD_{DID}) \rightarrow \hat{MD}_{DID}$. *The holder of the identifier DID may invoke an update function on the existent associated metadata MD_{DID}.*
6) **DEACTIVATE**: (DID), *The holder of a DID may deactivate the DID and the associated MD_{DID} by calling deactivate. DID, and MD_{DID} will be unusable and non-retrievable from that point onwards.*

3.1.2. Security & Privacy Properties

- **Uniqueness:** Each entity has a globally unique identifier. Identifiers can be derived from cryptographic keypairs or the address of a user's wallet.
- **Pseudonymity:** A system provides pseudonymity, if a user can generate several pseudonymous identifiers to hide its identity. No entity may identify the user through one or more of its pseudonyms, nor link different pseudonyms together.
- **Persistence:** Each identity is long-lasting, unless the user deactivates it. Persistence is fundamental for achieving portability and interoperability.

3.1.3. SOTA — Decentralized Identity Management

Decentralized Identity Management is defined in the standardization of the W3C [13], which most recently got adopted as an official W3C recommendation. The standardization defines the format of globally unique identifiers, and outlines their implementation in practice. While the general *format* of a decentralized identifier is specified by the standardization for decentralized identifiers, the *method* for creation and resolution of a decentralized identifier is application specific [16]–[18]. At the time of writing, there exist more than 130 methods that specify the creation of decentralized identifiers in their ecosystem [19]. We identify three dimensions with regard to Decentralized Identity Management that are to be considered when developing a DID method.

Identifier Creation. In each method, the creation of a decentralized identifier is specified either *explicitly* or *implicitly*. As an example, the most popular method for creation of identifiers on Ethereum [20], specifies identifier creation implicitly by expanding existing Ethereum addresses into the format of decentralized identifiers (e.g., `did:ethr:0xb9...`). To create an identifier explicitly, a method may specify a key derivation algorithm, which determines the direct derivation of an identifier from a public key [16].

Identifier Resolution. Decentralized Identifiers improve upon existing globally unique identifiers by ensuring cryptographic verifiability without an intermediary through *decentralized resolution* [13]. Decentralized resolution describes the resolution of an identifier to its associated metadata. The metadata contains cryptographic material for signature verification, and service endpoints associated with the owner of the identifier. Hence, decentralized resolution is one of the *core improvements* of decentralized identity. Every resolution mechanism demands for a *resolver* that resolves an identifier to its metadata. The resolver is operated by a user, or as an off-chain API service to enhance usability [21], [22]. The resolution of an identifier to its metadata can be conducted *locally* or *ledger-based*. While local resolution allows for deterministic decoding of an identifier to its metadata [16], ledger-based resolution requires the publication of metadata on a publicly available distributed ledger [20]. Local resolution is applied in peer-to-peer messaging, where communication with a public registry incurs a significant overhead.

Metadata Storage. By themselves, identifiers are no means to ensure the authenticity of an identity. For a user, the utility of its decentralized identity is directly proportional to the information that is associated with it. The metadata of an identifier may describe verification methods for credentials and service endpoints, to e.g. websites and external storage. A common use of service endpoints is referencing a public and content-addressable distributed storage, which holds further data associated with the owner of the identifier [8], [18], [23]. In general, one can distinguish storage solutions for decentralized identity into *(i)* on-chain storage, where data associated with an identifier is maintained through a smart-contract [24]–[26], and *(ii)* off-chain storage, where sensitive data is maintained in either an external or local storage [18], [23]. Choosing a suitable solution for storage of identity related data is both dependent on the type of information shared and for which use-case it is needed. In contrast to on-chain storage, external storage provides an affordable solution for storing large amounts of personal data and users can also store encrypted data off-chain.

3.1.4. Discussion — Dec. Identity Management

***Remark* — When do you need a decentralized identifier?** A DID is beneficial in a system with distributed trust, where participants intend to represent themselves without relying on a centralized registry for creation, resolution and deactivation of identifiers. As such, first applications emerged that make use of DIDs in practice. Ceramic makes use of DID to identify data streams written to IPFS [18]. Gitcoin leverages this system to create a user-specific "passport" (i.e. datastream), where users can add "stamps" (i.e., credentials) based on social profiles to their datastream to enhance trust and sybil-resistance. A user can prove that a passport belongs to themselves without having to ask a central system for permission.

***Remark* — What are existing limitations?** One major limitation of the current state of decentralized identity is interoperability. When a user with an ETH DID intends to communicate with a user that is registered with a Ceramic DID, resolution mechanisms of either DID methods need to be known to both parties. The prevalent solution for interoperability between DID methods is a *universal resolver*, which has been developed as part of a community effort to enhance usability [21]. However, the universal resolver reintroduces centralization to a certain extent. Whereas anyone can theoretically run a resolver, it is common practice for platforms to provide resolution services for enhanced usability. A resolution mechanism that is universal and scalable, yet rooted in distributed trust, is yet to be proposed in the community.

***Use-Case* — Decentralized Web Applications.** In Web 2.0, personal data is distributed in centralized data silos, diminishing privacy and limiting usability. Cross-corporational data silos are uncommon due to regulations and non-existent cooperations [4]. With decentralized identity management, users can enable services to access their *Decentralized Web Node (DWN)* (i.e. personal data store) [27]. Applications that instantiate DWNs rely on content-based distributed storage solutions (i.e., IPFS) [18], [23]. Importantly, a DID is essential to locate the DWN without relying on an intermediary. Whereas currently, a user's data is dispersed in data silos controlled by corporations, a DWN acts as a user's personal and controllable data silo. In a practical scenario, multiple services may access the same DWN to rely on the same data (i.e. different services based on the same travel itinerary). With DWNs, one can facilitate a novel set of decentralized web applications, which rely on DWNs rather than dispersed data silos to provide personalized web experiences.

***Use-Case* — On-Chain Account Management.** On-chain accounts are smart contracts, which may contain arbitrary logic to e.g., handle social recovery or multisignature authorization [28]. In this case, the identifier is the address created when creating the smart contract. This increases the flexibility in designing secure wallet solutions, which can even be controlled by multiple secret keys. Further, users may whitelist specific accounts, such that unaware reception of arbitrary tokens becomes infeasible.

Arguably, there is not yet a one-shot solution for decentralized identity management that covers both use-cases as outlined above. Weyl *et. al* introduce *soulbound* tokens & information [6], which provide a generalized notion for non-transferable data items that are bound to an individual's identity. Their definition applies regardless of whether a data item is stored on-chain, in the form of non-transferable tokens, or off-chain, in the form of a non-transferable credentials.

3.2. Decentralized Anonymous Credentials

Although credentials are well researched from a cryptographic perspective, they are under-analyzed in a decentralized context. The common baseline understanding is that a credential is a signed, non-transferable digital document, where the signature of a credential issuer attests to the validity of the credential's content. Decentralized Anonymous Credentials pursue the goal of associating unforgeable attestations (*credentials*) with an identifier, such that its owner is enabled to authenticate without relying on an intermediary. In the following, we aim to uncover synergies and deficiencies of varying proposals for decentralized anonymous credentials.

3.2.1. Formalization

In a system for decentralized anonymous credentials, each user is in control of a list of credentials, where each credential consists of a set of attribute-value pairs. The credential is computed with the issuer's secret key and can be non-interactively proven by the user to *any* verifier without the need to communicate with the issuer. Our definition does not assume any specific choice of cryptographic primitives, as we aim to enable general applicability of our definition.

Definition 2 (Decentralized Anonymous Credentials.)**.** *For decentralized anonymous credentials, a system for dec. identity management is augmented with the following participants,*

1) $\mathbf{I}$ *is the set of issuing authorities.*
2) $\mathbf{U}$ *is the set of users.*
3) $\mathbf{V}$ *is the set of verifiers.*
a tuple of spaces $(\mathcal{C}^{\mathcal{U}}, \mathcal{R}^{\mathcal{I}})$*,*
1) $\mathcal{C}^{\mathcal{U}}$ *is a set of credentials of* $\mathcal{U}$*,*
2) $\mathcal{R}^{\mathcal{I}}$ *is the set of credentials revoked by* $\mathcal{I}$*.*
and the following algorithms:

1) **REQUEST***:* $\mathcal{U} \times (pp, d_i^{\mathcal{U}}, aux, sk_{\mathcal{U}}) \rightarrow$ REQ*. This algorithm is executed by the user* $\mathcal{U}$*. On input public parameters* pp*, a data item* $d_i^{\mathcal{U}}$*, auxiliary data* aux *justifying the granting of the credential, and the user's secret key* $sk_{\mathcal{U}}$*, this algorithm outputs a credential request* REQ*.*
2) **ISSUE***:* $\mathcal{I} \times (pp, \text{REQ}, sk_{\mathcal{I}}) \rightarrow cred$*. This algorithm is executed by the issuer* $\mathcal{I}$*. Given public parameters* pp*, the credential request* REQ*, and the secret key* $sk_{\mathcal{I}}$ *of the issuer, the issuance protocol outputs a credential* $cred \in \mathcal{C}^{\mathcal{U}}$*.The issuer vouches for the validity of attributes in* $cred$*. The user stores the credential locally or on a distributed ledger. Not knowing the user's secret key ensures that the issuer cannot forge a proof without the consent of* $\mathcal{U}$*.*
3) **PROVE***:* $\mathcal{U} \times (pp, cred, sk_{\mathcal{U}}, \phi_{cred}) \rightarrow (\pi)$*. This algorithm is executed by the user* $\mathcal{U}$*. Given public parameters* pp*, the credential* $cred$*, the secret of the user* $sk_{\mathcal{U}}$*, and a predicate* ϕ_{cred}*, this algorithm outputs a proof* π*, which proves that* $cred$ *is well-formed with regard to the user-specified predicate* ϕ_{cred}*.*
4) ***VERIFY****:* $\mathcal{V} \times (pp, \mathcal{R}^{\mathcal{I}}, cred, \phi_{cred}, \pi, DID_{\mathcal{I}}) \rightarrow \{0,1\}$ *This algorithm is executed by the verifier* $\mathcal{V}$*. Given public parameters* pp*, the set of credentials revoked by the issuer* $\mathcal{R}^{\mathcal{I}}$*, the credential of the user* $cred$*, the predicate* ϕ_{cred}*, the proof* π *and the identifier of the* $\mathcal{I}$*, the algorithm outputs a binary value, indicating the success of verification.*
5) **REVOKE***:* $\mathcal{I} \times (pp, cred, sk_{\mathcal{I}}, \mathcal{R}^{\mathcal{I}}) \rightarrow \mathcal{R}^{\mathcal{I}'} \cup \{\perp\}$ *This algorithm is executed by the issuer* $\mathcal{I}$*. Given public parameters* pp*, a previously issued credential* $cred \in \mathcal{C}^{\mathcal{U}}$*, the issuer's secret key* $sk_{\mathcal{I}}$ *and the set of revoked credentials* $\mathcal{R}^{\mathcal{I}}$ *of an issuer* $\mathcal{I}$*, the revocation protocol outputs the set of revoked credentials* $\mathcal{R}^{\mathcal{I}'} = \mathcal{R}^{\mathcal{I}} \cup cred$*. If* $cred$ *is not issued by* $\mathcal{I}$*, the protocol returns* $\perp$*.*

3.2.2. Security & Privacy Properties

- **Unforgeability:** A system for decentralized identity is unforgeable, if it is infeasible for any malicious user to present a poof π that **VERIFY** would accept. Unforgeability is typically assured through the unforgeability of the underlying cryptographic primitive used for the issuance of a credential.
- **Selective Disclosure:** A user $\mathcal{U}$ should be able to show proof of a subset of claims in a credential. A naive approach to selective disclosure is the issuance of a new credential for each proof presentation. Selective disclosure can be enabled with a Zero Knowledge Proof [29].
- **Predicate Provability:** Predicate Provability allows the user to make an assertion about attributes in a credential. To achieve predicate provability, the predicate ϕ_{cred} specifies the expressivity of the proof π. The predicate may specify a range proof for an attribute value pair (a_1, v_1), such that $(age \geq 18)$. See [30] for in-depth treatment of predicate proofs for anonymous credentials.
- **Non-Transferability:** A system supports *non-transferability* if the credential issued is bound to a single holder, and cannot be presented by another user as its own. Non-Transferability prevents lending of credentials. It is either achieved by encoding highly sensitive information into the secret key, so that the user will be reluctant to share it [31], or by embedding the public key of the user in the credential.
- **Anonymity:** Requires that an adversary identifies a user $\mathcal{U}$ from a set of users $\mathbf{U}$ with negligible probability when observing the *presentation* or *issuance* of a credential. Accordingly, the proof π should not reveal the identity of $\mathcal{U}$. In this case, even the issuer cannot identify the user when observing a credential proof.
- **Unlinkability:** Given at least two executions of **PROVE** or **ISSUE** by at least two users $\mathcal{U}_1.\mathcal{U}_2 \in \mathbf{U}$, the system supports unlinkability, if it is infeasible for an adversary to distinguish which proof presentation belongs to whom. If credentials are unlinkable, the system is considered to be *multi-show* [32].
- **Credential Revocation:** A system supports revocation, if a credential issued at time t_1 can be invalidated at time $t_2 > t_1$. Revocation can be achieved through short validity periods or public credential revocation lists. The revocation status should be auditable by any verifier and credential holder. We refer readers to [33] for an introduction to revocation for anonymous credentials.
- **Anonymity Revocation:** A system supports anonymity revocation if the issuer $\mathcal{I}$ may disclose the identity of the user upon misbehavior. In practice, anonymity revocation is an additional agreement between the user and verifier which defines conditions under which a verifier is able to request the identity of users from the issuer [31], [34], [35].

3.2.3. SOTA — Decentralized Anonymous Credentials

In recent years, there has been a plethora of proposals for anonymous credentials, varying in specifications and system objectives [29], [31], [32], [36]–[48]. Yet, only few of them are specified for decentralized identity [38], [46]–[48]. A credential can be proven without violating the user's privacy rights by predicating statements on attested attributes. The W3C standard for verifiable credentials (VCs) intends to provide a standard interface for decentralized anonymous credentials [14]. Therein, two primitives are considered for implementation — anonymous credentials based on CL signature and BBS+ signatures [14]. CL signatures [29], which are the basis for

the first practical anonymous credential scheme, suffer from a proof size linear in the number of attributes in a credential. BBS+ signatures [29], [49] enable the same functionality as CL signatures, with the benefit of shorter key and signature sizes for a comparable level of security. Commonly, all anonymous credentials rely on the public key of the issuer to verify the authenticity of a credential presented by the user. This motivates the link between decentralized identity management and decentralized anonymous credentials, to ensure that the verifier obtains the public key of the issuer without an intermediary (cf. Figure 2). However, not all solutions for decentralized anonymous credentials follow the W3C verifiable credential specification [36], [50]. We further dissect the current state of the art in academia and industry by identifying issues and solutions for the *issuance* and *verification* of a credential and associated proofs.

Credential Issuance. Typically, a credential is issued through a single issuer, which applies a group signature on a committed value [29]. Sonnino *et. al* identify the issuer as a single point of failure, and propose an anonymous credential scheme with threshold issuance [48]. Doerner *et. al* propose a similar protocol for threshold issuance for BBS+ signatures [51]. However, this does not alleviate the need for auxiliary data, which is used by the issuer to validate claims asserted by the user. Rosenberg *et. al* pursue a similar goal and eliminate the need for an issuer that is trusted for cryptographic keys by constructing a credential without signatures [50]. In their construction, credentials are on-chain commitments. The user proves the opening of the commitment and the fulfillment of access criteria in zero-knowledge. Maram *et. al* use decentralized oracles to bootstrap credentials without explicit authorization of the issuer by providing a proof of data provenance [52]. They ensure sybil resistance by deduplicating credentials over a unique identifier (i.e., SSN).

Credential Presentation and Proof. Once issued, a decentralized anonymous credential is commonly stored in a user-owned wallet. The user can present a proof that the claims are valid with respect to a user chosen predicate. Therein, proofs can either be verified off-chain [46], [47], [53], through an off-chain verifier program, or on-chain, through a smart contract [48], [54]. Note that many applications deem non-fungible tokens (NFTs) as a valid option for gathering evidence of experiences in non-digital and digital environments (i.e., *metaverse*) [55]. Nevertheless, NFTs are transferable, while, in reality, experiences and attendance are not. Associating experiences, attendance and attributes with a digital identity requires for non-transferable tokens and credentials. Credentials for on-chain verification can significantly broaden the scope of applications for Web 3.0 (e.g., using an identity as collateral to facilitate undercollateralized lending), whereas it is to be shown that they can be sufficiently efficient.

3.2.4. Discussion — Dec. Anonymous Credentials

***Remark* — Trustless credential verification demands for decentralized identity management.** In general, all implementations of decentralized anonymous credentials (DACs) follow a similar rationale. To receive a credential, the user commits to its personal data with a hiding and binding commitment. The issuer stores its public key and metadata on a public distributed ledger (i.e., *on-chain*) for verifiability. To issue a credential, the issuer attests to the attributes of the user. To remain anonymous, a user can prove in zero-knowledge to any entity that *(i)* it is in possession of the corresponding secret key included in the credential, and that *(ii)* the disclosed attributes in the credential satisfy a specified predicate [29]. Upon verification, a service provider does not communicate with the issuer because the public key of the authority is obtainable through decentralized resolution.

***Remark* — Legacy Compatibility can increase the adoption of decentralized anonymous credentials.** Verification demands for trust in the issuer, which vouches for the validity of the user's attributes. Assuming the existence of an issuer is a strong assumption, as authorities might not comply with novel architectures in the years to come. Connecting the decentralized identity ecosystem to legacy architectures is therefore indispensable. Issuing credentials from legacy data requires proving where the data comes from without explicitly relying on an issuer. However, accessing and proving possession of legacy data is challenging because the user needs the permission of an authority to export its personal data. Maram *et. al* propose to use Transport Layer Security (TLS) to port credentials to a decentralized context [52], [56], [57]. By providing legacy compatibility, adoption of the DID ecosystem can be increased without demanding for server-side changes.

***Challenge* — On-chain verification for credentials.** In general, anonymous credentials are costly to verify due to their strong privacy guarantees and the computationally expensive Zero Knowledge Proof (ZKP)s that are applied for predicate provability [30]. Sonnino *et. al* [48] present anonymous credentials that enable verification through a smart contract with capabilities for predicate provability. However, their construction is yet too expensive for on-chain verification, primarily due to the costly proof verification (i.e., verifying a credential from their Coconut scheme would cost an equivalent of approx. $950 at the time of writing). Rathee *et. al* reduce the cost of on-chain verification by composing SNARKs with two layers of recursion [54]. Efficient verification of decentralized anonymous credentials is a nascent area, where general-purpose ZKPs are a promising alternative to special purpose anonymous credentials [50], [54], [58]–[60]. We expect that future works will apply general-purpose ZKPs in constructions for decentralized anonymous credentials.

***Challenge* — Issuing sybil resistant credentials at scale.** To be resistant against a sybil attack, an issuer should ensure that no user can obtain two different identities. Sybil Resistance can be addressed with credential deduplication [52]. Deduplication of a credential by relying on unique real-world identifiers is only possible during issuance, because tracking duplicates after issuance is difficult due to anonymity and unlinkability. Note that sybil resistant credentials can further prove the *personhood* of a user, when being issued through a per-person unique identifier (i.e. SSN). Otherwise, a *Proof of Personhood* can only be obtained through biological traits or pseudonym parties [61]. However, an inherent limitation is that issuers are oblivious to credentials issued by other equivalent issuing authorities. So far, there is no discussion or proposal

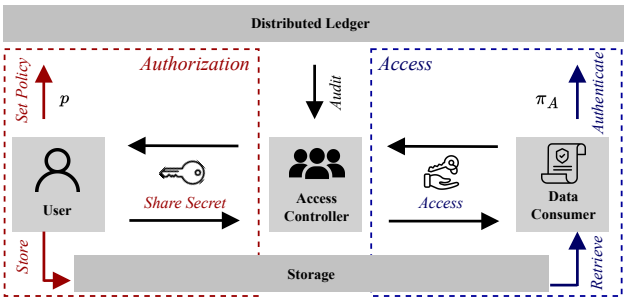

Figure 3: Overview of Decentralized Access Control. $\mathcal{U}$ sets a policy p, stores its personal data and shares its secret with access controllers **AC**. The data consumer $\mathcal{C}$ can only retrieve the plaintext if it can prove compliance with p.

in the literature on how authorities may coordinate for sybil-resistant attribute attestation.

***Use-Case* — DACs can ensure AML and KYC for decentralized applications.** With support for anonymity revocation, solutions for efficient on-chain verification of decentralized anonymous credentials can realize novel applications in decentralized finance by providing *identity as collateral* [6]. Protocol participants can use their identity and reputation as collateral, such that their identity is disclosed to authorities upon misbehavior. By adopting this paradigm, credentials with on-chain verification can be a solution for privacy-preserving Know Your Customer (KYC) in decentralized applications to comply with Anti-Money Laundering (AML) regulations [62].

4. Decentralized Access Control

Web 2.0 — Centralized Access Control. Access control allows disclosure of personal data by authorizing and authenticating data consumers [63]. To date, Web 2.0 access control systems rely on federated protocols, where centralized access controllers handle data protection and access management on behalf of users [11], [12], [64], [65]. Users control data access by configuring access policies at access controllers, which disclose personal data to policy-compliant data consumers. Key issues of Web 2.0 access control are that (i) users have to fully trust third-party access controllers to protect personal data, (ii) users are limited in specifying their policies as access controllers adopt pre-defined configurations of access policies, and (iii) users trust access controllers to transparently and correctly disclose regulation-enforced access audit logs. The fundamental disadvantage of Web 2.0 access control is that users have to trust access controllers in providing access and handling data faithfully.

Web 3.0 — Decentralized Access Control. Decentralized access control intends to remove the reliance of users on centralized access controllers [66]. In Web 3.0, decentralized access control should allow users to verifiably control who gets access to personal data. As such, users should be able to conditionally control access based on presented credentials, token ownership or Decentralized Autonomous Organization (DAO) membership [67]. We identify the main building blocks of decentralized access control to be *on-chain access policies* and *decentralized secret management*. On-chain policies can govern decentralized access management through transaction-based authentication of users. A committee of access controllers enforces the policy-based decision and releases the secret accordingly. By obtaining the secret, data consumers can decipher encrypted personal data. Based on this insight, we propose the *first* formal framework for decentralized access control in the following.

4.1. Decentralized Access Control

In this section, when referring to *access control*, we specifically mean how a data consumer can obtain access to raw data given a ciphertext. We assume that a data consumer can obtain the ciphertext from a personal data store by referring to its decentralized identifier (cf. § 3.1).

4.1.1. Formalization

In a decentralized access control system, we consider that each ciphertext $c_i^{\mathcal{U}}$ of a data item $d_i^{\mathcal{U}}$ is decrypted using a secret k. Each secret k is maintained by a set of secret managers in terms of secret shares. A ciphertext is associated with a user-defined access policy p. The policy p of a data item $d_i^{\mathcal{U}}$ contains statements specified by the predicate ϕ. The entities that fulfill the policy is denoted as the subset of authorized data consumers $\mathbf{C}_A \subset \mathbf{C}$.

Definition 3 (Decentralized Access Control System)**.** *A dec. access control system has the following participants:*

1) $\mathbf{U}$ *is the set of users.*
2) $\mathbf{C}$ *is the set of data consumers.*
3) ***AC*** *is the set of access controllers.*

Further, a system for decentralized access control employs the following protocols:

1) **SETUP***:* $1^\lambda \rightarrow pp$ *: Given a security parameter* λ *the system outputs a set of public parameters* pp*.*
2) **SETUP SHARES***:* $\mathcal{U} \times (pp, k) \rightarrow (k_i^{\mathcal{AC}})$*. Given the public parameters* pp *and a secret* k*, it outputs secret shares* $k_i^{\mathcal{AC}}$ *to each member in* ***AC****.*
3) **SET POLICY***:* $\mathcal{U} \times (pp, d_i^{\mathcal{U}}, k, \phi) \rightarrow (p, c_i^{\mathcal{U}})$*. This algorithm is executed by the user* $\mathcal{U}$*. Given the public parameters* pp*, a data item* $d_i^{\mathcal{U}}$*, a secret* k*, and the user-specified predicate* ϕ*. It outputs a ciphertext* $c_i^{\mathcal{U}}$*, which is the encrypted data item* $d_i^{\mathcal{U}}$ *under the secret* k*. The algorithm outputs an on-chain policy* $p \in \mathcal{L}$*, which defines the set of authorized data consumers* $\mathbf{C}_A$*.*
4) **AUTHENTICATE***:* $\mathcal{C} \times (pp, p, aux) \rightarrow (\pi_A) \cup \{\perp\}$*. This algorithm is executed by the data consumer* $\mathcal{C}$*. Given public parameters* pp *and the policy* $p \in \mathcal{L}$*, and auxiliary data* aux*, it outputs a proof* $\pi_A \in \mathcal{L}$*, which publicly proves the authorization of* $\mathcal{C}$*. If the data consumer* $\mathcal{C} \notin \mathbf{C}_A$ *does not satisfy the policy, it returns* $\{\perp\}$*.*
5) **ACCESS***:* $\mathcal{AC} \times (pp, \pi_A) \rightarrow (k_i^{\mathcal{AC}}) \cup \{\perp\}$*. Given public parameters* pp *and a proof* π_A*, access controllers outputs secret shares* $k_i^{\mathcal{AC}}$*, to the data consumer* $\mathcal{C}$ *if* $\pi_A \in \mathcal{L}$ *and* $\mathcal{C} \in \mathbf{C}_A$*, otherwise it outputs* $\{\perp\}$*.*
6) **RECONSTRUCT***:* $\mathcal{C} \times (pp, k_i^{\mathcal{AC}}, c_i^{\mathcal{U}}) \rightarrow (k, d_i^{\mathcal{U}})$*. Given public parameter* pp*, secret shares* $k_i^{\mathcal{AC}}$ *and a ciphertext* $c_i^{\mathcal{U}}$*, the data consumer* $\mathcal{C}$ *first reconstructs the secret* k*.* $\mathcal{C}$ *is then able to get the data item* $d_i^{\mathcal{U}}$ *by decrypting the ciphertext* $c_i^{\mathcal{U}}$ *using the secret* k*.*

7) **UPDATE:** $\mathcal{U} \times (pp, p, \phi) \rightarrow (p')$. *This algorithm is executed by* $\mathcal{U}$. *Given public parameters* pp, *a policy* p, *and a user-specified predicate* ϕ, *the algorithm outputs an updated on-chain policy* $p' \in \mathcal{L}$, *which defines an updated set of authorized consumers* $\mathbf{C}'_A$.

4.1.2. Security & Privacy Properties

We introduce security & privacy properties of a decentralized access control system as follows:

- **Data Confidentiality:** A system for decentralized access control ensures confidentiality if it is infeasible for an unauthorized adversary to obtain information about $d_i^{\mathcal{U}}$ from the ciphertext $c_i^{\mathcal{U}}$.
- **Anonymity:** Anonymity requires that no adversary should be able to distinguish which consumer $\mathcal{C} \in \mathbf{C}_A$ accessed the data item $d_i^{\mathcal{U}}$, unless requested by an audit. Accordingly, the data consumer $\mathcal{C}$ should remain anonymous during **AUTHENTICATE** and **ACCESS**.
- **Auditability:** A system for decentralized access control supports auditability if any invocation of **AUTHENTICATE** and **ACCESS** is publicly observable. In practice, this is achieved by emitting a transaction on the ledger $\mathcal{L}$ upon invocation of either **AUTHENTICATE** or **ACCESS**, which yields an auditable trace for auditability.
- **Policy Confidentiality:** A system for provides policy confidentiality, if the policy is only observable by the authorized set of data consumers $\mathbf{C}_A$. When an access policy p is stored on-chain in plaintext, the system does not support policy confidentiality [66].
- **Fair access:** Fairness is essential to prevent attacks that rely on information asymmetry (e.g., front-running [68]). Fair access indicates that data consumers simultaneously obtain the secret from access controllers.
- **Access Revocation:** A system for decentralized access control supports access revocation, if an authorized data consumer may be revoked at any time.

4.1.3. SOTA — Decentralized Access Control

As a primitive, decentralized access control has thus far not been formally defined in previous works. Several surveys on blockchain-based solutions for access control uncover that most works focus on a specific setting rather than a generic definition [69]. Our formal definition is based on the insight that user-control requires separation of *data consumer authorization* and *decentralized secret management*. We cover the *state-of-the-art* as follows.

Access Policies. An access policy specifies conditions under which a data consumer can access a specific resource. Besides the *confidentiality* of access conditions, a core property of interest is the *expressivity* of the language through which a policy can be specified. Kokoris Kogias *et. al* propose a system for decentralized access control that supports anonymity at the cost of low expressivity [66]. Shafagh *et. al* specify decentralized access control for IoT data streams, providing high granularity in an application specific domain [63]. Either solution yields policies that are observable in plain. Steffen *et. al* specify a language through which a user can specify access conditions as an arithmetic proof circuit, such that access conditions remain confidential [70], [71]. Naturally, this gain in privacy comes at the cost of computational efficiency. Most recently, Spruce [22] presented an experimental version of an access policy language that is able to authorize data consumers based on token or credential ownership [72]. Whilst not supporting policy confidentiality, their proposal serves as a first notion of industry efforts in developing decentralized, conditional access control.

Decentralized Secret Management. Users rely on a distributed committee to securely manage a secret without a centralized intermediary [66], [73]. Distributed committees receive secrets from users and disclose secrets to data consumers if access policy conditions are fulfilled. Alternatively, secure enclaves can be used to disseminate secrets, which comes at the cost of additional trust assumptions [74]. In general, two strategies can be employed when disseminating a secret with a committee of access controllers, either based on *(i)* Verifiable Secret Sharing (VSS) or *(ii)* Distributed Key Generation (DKG) [66]. VSS can be used to disseminate a secret to a *user-chosen* committee. To ensure consistency of shares, the VSS algorithm needs to be publicly verifiable. This paradigm is commonly applied in several orthogonal works, mainly improving upon efficiency and proactivity of the committee of access controllers [75]–[77]. Alternatively, a *fixed* set of committee nodes may perform DKG to obtain a keypair (pk, sk_i), where each node holds a share of the secret key. The user then encrypts its data encrypting key under pk, such that access controllers can obtain their secret share. Even though this paradigm is beneficial from a user perspective, it received less attention than the VSS-based approach for decentralized access control.

4.1.4. Discussion — Decentralized Access Control

***Remark* — Decentralized access control is the missing complementary to decentralized identity.** Decentralized Identity specifies how users can employ decentralized anonymous credentials to retain privacy when presenting their personal data [14]. The verifier creates a request, which specifies the requirements of the predicate applied by the user when presenting its credential. This approach, however, is not applicable for automated verification through a user specified policy, as requests need to be individually re-created for each new proof. Coull *et. al* describe how policies for pre-specified and fine-grained verification can be facilitated through policy graphs [78], [79]. Even though this approach facilitates expressive access control policies, all existing approaches are either *(i)* native to the applied anonymous credential or *(ii)* not applicable to a decentralized environment, and hence lack *auditability*. Ideally, decentralized access control serves as a complementary to decentralized identity by relying on user-owned credentials for policy-based authorization [67]. Combining their synergies for end-to-end *decentralized identity and access management* for personal data remains to be explored in future work.

***Challenge* — Expressive and confidential policies.** Access Control for web applications is well-researched and there are several languages to formulate access control policies [80]. *Expressivity* is a core property of an access policy, which enables the specification of fine-grained permissions for data consumers. Several works aim to adopt existing languages to enable auditable access control through blockchains [81], [82]. Because existing policy languages do not consider confidentiality, they fall short of achieving strong privacy guarantees. Zyskind *et. al*

are the first to propose the usage of distributed ledgers for *privacy-preserving* decentralized access control to personal data stored by an external storage provider [83], [84]. Recent work expands on this paradigm by identifying the essential separation of consumer authorization and key management to grant access to personal data in a distributed storage [63], [66], [85]. Goyal *et. al* suggest formulating the policy as a public condition for decryption, which can be fulfilled by the data consumer by submitting a proof of knowledge [75]. A prevalent, unresolved, tension in recently proposed systems occurs between privacy, expressivity and efficiency. An access policy language that specifies expressive access policies, that provide confidentiality in efficiently verifying conditions set by the user, remains an item of future work.

***Remark* — Cryptographically binding secrets to policies prevents replay attacks.** Replay attacks apply independent of the encryption scheme that is used to disseminate a secret of a user to the committee. In a replay attack on decentralized access control, the attacker tries to trick the system into disclosing the secret associated with a ciphertext to herself [66]. The attacker takes the publicly accessible ciphertext and creates a policy which announces itself as the legitimate data consumer. To counteract the attack, current systems cryptographically bind the user secret to the policy [66]. Additionally, transactions updating the policy expect a proof of knowledge of the secret. This way, an attacker is never able to create valid policies which map to the same secret.

***Remark* — Exploring efficiency improvements for secret dissemination.** A straight forward approach for a user to share a secret with a committee of access controllers is VSS, a special form of *secret-sharing* which enforces verifiable consistency of secret shares [86], [87]. Recent work shows how to enhance this baseline for decentralized secret management with *proactivity* [73], [75], [77], [88] and *scalability* [76]. However, having the user directly disseminate shares to a chosen committee introduces a linear overhead in the transaction size, as all encrypted shares need to be published on-chain upon definition of the policy. Note that all encrypted shares need to be published for data consumers to verify that shares have been correctly decrypted by the committee nodes [66]. Improving upon this drawback can be a valuable direction for future work. By demanding for a *static* committee of access controllers, that maintains a single public key and a shared secret key, this deficiency can be alleviated [66]. The user can encrypt its secret under the committee's public key, such that each access controller can calculate its share on its own. This way, the transaction size of policy updates is independent of the number of committee members. However, proactivizing the scheme requires for round-based re-instantiation of committee members. Exploring efficiency improvements for large, dynamic committees remains an open problem.

***Challenge* — Enhancing security guarantees by selecting blockchain nodes as access controllers.** Recent work proposes a natural extension for decentralized secret management by selecting blockchain validator nodes as access controllers [75], [77]. Next to running a consensus protocol, each validator node participates in Dynamic

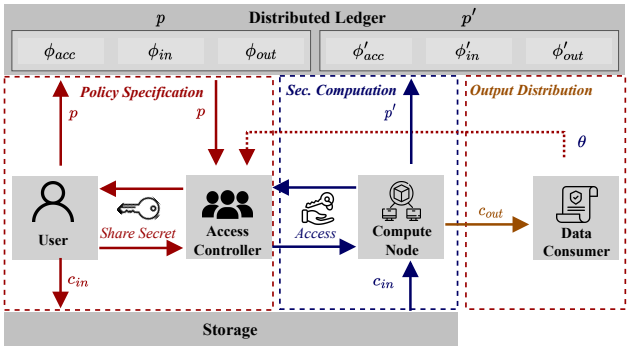

Figure 4: Overview of Policy-Compliant Decentralized Computation (PCDC). $\mathcal{U}$ sets a policy p, stores its personal data c_{in} in storage and shares its secret. The access controller receives the program Θ, analyzes if it fulfills the policy and forwards the program and the secret to the compute node, who executes the computation and atomically updates the policy. The output is successively distributed to the data consumer.

Proactive Secret Sharing (DPSS) to maintain secrets on behalf of users. However, the verification time of DPSS schemes grows linear in the number of committee members. Gentry *et. al* reduce the size of the committee by differentiating between honest and malicious committee members [76]. We are not aware of any analysis of the practicability of similar solutions. We expect that choosing committee nodes to enhance security for decentralized secret management will gain attention in the future.

***Use-Case* — Conditional Access to Decentralized Web Nodes.** Decentralized access control ensures the release of secrets under arbitrary conditions. By default, it represents a powerful primitive for various applications. However, decentralized access control excels in applications that demand for a stronger notion of fairness and auditability. As such, recent industry efforts aim to develop access control languages that specify policies allowing access upon presentation of tokens, certificates, or anonymous credentials [67], [72]. Looking ahead, decentralized access control can be applied to facilitate user-controlled access to DWN's holding user-affiliated personal data. Consider an advertising campaign that analyzes social media data of users who have used a smart contract. Upon receiving payment from the advertiser, the user can disclose their social media identifier to enable the analysis. However, once access is granted through plain decentralized access control, data consumer's may arbitrarily copy and disseminate personal data due to its non-rival nature.

5. Policy-Compliant Decentralized Computation

We now turn our attention to policy-compliant decentralized computation, which approaches the elusive goal of protecting personal data beyond the blanket access to a data item. As such, Policy-Compliant Decentralized Computation (PCDC) improves upon decentralized access control as data consumers only gain access to the computation result rather than the personal data in plain.

Web 2.0 — Centralized Computation. The rapid expansion of the web over the past few decades was coupled

with increased centralization. Few large companies own most systems and servers, and consequently, gather large amounts of data. The accompanied lack of transparency leads to frequent exploits and data breaches [89]. The raw data is either *(i)* not used and remains in data silos or *(ii)* copied, rendering data usage control and auditability infeasible [90]. As a result, users are neither in control over how their data is stored, nor are they reimbursed for the commercial utility their data yields.

Web 3.0 — Policy Compliant Computation. Current approaches to confidential smart contracts for Web 3.0 enable users to confidentially invoke decentralized applications [70], [71], [74], [91]–[95]. However, they solely focus on how users can invoke programs with locally maintained data, without accounting for third-party program invocation, combinations of data sources, and policy-based data governance. We argue that the gap between how Web 3.0 applications are envisioned, and how Web 2.0 applications are currently operating, demands for a novel description of user-governed computation. To account for this dichotomy, we propose the notion of *PCDC*, which allows users to govern the execution of arbitrary external programs. PCDC enables a new class of protocols, where users put their data to use whilst obtaining previously unattainable guarantees for privacy and computational correctness. To achieve privacy and protect data beyond access, PCDC relies on secure computation. The data consumer therefore only observes the result, whereas the personal data, which is used as an input, can only be observed by the user. To ensure auditability, PCDC demands for proofs of computational correctness. At a high level, PCDC follows as a natural extension to decentralized access control (cf. Figure 4). The access controller analyzes the compliance of a program with regard to a user-specified policy. Upon validation, the access controller sends the program, the data decrypting key and the corresponding residual policy to the compute nodes. Once executed, compute nodes deliver the result and update the *residual* policy. In the following, we propose the *first* formal definition of PCDC and outline its relevant security and privacy properties.

5.1. Policy-Compliant Decentralized Computation

5.1.1. Formalization

We assume that a data consumer $\mathcal{C}$ intends to perform a computation on a data item $d_i^{\mathcal{U}}$ by executing a program Θ. The user-specified policy p defines whether a computation can be executed (cf. Figure 3). Each program execution updates the hidden on-chain policy from p to p' (p' is the residual policy). The residual policy includes the updated privacy budget of $d_i^{\mathcal{U}}$ after the execution of the program. The proof of correct computation and policy update is verified by nodes maintaining the ledger. In our abstraction, a policy is specified by three predicates — an access predicate ϕ_{acc}, which specifies access to $d_i^{\mathcal{U}}$, an input predicate ϕ_{in}, which specifies the constraints for consumption of a data item, and an output predicate ϕ_{out}, which determines constraints on the computation output, such as the set of data consumers eligible to observe the result. For ease of abstraction, we assume a *single* policy that contains all predicates. However, note that predicates may be specified by different parties in distinct policies (i.e., *data-use*, *privacy*, and *output consumption policies*). Further, access controllers analyze the policy-compliance of Θ, publish a computation commitment ct_{comp} and a proof of program compliance π_A. The access controllers send information needed for program execution to the compute node. Note that access controllers and compute nodes can be the same entity. We separate them in our definition for semantic clarity — compute nodes execute the program when instructed by the access controllers.

Definition 4 (Policy-Compliant Decentralized Computation)**.** *A policy-compliant decentralized computation system consists of a set of participants, where*

1) $\mathbf{U}$ *is the set of users.*
2) $\mathbf{C}$ *is the set of data consumers.*
3) ***AC*** *is the set of access controllers.*
4) ***CN*** *is the set of compute nodes.*

and the following algorithms:

1) **SETUP***:* $1^\lambda \rightarrow pp$*. Given a security parameter* λ*, the system outputs a set of public parameters* pp*.*
2) **COMMIT***:* $\mathcal{U} \times (pp, d_i^{\mathcal{U}}, k^{in}, s, \phi_{acc}, \phi_{in}, \phi_{out}) \rightarrow (c_{in}, p)$*. This algorithm is executed by the user* $\mathcal{U}$*. Given public parameters* pp*, a symmetric key* k^{in}*, a policy encrypting secret* s*, a data item* $d_i^{\mathcal{U}}$ *and the predicates* $(\phi_{acc}, \phi_{in}, \phi_{out})$*, it outputs the ciphertext* c_{in}*, which encrypts* $d_i^{\mathcal{U}}$ *under the key* k^{in}*, and an on-chain policy* $p \in \mathcal{L}$*, where* p *is a concatenation of a pointer to* c_{in}*, and the associated predicates. Only the entity with the policy encrypting key* s *can update the policy.*
3) **ANALYZE***:* ***AC*** $\times$ $(pp, p, \Theta) \rightarrow (ct_{comp}, \pi_A)$*. This algorithm is executed by* ***AC****. Given public parameters* pp*, the user-specified policy* p *and the program* Θ *provided by the data consumer, it outputs an on-chain computation commitment* ct_{comp}*, which commits to the concatenation of the program* Θ *and the policy* p*. Also it outputs a proof* $\pi_A \in \mathcal{L}$ *that asserts the program-compliance with the user specified policy.*
4) **COMPUTE***:* ***CN*** $\times (pp, s, p, c_{in}, k^{in}, \Theta) \rightarrow$ $(p', c_{out}, k^{out}, \pi_C) \cup \{\perp\}$*. This algorithm is executed by the compute nodes* ***CN****. The input is the public parameters* pp*, the policy encrypting secret* s*, the current state policy* p*, the input ciphertext* c_{in}*, the symmetric key* k^{in} *and the program* Θ*. The algorithm outputs the updated residual policy* $p' \in \mathcal{L}$*, the output ciphertext* c_{out}*, the output key* k^{out} *and the proof of correct computation* $\pi_C \in \mathcal{L}$*.*
5) **CLAIM***:* $\mathcal{C} \times (pp, c_{out}, k^{out}) \rightarrow (d_{out}) \cup \{\perp\}$ *This algorithm is executed by the data consumer* $\mathcal{C}$*. Given public parameters* pp*, the ciphertext* c_{out} *and the output hiding secret* k^{out}*, the algorithm outputs* d_{out}*.*

5.1.2. Security & Privacy Properties

We introduce *security & privacy properties* for PCDC as follows:

- **Confidentiality:** Confidentiality is ensured, if *(i)* only the entities with with the correct encryption key can observe the data item $d_i^{\mathcal{U}}$, *(ii)* only the authorized set of data consumers can observe the policy, *(iii)* only authorized participants can observe the invoked function, and *(iv)* d_{out} is only visible to authorized data consumers.
- **Anonymity:** *User-anonymity* describes that an adversary cannot identify the user $\mathcal{U} \in \mathbf{U}$ which provides data

items as input to the program Θ. *Consumer-anonymity* describes that an adversary cannot identify the data consumer $\mathcal{C} \in \mathbf{C}$, which receives the output of Θ [91]. Note that anonymity can be *optional* for certain applications.

- **Computation Correctness:** Compute nodes generate a proof $\pi_C \in \mathcal{L}$ which attests to the correct computation of a program. Any entity should be able to verify π_C, and the correctness of the updated policy.
- **Atomic Data Delivery:** Ensures that once a program has been executed, the policy is updated to reflect the new policy state p'. The data consumer is unable to claim the output d_{out} without a previously updated policy. Once the policy is updated, the data consumer should always be able to claim the output.
- **Differential Privacy:** A system supports differential privacy, if the presence or absence of one user's information in a dataset, only has a small effect on the output distribution of the computation [96]. In the context of PCDC, privacy consumption is monitored through the residual policy p', which accounts for the updated policy state after the execution of a program.

5.1.3. SOTA — Policy-Compliant Decentralized Computation

We outline the state-of-the-art for PCDC and orthogonal technologies. We discuss the following dimensions:

Policy Specification & Analysis. Prior to executing a computation, it's crucial to specify an on-chain policy that outlines acceptable program execution. Bowman *et. al* propose sharing and processing data according to pre-agreed policies in trusted hardware [97]. Wang *et al.* describe a user-configurable data analysis framework, where policies describe privacy requirements for data items [98]. GDPR [99] or CCPA [2] compliance is achieved by providing a baseline policy upon which the user can add additional conditions. They suggest using *static analysis* to detect privacy violations and quantify privacy degradation through *differential privacy*. The program analysis can either be done by the entity controlling access, or by the provider of the program. If the program provider conducts the analysis, policy compliance can be proven through a *verifiable proof of differential privacy* [100].

Computation. Note, that PCDC is different from systems for confidential smart contracts. Confidential smart contracts do not have a notion of user-defined policies that govern usage of personal data. In both PCDC and confidential smart contracts, only the entity that initialized the state can successively update it. With PCDC, compute nodes may further execute arbitrary stateless programs. Khosba *et. al* proposed confidential smart contracts through dedicated off-chain compute nodes [101]. In optimizing for efficiency, most subsequent approaches rely on trusted hardware to instantiate off-chain compute nodes [74], [97], [102]–[104]. Several other works apply either Multi-Party Computation (MPC) [105]–[107] , Fully Homomorphic Encryption (FHE) [71], [91] or ZKPs [94], [108] to alleviate the need for trusted hardware, with differing notions of privacy concerning the *input*, *output*, and the *function* of each computation. For a comprehensive overview of confidential smart contracts, we refer the interested reader to [109]. We highlight that PCDC is distinctly different from techniques for secure computation (i.e., MPC) and solutions for confidential smart contracts. PCDC focuses on policy-based governance of computation and auditing for compliance, confidential contract execution does not have the notion of policy-compliance, it ensures that the execution of the smart contract and its data is confidential and does not reveal sensitive information to attackers.

Output Distribution. Once the computation is executed, the compute node encrypts the output under the recipient(s) public key(s), and updates the on-chain policy. Wang *et. al* focus on multi-step analyses by re-encrypting the computation output with a residual policy, such that privacy requirements can be satisifed in multiple steps [98]. There are several sub-problems that demand consideration — atomic delivery of computation results and policy updates, ensuring well-behavior of rational participants and adequate incentivization of protocol participants. Cheng *et al.* propose a protocol with atomic output delivery by restricting the key release to the consumer only after the state update has been verified on-chain [74]. Das *et. al* achieve *financial fairness* with a collateral system that punishes malicious parties and TEE operators if they deviate from the protocol [110].

5.1.4. Discussion — PCDC

***Remark* — Policy selection can be delegated to a fiduciary.** Setting a policy for compliant program execution enables user control over data processing. In practice, there are many type of policies that vary in complexity. Users may require complex attribute dependencies and privacy budgets, or resort to a simple whitelist of publicly known programs. However, choosing the appropriate policy can be challenging, as users may struggle in determining appropriate privacy protection. To enhance usability, users can resort to *data fiduciaries* to choose a policy [98], [125]. Relying on a fiduciary does not limit control, as it does not restrict the degree of programmable privacy.

***Remark* — An auditable privacy budget ensures user–controlled degradation of privacy.** Validating the compliance of programs with respect to data privacy regulations may seem enough to preserve privacy. Unfortunately, this is not the case, as an attacker can accumulate computation results [102]. With differential privacy, a user can set a privacy budget (quantified by the parameter ϵ) to prevent information leakage over multiple observations. In our definition, we require user-controlled privacy degradation. Access controllers analyze the submitted program, update the privacy budget, and generate a residual policy.

***Remark* — Analyzing a program yields a dichotomy between function privacy and policy confidentiality.** Analysis of a program for conformity with the user-specified privacy budget can either be conducted by the provider of the program, or the access controller. However, either approach has yet to be explored, and either approach has its unique pitfalls to consider. When the program provider creates a proof of differential privacy, knowledge of the policy is a necessity. When the access controller analyzes the program, the program must be known for analysis. Further, this shows the relevance of function privacy for PCDC during both program execution *and* program analysis for updating the policy. Proposing a solution which unravels the dichotomy between func-

Layers	Functionality	Academia Solutions					Industry Solutions																
		Candid†,▼ [52]	Coconut*,▼ [48]	DAC▼ [111]	Zk-creds*,▼ [50]	Calypso [66]	SIWE (ERC-4361) [112]	SpruceID†,+ [22]	iden3 (go-iden3-core)*,▼ [59]	PolygonID*,▼ [60]	Mattr†,+,▼ [113]	Trinsic†,+,▼ [114]	Veramo†,+ [53]	Aries (ACA-Py)†,+,▼ [115]	Sismo▼,* [116]	SemaphoreID▼,* [117]	Verite†,+ [118]	Gitcoin Passport†,+ [119]	Serto†,+ [120]	PAD Protocol [121]	Ocean Protocol [122]	Oasis Parcel [123]	iExec [124]
Decentralized Identity Management (Definition 1)	Generate	●	●	●	●		●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	●
	Create	●	○	●	○		●	●	●	●	●	●	●	●	●	●	●	●	●		●	○	○
	Resolve	●	○	●	○		●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	○
	Update	●	○	●	○		○	●	●	●	●	○	●	●	●	○	○	●	○		●	●	○
	Deactivate	●	○	○	○		○	●	●	●	●	○	●	●	○	○	○	●	○		○	●	○
Decentralized Anonymous Credentials (Definition 2)	Request	◐	●	●	●			◐	●	●	●	●	◐	●	●	○	◐	◐	◐				
	Issue	◐	●	●	●			◐	●	●	●	●	◐	●	●	●	◐	◐	◐				
	Prove	◐	●	●	●			◐	●	●	●	●	◐	●	●	●	◐	◐	◐				
	Verify	◐	●	●	●			◐	●	●	●	●	◐	●	●	●	◐	◐	◐				
	Revoke	◐	○	○	●			○	●	●	●	●	○	●	○	●	◐	○	○				
Decentralized Access Control (Definition 3)	Setup Shares					●														●	○		
	Set Policy					●														○	○	●	○
	Authenticate					●														●	●	●	○
	Access					●														●	●	●	●
	Reconstruct					●														●	○		
	Update					●														●	●	●	○
Policy-Compliant Decentralized Computation (Definition 4)	Commit																					●	●
	Analyze																					○	○
	Compute																					●	●
	Claim																					●	●

TABLE 1: Functionality provided by industry and research solutions with regard to definitions as in sections 3 — 5. A functionality is either addressed ●, not addressed ○ or not applicable. For decentralized identity: † = compliant with W3C DID; ✢ = compliant with W3C VC; ❋ = supports on-chain verification; ▼ = supports privacy-preserving credentials. If a solution supports credentials without anonymity, we denote it with ◐.The methodology for inclusion and exclusion of a solution is outlined in Appendix A.

tion privacy and policy confidentiality, without relying on trusted hardware, remains an open problem.

***Challenge* — Concurrent policy updates.** Concurrency describes asynchronous out-of-order evaluation of a program [126]. When executing a program, compute nodes need to update the policy, including the associated privacy budget. Policy updates can lead to the occurrence of race conditions, when more than one entity tries to commit a state update to the ledger simultaneously. All solutions for concurrent smart contracts are applicable to public programs — supporting concurrent updates for confidential policies is equally an issue yet to be addressed for PCDC. However, several approaches have shown the viability of concurrent execution for confidential smart contracts through atomic delivery of computation outputs and state updates [74], pre-ordering the computation before execution [126], and rejecting any state update based on a stale view of the ledger [102]. For PCDC, concurrently tracking a privacy budget is difficult, because a decrease of the privacy budget for multiple computations is equivalent to the addition of privacy budgets consumed in individual computations [127], [128]. Achieving concurrent and replay-attack resistant policy updates of confidential policies and privacy budgets is an open problem.

***Challenge* — Tracking privacy budgets for dataset composition.** Orthogonally to executing multiple programs on a single input, one may intend to execute a single program with more than one input. The latter may be especially relevant for *decentralized data science*. It remains an open problem how to coordinate and quantify the privacy loss for individual datasets that are merged and consumed by a computation with multiple inputs. An additional complexity is presented when datasets are frequently updated, and computation outputs are further reused in a secondary step of the computation. Addressing this complexity by quantifying the privacy loss and gain of complex setups remains equally relevant in practice.

***Remark* — PCDC enables provable data lineage.** PCDC achieves an auditable history of applied computations on a specific data item through on-chain commitments. The immutable property of blockchains, and data-access restrictions by PCDC, facilitates data lineage — a problem that is hard to solve beyond isolated environments [129]. Data lineage tools track the data lifecycle, including its origin, destination and each successive transformation in-between. Verifiable data lineage can help in solving key problems pertaining to data-sovereignty. By identifying data that is most critical to the end-result of a multi-step computation, data consumers can determine individual contributions and incentivize accordingly [130]. Data Provenance, which proves the origin of data, and presents a sub-problem of data lineage, has received notable attention in recent years [56], [57], [131], [132]. Using PCDC to achieve verifiable data lineage is sparsely explored and yields an interesting problem for research.

***Use-Case* — Decentralized Data Science.** The abundance of data has fueled the emergence of artificial intelligence and machine learning algorithms with practical applications. To enhance data availability, various protocols propose *decentralized data marketplaces*, where datasets can be published, discovered and consumed [122]. However, they typically trade raw data and user control takes a secondary role due to data non-rivalry. PCDC serves as an additional layer of governance, where users specify policies to restrict the conducted analyses. This way, PCDC approaches a responsible data economy, where

users can monetize their data without surrendering privacy. Yet, there are many open challenges to be addressed in reaching a practical implementation. It is yet to be addressed how users can control data processing for data in a DWN (cf. § 3.1), e.g., by whitelisting programs.

***Use-Case* — Data Co-Ownership.** Many use-cases are already sufficiently covered through PCDC with a single user that governs a single dataset. However, consolidating datasets of multiple users can increase the utility as compared to each individual dataset. Solving this problem demands for governance solutions for data co-ownership. Users, which merge their data for enhanced co-utility, may govern program executions through a *Data DAO* (Decentralized Autonomous Organization). PCDC can enforce decisions through a shared policy, that is governed by the DAO. Developing governance structures by applying multi secret-sharing schemes [133], and determining the shared value of datasets are equally open research areas.

6. Discussion

Comparison — Strengths, weaknesses and connections. Table 1 identifies the key functionalities provided by each solution and indicates those that are missing. It is clearly shown that industry has put a particular focus on technologies for decentralized identity. This is natural, as, in many ways, decentralized identity is a return to the internet's roots, favoring decentralization over siloed web services. Further, we observe that recent work on decentralized anonymous credentials primarily relies on SNARKs for efficient on-chain verification [50], [54], [58]. Meanwhile, decentralized access control and PCDC have not received widespread attention. We expect that recent improvements for SNARKs can further improve the practicality of applications for data sovereignty. We encourage further work in this space and hope this analysis clarifies missing features and open research areas.

Transparency Technologies. Transparency technologies offer an alternative solution to reduce trust in service providers by requiring them to append requests to an append-only log, such that entities are held accountable upon misbehavior. Transparency logs have been applied to many problem areas in the past, such as for key transparency [134] or certificate transparency [135]. Similarly, transparency logs can augment technologies for data sovereignty. A decentralized identifier can be resolved with a key transparency log, credentials can be transparently managed by third parties through credential transparency logs [136], and the committee of access controllers can record access attempts to secret keys with transparency logs [137], [138]. Whereas, a verifiable registry enhances security and privacy, transparency logs can provide enhanced performance by offering a middle ground between Web 2.0 centralization and Web 3.0 decentralization. We encourage further research on transparency logs for data sovereignty.

Data Authenticity and Provenance. Although users may achieve data sovereignty through the previously mentioned technologies, the challenge of proving the authenticity of data remains. Data Consumers care about the validity of the data that they are provided with, whereas users fail to convince data consumers without further attestations. To effectively address this problem, each data item should be accompanied with a *claim* and an independently verifiable *proof*. We find that there are three possible pathways to overcome this challenge — acquiring an attestation from a reputable third party, proving data provenance, or obtaining an attestation directly from the data source. At the time of writing, these approaches can be implemented by issuing credentials [14], using TLS oracles to prove data provenance [52], [56], [57], or using content provenance as e.g., recently proposed in the C2PA standard [139]. We hope to witness a graceful transition from the first to the third solution in the coming years.

7. Concluding Remarks

In summary, we adhere that simultaneously achieving privacy, transparency and efficiency is a challenging task without further trust assumptions. Our findings suggest, that data sovereignty is an important step towards the next generation of the web, where all actions on personal data are controlled by the user. We believe that research on efficiency in privacy preserving technologies, and tooling for interoperability between the sub-areas studied in this work, will allow the community to bridge the gap between platform-centric and user-centric web technologies.

Acknowledgement. We thank Deevashwer Rathee, Sriram Sridhar and Bryan Ford for their valuable comments. This work is partially supported by the Center for Responsible, Decentralized Intelligence at Berkeley (Berkeley RDI), Chainlink Labs, and the Algorand Centres of Excellence programme managed by Algorand Foundation and the Federal Ministry of Education and Research of Germany in the programme of "Souverän. Digital. Vernetzt.". Joint project 6G-life (project number 16KISK002). Canetti was supported by DARPA under Agreement No. HR00112020021.

References

[1] H. Saleem and M. Naveed, "Sok: Anatomy of data breaches." *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 4, pp. 153–174, 2020.

[2] S. of California Department of Justice, "The california consumer privacy act of 2018 (ccpa)," *Department of Justice*, 2018.

[3] "Web3 is self-certifying," https://jaygraber.medium.com/web3-is-self-certifying-9dad77fd8d81.

[4] A. Krotova, A. Mertens, and M. Scheufen, "Open data and data sharing: An economic analysis," IW-Policy Paper, Tech. Rep., 2020.

[5] "Gaia-x: Driver of digital innovation in europe," https://www.data-infrastructure.eu/GAIAX/.

[6] E. G. Weyl, P. Ohlhaver, and V. Buterin, "Decentralized society: Finding web3's soul," *Available at SSRN 4105763*, 2022.

[7] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 104–121.

[8] J. Benet, "Ipfs-content addressed, versioned, p2p file system (draft 3)," *arXiv preprint arXiv:1407.3561*, 2014.

[9] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 553–567.

[10] D. Fett, R. Küsters, and G. Schmitz, "A comprehensive formal security analysis of oauth 2.0," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1204–1215.

[11] E. Y. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague, "Oauth demystified for mobile application developers," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 892–903.

[12] D. Hardt *et al.*, "The oauth 2.0 authorization framework," 2012.

[13] D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant, and M. Sabadello, "Decentralized identifiers (dids) v1. 0 core data model and syntaxes," *W3C First Public Working Draft, https://www.w3.org/TR/did-core/*, 2019.

[14] W. W. W. Consortium *et al.*, "Verifiable credentials data model v1.1," *W3C First Public Working Draft, https://www.w3.org/TR/vc-data-model/?#core-data-model*, 2019.

[15] M. Jones, J. Bradley, and N. Sakimura, "Json web token (jwt)," Tech. Rep., 2015.

[16] "Peer did method," https://identity.foundation/peer-did-method-spec/.

[17] "Polygon did implementation," https://docs.polygon.technology/docs/develop/did-implementation/getting-started/.

[18] "Ceramic network," https://ceramic.network/.

[19] W. W. W. Consortium, "W3C DiD Method registry," https://w3c.github.io/did-spec-registries/#did-methods, 2021.

[20] "Ethr-did library," https://github.com/uport-project/ethr-did.

[21] "Did universal resolver," https://dev.uniresolver.io/.

[22] "Spruce id," https://www.spruceid.com/.

[23] "Web5: An extra decentralized web platform," https://developer.tbd.website/projects/web5/.

[24] "Lukso," https://www.lukso.network/.

[25] "Erc725 v.2," https://github.com/ethereum/EIPs/issues/725.

[26] "Erc735," https://github.com/ethereum/EIPs/issues/735.

[27] "Dif decentralized web node," https://identity.foundation/decentralized-web-node/spec/.

[28] "Erc-4337: Account abstraction using alt mempool," https://eips.ethereum.org/EIPS/eip-4337.

[29] J. Camenisch and A. Lysyanskaya, "Signature schemes and anonymous credentials from bilinear maps," in *Annual international cryptology conference*. Springer, 2004, pp. 56–72.

[30] S.-Y. Tan and T. Groß, "Monipoly—an expressive q-sdh-based anonymous attribute-based credential system," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 498–526.

[31] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2001, pp. 93–118.

[32] L. Hanzlik and D. Slamanig, "With a little help from my friends: Constructing practical anonymous credentials," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2004–2023.

[33] J. Camenisch, M. Drijvers, and J. Hajny, "Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs," in *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, 2016, pp. 123–133.

[34] M. Schaffer and P. Schartner, "Anonymous authentication with optional shared anonymity revocation and linkability," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2006, pp. 206–221.

[35] R. Li, D. Galindo, and Q. Wang, "Auditable credential anonymity revocation based on privacy-preserving smart contracts," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2019, pp. 355–371.

[36] "Espresso systems," https://www.espressosys.com/.

[37] Hyperledger, "Hyperledger Indy," https://www.hyperledger.org/use/hyperledger-indy, 2017.

[38] Evernym, "BBS+ for Verifiable Credentials," https://www.evernym.com/blog/bbs-verifiable-credentials/, 2021.

[39] J. Camenisch and E. Van Herreweghen, "Design and implementation of the idemix anonymous credential system," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 21–30.

[40] F. Baldimtsi and A. Lysyanskaya, "Anonymous credentials light," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, 2013, pp. 1087–1098.

[41] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn, "Malleable signatures: New definitions and delegatable anonymous credentials," in *2014 IEEE 27th Computer Security Foundations Symposium*. IEEE, 2014, pp. 199–213.

[42] G. Fuchsbauer, C. Hanser, and D. Slamanig, "Structure-preserving signatures on equivalence classes and constant-size anonymous credentials," *Journal of Cryptology*, vol. 32, no. 2, pp. 498–546, 2019.

[43] J. Camenisch, M. Dubovitskaya, K. Haralambiev, and M. Kohlweiss, "Composable and modular anonymous credentials: Definitions and practical constructions," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2015, pp. 262–288.

[44] D. Deuber, M. Maffei, G. Malavolta, M. Rabkin, D. Schröder, and M. Simkin, "Functional credentials." *Proc. Priv. Enhancing Technol.*, vol. 2018, no. 2, pp. 64–84, 2018.

[45] O. Sanders, "Efficient redactable signature and application to anonymous credentials." in *IACR International Conference on Public-Key Cryptography*. Springer, 2020, pp. 628–656.

[46] Hyperledger, "Hyperledger Indy specification Anonymous Credentials," https://github.com/hyperledger-archives/indy-crypto/blob/master/libindy-crypto/docs/AnonCred.pdf, 2018.

[47] ——, "Hyperledger specification Anonymous Credentials 2.0," https://lists.hyperledger.org/g/ursa/topic/30142553, 2020.

[48] A. Sonnino, M. Al-Bassam, S. Bano, S. Meiklejohn, and G. Danezis, "Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers," in *26th Annual Network and Distributed System Security Symposium (NDSS)*, 2019.

[49] M. H. Au, W. Susilo, and Y. Mu, "Constant-size dynamic k-taa," in *International conference on security and cryptography for networks*. Springer, 2006, pp. 111–125.

[50] M. Rosenberg, J. White, C. Garman, and I. Miers, "zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023.

[51] J. Doerner, Y. Kondi, E. Lee, L. Tyner *et al.*, "Threshold bbs+ signatures for distributed anonymous credential issuance," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2022, pp. 2095–2111.

[52] D. Maram, H. Malvai, F. Zhang, N. Jean-Louis, A. Frolov, T. Kell, T. Lobban, C. Moy, A. Juels, and A. Miller, "Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1348–1366.

[53] "Veramo," https://veramo.io/.

[54] D. Rathee, G. V. Policharla, T. Xie, R. Cottone, and D. Song, "Zebra: Anonymous credentials with practical on-chain verification and applications to kyc in defi," *Cryptology ePrint Archive*, 2022.

[55] "Get protocol," https://www.get-protocol.io/.

[56] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 270–282.

[57] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, "Deco: Liberating web data using decentralized oracles for tls," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1919–1938.

[58] M. Chase, E. Ghosh, S. Setty, and D. Buchner, "Zero-knowledge credentials with deferred revocation checks," Tech. Rep, Tech. Rep., 2020.

[59] "iden3.io," https://iden3.io/.

[60] "polygon id," https://polygon.technology/polygon-id/.

[61] B. Ford, "Identity and personhood in digital democracy: Evaluating inclusion, equality, security, and privacy in pseudonym parties and other proofs of personhood," *arXiv preprint arXiv:2011.02412*, 2020.

[62] "Proposal for a regulation of the european parliament and of the council on information accompanying transfers of funds and certain crypto-assets," https://data.consilium.europa.eu/doc/document/ST-10290-2021-INIT/en/pdf.

[63] H. Shafagh, L. Burkhalter, S. Ratnasamy, and A. Hithnawi, "Droplet: Decentralized authorization and access control for encrypted data streams," in *29th USENIX Security Symposium*.

[64] "User managed access," https://kantarainitiative.org/confluence/display/LC/User+Managed+Access.

[65] M. P. Andersen, S. Kumar, M. AbdelBaky, G. Fierro, J. Kolb, H.-S. Kim, D. E. Culler, and R. A. Popa, "{WAVE}: A decentralized authorization framework with transitive delegation," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 1375–1392.

[66] E. Kokoris-Kogias, E. C. Alp, L. Gasser, P. Jovanovic, E. Syta, and B. Ford, "Calypso: Private data management for decentralized ledgers," *Proceedings of the VLDB Endowment*, vol. 14, no. 4, pp. 586–599, 2020.

[67] "Kepler data storage," https://docs.kepler.xyz/.

[68] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-frequency trading on decentralized on-chain exchanges," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 428–445.

[69] S. Rouhani and R. Deters, "Blockchain based access control systems: State of the art and challenges," in *IEEE/WIC/ACM International Conference on Web Intelligence*, 2019, pp. 423–428.

[70] S. Steffen, B. Bichsel, M. Gersbach, N. Melchior, P. Tsankov, and M. Vechev, "zkay: Specifying and enforcing data privacy in smart contracts," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1759–1776.

[71] S. Steffen, B. Bichsel, R. Baumgartner, and M. Vechev, "Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2022, pp. 1543–1543.

[72] "Spruceid cryptoscript," https://github.com/spruceid/cryptoscript.

[73] S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song, "Churp: dynamic-committee proactive secret sharing," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2369–2386.

[74] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 185–200.

[75] V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song, "Storing and retrieving secrets on a blockchain," in *IACR International Conference on Public-Key Cryptography*. Springer, 2022, pp. 252–282.

[76] C. Gentry, S. Halevi, and V. Lyubashevsky, "Practical non-interactive publicly verifiable secret sharing with thousands of parties," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022, pp. 458–487.

[77] F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin, "Can a public blockchain keep a secret?" in *Theory of Cryptography Conference*. Springer, 2020, pp. 260–290.

[78] S. Coull, M. Green, and S. Hohenberger, "Controlling access to an oblivious database using stateful anonymous credentials," in *International Workshop on Public Key Cryptography*. Springer, 2009, pp. 501–520.

[79] J. Camenisch, M. Dubovitskaya, and G. Neven, "Oblivious transfer with access control," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 131–140.

[80] P. Kumaraguru, L. Cranor, J. Lobo, and S. Calo, "A survey of privacy policy languages," in *Workshop on Usable IT Security Management (USM 07): Proceedings of the 3rd Symposium on Usable Privacy and Security, ACM*, 2007.

[81] D. D. F. Maesa, P. Mori, and L. Ricci, "Blockchain based access control services," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 1379–1386.

[82] ——, "A blockchain based approach for the definition of auditable access control systems," *Computers & Security*, vol. 84, pp. 93–119, 2019.

[83] G. Zyskind, O. Nathan *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *2015 IEEE Security and Privacy Workshops*. IEEE, 2015, pp. 180–184.

[84] G. Zyskind, O. Nathan, and A. Pentland, "Enigma: Decentralized computation platform with guaranteed privacy," *arXiv preprint arXiv:1506.03471*, 2015.

[85] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy, "Towards blockchain-based auditable storage and sharing of iot data," in *Proceedings of the 2017 on cloud computing security workshop*, 2017, pp. 45–50.

[86] M. Stadler, "Publicly verifiable secret sharing," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1996, pp. 190–199.

[87] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[88] T. Yurek, Z. Xiang, Y. Xia, and A. Miller, "Long live the honey badger: Robust asynchronous dpss and its applications," *Cryptology ePrint Archive*, 2022.

[89] J. Isaak and M. J. Hanna, "User data privacy: Facebook, cambridge analytica, and privacy protection," *Computer*, vol. 51, no. 8, pp. 56–59, 2018.

[90] C. I. Jones and C. Tonetti, "Nonrivalry and the economics of data," *American Economic Review*, vol. 110, no. 9, pp. 2819–58, 2020.

[91] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 423–443.

[92] R. Solomon and G. Almashaqbeh, "smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption." *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 133, 2021.

[93] T. Kerber, A. Kiayias, and M. Kohlweiss, "Kachina–foundations of private smart contracts," in *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. IEEE, 2021, pp. 1–16.

[94] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "Zexe: Enabling decentralized private computation," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 947–964.

[95] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller, "Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 887–903.

[96] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*. Springer, 2008, pp. 1–19.

[97] M. Bowman, A. Miele, M. Steiner, and B. Vavala, "Private data objects: an overview," *arXiv preprint arXiv:1807.05686*, 2018.

[98] L. Wang, U. Khan, J. Near, Q. Pang, J. Subramanian, N. Somani, P. Gao, A. Low, and D. Song, "Privguard: Privacy regulation compliance made easier," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[99] H. Bitar and B. Jakobsson, "Gdpr: Securing personal data in compliance with new eu-regulations," 2017.

[100] A. Narayan, A. Feldman, A. Papadimitriou, and A. Haeberlen, "Verifiable differential privacy," in *Proceedings of the Tenth European Conference on Computer Systems*, 2015, pp. 1–14.

[101] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE symposium on security and privacy (SP)*.

[102] R. Sinha, S. Gaddam, and R. Kumaresan, "Luciditee: A tee-blockchain system for policy-compliant multiparty computation with fairness," *Cryptology ePrint Archive*, 2019.

[103] G. Kaptchuk, I. Miers, and M. Green, "Giving state to the stateless: Augmenting trustworthy computation with ledgers," *Cryptology ePrint Archive*, 2017.

[104] "Hyperledger private data objects," https://github.com/hyperledger-labs/private-data-objects,.

[105] C. Cordi, M. P. Frank, K. Gabert, C. Helinski, R. C. Kao, V. Kolesnikov, A. Ladha, and N. Pattengale, "Auditable, available and resilient private computation on the blockchain via mpc," in *Cyber Security, Cryptology, and Machine Learning: 6th International Symposium, CSCML 2022, Be'er Sheva, Israel, June 30–July 1, 2022, Proceedings*. Springer, 2022, pp. 281–299.

[106] D. Demirag and J. Clark, "Absentia: Secure multiparty computation on ethereum," in *Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25*. Springer, 2021, pp. 381–396.

[107] C. Baum, J. H.-y. Chiang, B. David, and T. K. Frederiksen, "Eagle: Efficient privacy preserving smart contracts," *Cryptology ePrint Archive*, 2022.

[108] S. Steffen, B. Bichsel, and M. Vechev, "Zapper: Smart contracts with data and identity privacy," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2735–2749.

[109] G. Almashaqbeh and R. Solomon, "Sok: Privacy-preserving computing in the blockchain era," in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2022, pp. 124–139.

[110] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi, "{FastKitten}: Practical smart contracts on bitcoin," in *28th USENIX Security Symposium*.

[111] C. Garman, M. Green, and I. Miers, "Decentralized anonymous credentials," *Cryptology ePrint Archive*, 2013.

[112] "Sign-in with ethereum," https://login.xyz/.

[113] "Mattr global," https://mattr.global/.

[114] "trinsic.id," https://trinsic.id/.

[115] "Hyperledger aries," https://github.com/hyperledger/aries.

[116] "Sismo," https://www.sismo.io/.

[117] "Semaphoreid," https://semaphore.appliedzkp.org/.

[118] "Verite," https://docs.centre.io/verite/.

[119] "Gitcoin passport," https://passport.gitcoin.co/.

[120] "Serto," https://www.serto.id/.

[121] "Pad protocol," https://www.padprotocol.org/.

[122] "Ocean protocol," https://oceanprotocol.com/.

[123] "Oasis labs," https://www.oasislabs.com/.

[124] "iexec," https://iex.ec/.

[125] J. M. Puaschunder, "Data fiduciary in order to alleviate principal–agent problems in the artificial big data age," in *Information for Efficient Decision Making: Big Data, Blockchain and Relevance*, 2021.

[126] K. Wüst, S. Matetic, S. Egli, K. Kostiainen, and S. Capkun, "Ace: asynchronous and concurrent execution of complex smart contracts," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 587–600.

[127] N. Hynes, D. Dao, D. Yan, R. Cheng, and D. Song, "A demonstration of sterling: a privacy-preserving data marketplace," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 2086–2089, 2018.

[128] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler, "Gupt: privacy preserving data analysis made easy," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*.

[129] P. McDaniel, "Data provenance and security," vol. 9, 2011, pp. 83–85.

[130] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. Hynes, N. M. Gürel, B. Li, C. Zhang, D. Song, and C. J. Spanos, "Towards efficient data valuation based on the shapley value," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1167–1176.

[131] S. Capkun, E. Ozturk, G. Tsudik, and K. Wüst, "Rosen: Robust and selective non-repudiation (for tls)," in *Proceedings of the 2021 on Cloud Computing Security Workshop*, 2021, pp. 97–109.

[132] H. Ritzdorf, K. Wüst, A. Gervais, G. Felley, and S. Capkun, "Tls-n: Non-repudiation over tls enabling-ubiquitous content signing for disintermediation," in *25th Annual Network and Distributed System Security Symposium (NDSS)*, 2018.

[133] C. Blundo, A. D. Santis, G. D. Crescenzo, A. G. Gaggia, and U. Vaccaro, "Multi-secret sharing schemes," in *Annual International Cryptology Conference*. Springer, 1994, pp. 150–163.

[134] M. Chase, A. Deshpande, E. Ghosh, and H. Malvai, "Seemless: Secure end-to-end encrypted messaging with less trust," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 1639–1656.

[135] B. Laurie, "Certificate transparency," *Communications of the ACM*, vol. 57, no. 10, pp. 40–46, 2014.

[136] M. Chase, G. Fuchsbauer, E. Ghosh, and A. Plouviez, "Credential transparency system," in *Security and Cryptography for Networks: 13th International Conference, SCN 2022, Amalfi (SA), Italy, September 12–14, 2022, Proceedings*. Springer, 2022, pp. 313–335.

[137] "Privacy preserving accountable decryption - pad tech," https://www.padprotocol.org/.

[138] M. Chase, H. Davis, E. Ghosh, and K. Laine, "Acsesor: A new framework for auditable custodial secret storage and recovery," *Cryptology ePrint Archive*, 2022.

[139] L. Rosenthol, "C2pa: the world's first industry standard for content provenance," in *Applications of Digital Image Processing XLV*, vol. 12226. SPIE, 2022, p. 122260P.

[140] "did:web method specification," https://w3c-ccg.github.io/did-method-web/.

[141] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," Cryptology ePrint Archive, 2000.

[142] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," *IACR Cryptology ePrint Archive*, vol. 2011, 01 2011.

[143] J. Brorsson, B. David, L. Gentile, E. Pagnin, and P. S. Wagner, "Papr: Publicly auditable privacy revocation for anonymous credentials," *Cryptology ePrint Archive*, 2023.

Appendix A. Inclusion & Exclusion Criteria

We compare the functionality supported by *(i)* research papers and *(ii)* industry solutions in Table 1. To systematically collect the body of relevant works, we only include works based on the following criteria:

- **Inclusion** — The tool/research clearly intends to solve either a problem concerning *(i)* decentralized identity, *(ii)* decentralized access control or *(iii)* PCDC.
- **Inclusion** — The tool leverages techniques from one of the sub-areas (e.g., decentralized identity) to solve problems in another (e.g., decentralized access control).
- **Exclusion** — The tool does not provide sufficient documentation and supporting information to prove that it fulfills a claimed functionality.
- **Exclusion** — The research is not peer-reviewed at the time of writing.

We further provide an explanation of Table 1, which provides an overview of the functionalities supported by different solutions in academia and industry. We evaluate Table 1 with regard to the functionality of systems as introduced in the formal definitions in Sections 3 — 5. We aim to objectively analyze the support of different solutions to the best of our knowledge, without favoring one scheme over another. For decentralized identity management, we note that if a paper or system implicitly relies on a decentralized identifier infrastructure (e.g., CanDID [52]), we say it fulfills the properties for decentralized identity management with respect to the referenced concrete implementation. Certain systems [48], [50] do not explicitly mention the reliance on an infrastructure for decentralized identity management and solely rely on pseudonymous public keys for their proposed construction. SpruceID [22] provides a library for decentralized identity management and verifiable credentials. Whether an implementation relying on SpruceID supports all functionalities for decentralized identity management depends on the DID method (e.g., did-web [140]).

For decentralized anonymous credentials, some systems focus on credentials without explicitly supporting anonymity. We denote this in Table 1 with ◐. Further, certain systems provide libraries for issuance, proving and verification of credentials. We say that they provide the "Request" functionality if they provide tools for agent to agent communication. Note that Semaphore [117] is a signaling protocol, which does not issue traditional credentials. An identity in Semaphore is a commitment, and users can prove membership of certain groups, which are represented as binary Merkle Trees. In this case, we refer to issuance as adding a member to the group and revocation as deleting a member to the group.

For decentralized access control, some systems rely on trusted hardware rather than a secret-sharing approach [123], [124]. In this case, we say that "Setup Shares" and "Reconstruct" does not apply. The PAD protocol presents a solution for decentralized access control that uses a transparent log to record access attempts [121]. They do not have a direct notion of a policy, access is delegated by explicitly sending an access token. Note that while Ocean Protocol [122] does support computation on user data, technologies for secure computing are seen as orthogonal and not explicitly supported.

Appendix B. Supplementary Formalism

In this section, we formally present the properties and security goals of the definitions as introduced in previous sections for decentralized identity, decentralized access control, and policy-compliant computation, in the ideal-world/real-world paradigm (UC model [141]).The UC model is widely recognized as the gold standard for establishing security in various cryptographic applications, such as for example MPC and ZKPs, providing a unified, "ideal", description that satisfies all desired properties. Successively, a protocol designer can prove that a concrete instantiation of a protocol "emulates" this ideal description, such that all security flaws in the concrete instantiation would also be possible in the ideal description (i.e., the ideal functionality). At the time of writing, definitions for protocols as described in Sections 3, 4 and 5 exist in various different formats. In the following, we provide a set of ideal functionalities that fulfill the properties we deem essential for the previous definitions. The functionalities presented are intended to serve as templates for the formalization of data sovereign systems. It should be noted that specific implementations may incorporate additional restrictions or variations in the manner in which identities and data are managed and controlled. Furthermore, the conditions under which the security properties of these systems may be compromised can vary.

B.1. Ideal Functionalities

An ideal functionality represents an ideal protocol that carries out the desired task. All parties send their inputs to the ideal functionality. The ideal functionality computes the output and returns the output to all parties. The security of a protocol ϕ is examined by comparing the view of an environment Z when ϕ is executed in the real world against that of Z in the ideal world defined in F_ϕ. In both worlds, the environment Z chooses the inputs of the parties and collects their outputs. In the ideal world, Z interacts with dummy parties, that send messages between Z, ideal functionality F_ϕ, and a simulator. In the real world, the environment Z interacts freely with any adversary, who can corrupt any party. We say a protocol is secure (UC realizes the ideal functionality) if the environment Z can't distinguish between the real world and ideal world executions.

In the UC model [141], parties, functionalities, the adversary, and the environment are (instances of) interactive Turing machines (ITMs) running in probabilistic polynomial time (PPT). The adversary controls the scheduling of the delivery of all messages exchanged between parties. To start an execution of the protocol ϕ, the environment initiates a protocol execution session, identified by a session identifier sid , and activates all session participants. The environment Z invokes all parties and assigns a unique identifier to each.

In this paper, we consider a network that is partially synchronous, meaning that the adversary delivers messages within a fixed bounded delay. This partial synchronous network can be effectively represented by an ideal functionality known as Eventual-Delivery Secure Message Transmission Functionality $\mathcal{F}_{ED-SMT}$ as de-

scribed in [142]. However, explaining the integration of this functionality is not within the scope of this paper, as we aim to represent ideal functionalities representing relevant properties, rather than introducing a fully-working protocol that UC-realizes the proposed functionalities.

B.2. Dec. Identity Management Ideal Functionality ($\mathcal{F}_{DIM}$)

We express the security goals of the decentralized identity model in terms of an ideal functionality $\mathcal{F}_{DIM}$ defined in Figure 5. We further give an explanation and intuition behind our design. First, the functionality maintains a list of registered parties $\mathcal{L}_{pk}$, a list of identity tuples $\mathcal{L}_{tup}$ and a list of registered identifiers $\mathcal{L}_{reg}$ initially set to empty. Upon initialization, the adversary $\mathcal{A}$ chooses the **GeneratePK**() algorithm. When the functionality receives a "generate" message from a party $\mathcal{P}$, the functionality generates a secret identifier pk using **GeneratePK**(). We assume the generate algorithm supplied by the adversary randomly samples a unique random key for every request to fulfill the uniqueness property. Once the functionality receives the public key, it records it in list $\mathcal{L}_{pk}$ and returns the pk to the party. $\mathcal{F}_{DIM}$ successively does not check the "system identity" of the caller, but rather identifies the calling party by pk. Once the functionality receives a "create" message from a party, it checks if the party is already registered in $\mathcal{L}_{pk}$. If it is not, it ignores the request. Else, it executes **GenerateDID**() to sample an identifier given a pk. The fact that $\mathcal{F}_{DIM}$ returns the DID immediately to the user implies that the implementation of $\mathcal{F}_{DIM}$ needs to process the query locally. Note, that this detail is implementation dependent and varies with respect to the DID method. In contrast, not returning the output immediately, but rather letting the user query for the output in another call, implies that an implementation can be distributed, but would need to make sure that the information is always available for inspection. In "generate" and "create", parties can call them an unlimited amount of times. To retrieve metadata MD_{DID} for some pseudonymous identifier, any party can send "resolve" to $\mathcal{F}_{DIM}$. $\mathcal{F}_{DIM}$ returns the MD_{DID} if the identifier and metadata exists. A party can "update" its MD_{DID} by sending the a requests to the functionality, modifies the lists maintained accordingly. We model the corruption of $\mathcal{F}_{DIM}$ through a set of constituent identities, which represent the identities that realize the functionalities services — in this case the verifiable registry modeled as an internal, append-only list. If the set of corrupted constituent identities is sufficiently large, $\mathcal{F}_{DIM}$ gets corrupted and returns random strings when queried for entries in $\mathcal{L}_{tup}$.

The ideal functionality encompasses the following properties from Sec. 3:

- **Uniqueness:** Each user receives a unique identifier as it is assumed to be generated using a random sampling technique with negligible collision probability.
- **Persistence:** The user is the one in control of generating, creating and deactivating any of its identity tuples, and it will continue to exist unless the user deactivates it. The $\mathcal{F}_{DIM}$ checks if any of the "create", "update", and "deactivate" comes from a user with the correct associated public key before honoring the request.
- **Pseudonymity:** There is no restriction on how many times a user can call "create" to generate pseudonymous identifiers. By enabling the creation of more than one identifier, it impedes linkability while allowing entities to present themselves differently in different contexts. Protecting against sybil attack is out of scope for this work.

$\mathcal{F}_{DIM}$ (κ, L) **with adversary** $\mathcal{A}$

The functionality is parameterized by a list L of *constituent identities*. (These are the identities of the parties that realize the functionality's services). It also maintains lists $\mathcal{L}_{pk}, \mathcal{L}_{reg}, \mathcal{L}_{tup}$, Initially set to $\emptyset$, and set corruption state to uncorrupted.

// Initialization

On receive ("init"):
  send "init" to $\mathcal{A}$, obtain **GeneratePK**(), **GenerateDID**() from $\mathcal{A}$

// Generate secret identifier

On receive ("generate") from $\mathcal{P}$:
  notify $\mathcal{A}$ of ("generate")
  sample pk = **GeneratePK**() and add (pk) to $\mathcal{L}_{pk}$.
  (If pk already appears in $\mathcal{L}_{pk}$, set pk to be the first unused string.)
  output ("generated", pk) to $\mathcal{P}$

// Create a new public identifier DID

On receive ("create", pk) from some party:
  check if $pk \in \mathcal{L}_{pk}$, else return $\perp$ to the caller
  notify $\mathcal{A}$ of ("create")
  sample DID = **GenerateDID**() and add (DID) to $\mathcal{L}_{reg}[pk]$
  (If DID already appeared, set DID to be the first unused string.)
  set $\text{MD}_{\text{DID}} := \emptyset$; $\mathcal{L}_{tup}[\text{DID}] := (\text{MD}_{\text{DID}})$
  send (DID) to the caller

// Resolve an identifier to its metadata

On receive ("resolve", DID) from some party:
  check that $\mathcal{L}_{tup}[\text{DID}] \neq \emptyset$, else return $\perp$
  notify $\mathcal{A}$ of ("resolve", DID)
  set $(\text{MD}_{\text{DID}}) := \mathcal{L}_{tup}[\text{DID}]$
  output (MD_{DID}) to the caller

// Update the associated metadata

On receive ("update", pk, MD_{DID}) from some party:
  if $pk \notin \mathcal{L}_{pk}$ or $\mathcal{L}_{tup}[\text{DID}] = \emptyset$, return $\perp$
  notify $\mathcal{A}$ of ("update", MD_{DID})
  set $\mathcal{L}_{tup}[\text{DID}] := (\text{MD}_{\text{DID}})$

On receive ("corrupted") from $\mathcal{P}$:
  If $\mathcal{P}$ is in the set L of constituents
    add $\mathcal{P}$ to the set C of corrupted constituents
  If the set C is [sufficiently large]
    set state to corrupted and return an element randomly sampled
    from $\{0,1\}^{\kappa}$ when being queried for $\mathcal{L}_{tup}$.

Figure 5: Ideal functionality for Dec. Identity Management $\mathcal{F}_{DIM}$

B.3. Decentralized Anonymous Credentials Ideal Functionality ($\mathcal{F}_{DAC}$)

Likewise, we express the security goals of the decentralized anonymous credentials in terms of an ideal functionality $\mathcal{F}_{DAC}$ defined in Figure 6.We assume that each party is registered with $\mathcal{F}_{DIM}$ and has an identifier DID. We build upon previous UC-functionalities that propose the issuance of credentials and proofs thereof. We extend the proposed approaches to align with the previously introduced functionality for decentralized identity management, which provides a functionality to manage pseudonyms in a decentralized fashion. The **Validate**() algorithm is provided as a parameter to the functionality. First, the functionality is initialized by the issuer. When

initialized, the functionality asks the adversary to provide the algorithms necessary, **Prove**() and **GenerateCred**(), which are PPT ITM. Further, the list of requests, the list of issued credentials and the list of presented credentials is initialized. The user creates a credential request by providing a set of claims c, and some auxiliary data needed to verify the correctness of the claims. Note, that we model the issuance of a credential as a single session where the functionality is parametrized by a single issuer, such that for multiple issuers multiple sessions of $\mathcal{F}_{DAC}$ need to be run in parallel. Once the functionality checked that the user has previously invoked the "request" subroutine, it proceeds to inform the adversary, validates the correctness of claims by invoking **Validate**(), generates the credential cred and adds the relevant values for issuance in the list of issued credentials $\mathcal{L}_{\text{cred}}$. Thereafter, $\mathcal{F}_{DAC}$ sends the issued credential to the user. To prove the credential, the functionality takes the credential, a predicate specifying the predicates to prove, and the pseudonymous identifier as an input from the user. It checks that the credential has indeed been issued and invokes **Prove**() to generate a proof that is handed back to the user. Note that in this step no private or linkable information is associated with proving the credential. Lastly, upon receiving "verify", the functionality checks that the presented proof and predicate coincide with the previously issued tuple. Note that this ensure that 1) no private information is disclosed to the verifier and 2) that no arbitrary user can present a valid proof for a credential that doesn't belong to it. To revoke a credential, an issuer of a specific credential can remove the tuple from the list of issued credentials. Note, that Brorsson *et. al* [143] concurrently proposed an ideal functionality for (non-decentralized) anonymous credentials that further supports anonymity revocation.

In detail, $\mathcal{F}_{DAC}$ encompasses the following properties from Sections 3:

- **Unforgeability:** $\mathcal{F}_{DAC}$ only returns "verified" in "verify" if a proof π_{cred} has previously been issued. The proof π_{cred} will only be granted if the credential satisfies ϕ_{cred}.
- **Anonymity:** The functionality relies on pseudonymous identifiers. $\mathcal{F}_{DAC}$ uses pseudonyms instead of the user's identity $\mathcal{U}$ when forwarding user's requests to issuers or verifiers.
- **Predicate Provability**: $\mathcal{F}_{DAC}$ supports predicate provability by building a valid proof π for the presentation statement. For predicate provability, **Prove**() constructs the proof from the credential.
- **Credentials Revocation:** $\mathcal{F}_{DAC}$ supports credential revocation by removing a tuple from the list of issued credentials.On receiving "revoke" by the issuer, $\mathcal{F}_{DAC}$ removes the credentials from $\mathcal{L}_{cred}$.
- **Non-Transferability:** $\mathcal{F}_{DAC}$ supports non-transferability by associating each credential with the the identity $\mathcal{U}$ of the requesting user. Hence, the credential is bound to a single holder.
- **Unlinkability:** $\mathcal{F}_{DAC}$ ensures unlinkability as the user creates a pseudonym $\text{DID}_{\mathcal{U}}{}'$ to generate proofs with respect to the pseudonym.

$\mathcal{F}_{DAC}$ $(\kappa, \mathcal{I})$ **with adversary** $\mathcal{A}$

The functionality is parameterized by an algorithm **Validate**() which validates claims provided in a credential request. It also maintains lists $\mathcal{L}_{\text{req}}, \mathcal{L}_{\text{cred}}, \mathcal{L}_{\text{show}}$, initially set to $\emptyset$, and a corruption state initially set to uncorrupted.

// Initialization by the issuer $\mathcal{I}$
On receive ("init") from $\mathcal{I}$:
  send ("init", $\mathcal{I}$) to $\mathcal{A}$;
  wait for $\mathcal{A}$ to respond with ("init", **GenerateCred**(), **Prove**())
  store the returned algorithms

// Request a credential
On receive ("request", c, $\text{DID}_{\mathcal{U}}$, $aux(\mathcal{U})$, $\text{DID}_{\mathcal{I}}$) from $\mathcal{U}$:
  set $\pi_{\text{req}} := aux(\mathcal{U}) \,\|\, \text{DID}_{\mathcal{I}}$, and add π_{req} to $\mathcal{L}_{\text{req}}$
  send $(c, \pi_{\text{req}}, \text{DID}_{\mathcal{U}})$ to $\mathcal{U}$

// Issue the credential
On receive ("issue", c, π_{req}, $\text{DID}_{\mathcal{U}}$) from $\mathcal{I}$:
  and check that $\pi_{\text{req}} \in \mathcal{L}_{\text{req}}$, else return $\perp$
  notify $\mathcal{A}$ of ("issue", c, π_{req}, $\text{DID}_{\mathcal{U}}$)
  check **Validate**$(c, \pi_{\text{req}}) = 1$, else return $\perp$
  set cred := **GenerateCred**()
  add $(\mathcal{U}, \text{cred})$ to $\mathcal{L}_{\text{cred}}$
  send ("success", cred) to $\mathcal{I}$

// Prove the credential
On receive ("prove", cred, ϕ_{cred}, $\text{DID}_{\mathcal{U}}{}'$) from $\mathcal{U}$:
  notify $\mathcal{A}$ of ("prove", cred, ϕ_{cred}, $\text{DID}_{\mathcal{U}}{}'$)
  check that $(\mathcal{U}, \text{cred}) \in \mathcal{L}_{\text{cred}}$, else return $\perp$
  set π_{cred} := **Prove**(cred, ϕ_{cred}, $\text{DID}_{\mathcal{U}}{}'$)
  add $(\pi_{\text{cred}}, \phi_{\text{cred}})$ to $\mathcal{L}_{\text{show}}[\text{DID}_{\mathcal{U}}{}']$
  send (π_{cred}) to $\mathcal{U}$

// Verify a presentation
On receive ("verify", π_{cred}, ϕ_{cred}, $\text{DID}_{\mathcal{U}}{}'$) from $\mathcal{V}$:
  set $(\pi'_{\text{cred}}, \phi'_{\text{cred}}) := \mathcal{L}_{\text{show}}[\text{DID}_{\mathcal{U}}{}']$
  check that $\pi_{\text{cred}} = \pi'_{\text{cred}}$ and $\phi_{\text{cred}} = \phi'_{\text{cred}}$
  send ("verified") to $\mathcal{V}$

// Revoke a credential
On receive ("revoke", cred, $\text{DID}_{\mathcal{U}}$) from $\mathcal{I}$:
  check that $(\mathcal{U}, \text{cred}) \in \mathcal{L}_{\text{cred}}$, else return $\perp$
  notify $\mathcal{A}$ of ("revoke", cred, $\text{DID}_{\mathcal{U}}$)
  remove $(\mathcal{U}, \text{cred})$ from $\mathcal{L}_{\text{cred}}$
  send ("revoked", cred, $\text{DID}_{\mathcal{U}}$) to $\mathcal{I}$

On receive ("corrupted") from $\mathcal{I}$:
  set state to corrupted and return an element randomly sampled from $\{0,1\}^{\kappa}$ when queried for cred with "issue".

Figure 6: Ideal Functionality for Decentralized Anonymous Credentials $\mathcal{F}_{DAC}$.

B.4. Decentralized Access Control Ideal Functionality ($\mathcal{F}_{access}$)

Decentralized access control allows for the user-controlled authorization of a set of data consumers and ensures automated access to the resource in question. We detail the functionality $\mathcal{F}_{access}$, which describes the above scenario for a single resource, protected by a user-specified policy, in the following. The functionality maintains a list of authenticated data consumers $\mathcal{L}_{auth}$, a list of shares $\mathcal{L}_{share}$ and a list of policies $\mathcal{L}_{pol}$. The functionality behaves fairly simple, in a way that ensures that only authenticated data consumers can access a data item, whilst accounting for the corruption of a subset of access controllers. First, upon receiving a request to "deal" the shares for a certain key, $\mathcal{F}_{access}$ stores the generated key in an internal list indexed by the kid. The functionality further stores the policy p provided by the user. To "authenticate", a data

consumer presents a valid proof, and its pseudonymous identifier is added to the list of authenticated data consumers. Upon receiving "access", $\mathcal{F}_{access}$ checks whether the calling data consumer is authenticated. Hence, only data consumers who previously authenticated can access the secret maintained by the access controllers. We model corruption in a similar sense to $\mathcal{F}_{DIM}$, such that a set of constituent parties (access controllers) maintains the shared key.

The ideal functionality encompasses the following properties:

- **Data Confidentiality:** $\mathcal{F}_{access}$ supports data confidentiality by only allowing authenticated data consumers to access the secret.
- **Anonymity:** Anonymity requires that no adversary should be able to distinguish which consumer $\mathcal{C}$ accessed the data item $d_i^{\mathcal{U}}$, unless requested by an audit. $\mathcal{F}_{access}$ provides pseudonymity, by only returning the pseudonymous identifier upon request of an audit.
- **Auditability:** For auditability, any invocation of "authenticate" and "access" should be publicly observable. $\mathcal{F}_{access}$ supports auditability for authentication by allowing read access to the list of authenticated consumers.
- **Policy Confidentiality:** The policy is maintained by $\mathcal{F}_{access}$. The data consumer only gains knowledge about whether the provided proof satisfies the policy.
- **Fair access:** $\mathcal{F}_{access}$ supports fair access by sending the secret k to data consumers simultaneously, such that all participants are treated equally.
- **Access Revocation:** $\mathcal{F}_{access}$ supports access revocation by allowing $\mathcal{U}$ to call "revoke" to remove the data consumer in question.

B.5. Functionality Wrapper for Policy Compliant Decentralized Computation ($\mathcal{W}_{comp}$)

We formally represent policy-compliant decentralized computation in terms of a functionality wrapper $\mathcal{W}_{comp}$ shown in Figure 8. The wrapper further ensures that data may not be copied and reused after giving access to it, precluding the problem of data nonrivalry. $\mathcal{W}_{comp}$ specifies the following security and privacy guarantees. The data provided in "commit" by $\mathcal{U}$ remains confidential towards $\mathcal{C}$. Likewise, the program Θ provided by $\mathcal{C}$ is inaccessible to $\mathcal{U}$. However, $\mathcal{U}$ can obtain the pseudonymous identifier of $\mathcal{C}$, observing that $\mathcal{C}$ has executed a computation on $d_i^{\mathcal{U}}$. The consumer discloses the program Θ to "analyze" whether it is compliant with a policy p set by $\mathcal{U}$. To ensure the correctness of computations, $\mathcal{W}_{comp}$ only computes on the data, if the program provided has previously been analyzed for the data in question. Note that we model $\mathcal{W}_{comp}$ with a single compute node, forfeiting decentralization for semantic clarity. The functionality additionally updates the residual policy given the execution of the program to model degradation of privacy. A consumer $\mathcal{C}$ can always "claim" the output of a program, given that the program has been analyzed and complies with the policy.

In detail, $\mathcal{W}_{comp}$ works as follows. $\mathcal{W}_{comp}$ is parameterized by access controllers $\mathcal{AC}$, compute nodes $\mathcal{CN}$ and a leakage function l. The functionality maintains a list of policies $\mathcal{L}_{pol}$, a list of user data $\mathcal{L}_{data}$, a list of computation requests $\mathcal{L}_{comp}$, and a list of computation output

$\mathcal{F}_{access}$ (κ,l, $\{\mathcal{AC}_i\}_{i\in[0,n]}$) with adversary $\mathcal{A}$

The functionality is parametrized with a set of n access controllers $\{\mathcal{AC}_i\}_{i\in[0,n]}$
The functionality maintains $\mathcal{L}_{auth}, \mathcal{L}_{share}, \mathcal{L}_{pol}$ initially set to $\emptyset$

// Deal the shares to the access controllers and set the policy
On receive ("deal", p, k) from $\mathcal{U}$:
  set kid $\xleftarrow{\$}$ $\{0,1\}^\kappa$
  notify $\mathcal{A}$ of ("deal", $l(p)$, $l(\mathcal{U})$)
  set $\mathcal{L}_{pol}$[kid] := p
  set $\mathcal{L}_{share}$[kid] := k
  send ("success", kid) to $\mathcal{U}$

// Authenticate some party based on proof π provided
On receive ("authenticate", π, kid, $\text{DID}_\mathcal{C}$) from some party:
  notify $\mathcal{A}$ of ("authenticate", π, $\text{DID}_\mathcal{C}$)
  set $p := \mathcal{L}_{pol}$[kid], check that $p(\pi) = 1$, else return $\perp$
  add ($\text{DID}_\mathcal{C}$) to list $\mathcal{L}_{auth}$[kid]
  send ("success") to the caller

// Obtain access
On receive ("access", kid, $\text{DID}_\mathcal{C}$) from some party:
  check that ($\text{DID}_\mathcal{C}$) $\in \mathcal{L}_{auth}$[kid] and $\mathcal{L}_{share}$[kid] $\neq \emptyset$,
  else return $\perp$
  notify $\mathcal{A}$ of ("access", kid, $\text{DID}_\mathcal{C}$)
  set $k := \mathcal{L}_{share}$[kid]
  send k to the caller

// Allow read to list of auth. consumers for auditability
On receive ("read",kid) from $\mathcal{U}$:
  check that $\mathcal{L}_{auth}$[kid] $\neq \emptyset$, else return $\perp$
  send ($\mathcal{L}_{auth}$[kid]) to $\mathcal{U}$

// Revoke access
On receive ("revoke", kid, $\text{DID}_\mathcal{C}$) from $\mathcal{U}$:
  check that ($\text{DID}_\mathcal{C}$) $\in \mathcal{L}_{auth}$[kid], else return $\perp$
  notify $\mathcal{A}$ of ("revoke", kid, $\text{DID}_\mathcal{C}$)
  remove ($\text{DID}_\mathcal{C}$) from $\mathcal{L}_{auth}$[kid]
  send ("revoked", kid, $\text{DID}_\mathcal{C}$) to $\mathcal{U}$

// Update the policy
On receive ("udpate", kid, p') from $\mathcal{U}$:
  check that $\mathcal{U}$ has called "deal" before
  set $\mathcal{L}_{pol}$[kid] := p'
  notify $\mathcal{A}$ of ("update", kid, p')
  send ("success") to $\mathcal{U}$

On receive ("corrupted") from $\mathcal{AC}_i$:
  add $\mathcal{AC}_i$ to the set C of corrupted parties
  If the set C is [sufficiently large]
    send $\mathcal{L}_{share}$ to $\mathcal{A}$; set state to corrupted and return an element
    sampled from $\{0,1\}^\kappa$ when queried for k with "access".
    When queried for "deal" let $\mathcal{A}$ respond to the caller.

Figure 7: Ideal Functionality for Decentralized Access Control $\mathcal{F}_{access}$.

and correctness proofs $\mathcal{L}_{out}$ that are initially set to empty. $\mathcal{W}_{comp}$ relays messages between $\mathcal{F}_{access}$ and other parties. If $\mathcal{W}_{comp}$ receives requests from a user to commit its data and policy, it relays the message without restriction. Once receiving "commit" from $\mathcal{U}$, the wrapper forwards a "deal" request to $\mathcal{F}_{access}$. Before computing a program on a certain user's data, $\mathcal{C}$ has to prove that the program abides by the user's associated policy. In "analyze", a consumer $\mathcal{C}$ tells the functionality to analyze the program Θ for the data item identified by kid. On receiving "compute" from $\mathcal{CN}$, $\mathcal{W}_{comp}$ only proceeds if the program provided by $\mathcal{CN}$ has previously been analyzed. The wrapper first forwards "access" on behalf of $\mathcal{CN}$ to obtain access to $d_i^{\mathcal{U}}$. If the wrapper receives "success" from $\mathcal{F}_{access}$, it runs the program Θ and generates π_C, which proves the

$\mathcal{W}_{comp}(\mathcal{F})$ $(\kappa, l, \mathcal{CN}, \{\mathcal{AC}_i\}_{i\in[0,n]})$ with adversary $\mathcal{A}$

The functionality is parameterized by an algorithm **GenerateProof**() which outputs a proof π of compliance of the program provided by $\mathcal{C}$. It also maintains lists $\mathcal{L}_{pol}, \mathcal{L}_{data}, \mathcal{L}_{comp}, \mathcal{L}_{out}$ initially set $\emptyset$, and a corruption state initially set to uncorrupted.

// Initialization
On receive ("init") from $\mathcal{P}$:
    send ("init", $\mathcal{P}$) to $\mathcal{A}$
    forward ("init") to $\mathcal{F}$

// Share data and policy with Access Controllers
On receive ("commit", $d_i^{\mathcal{U}}$, p, k) from $\mathcal{U}$
    notify $\mathcal{A}$ of ("commit", $l(d_i^{\mathcal{U}})$, $l(p)$);
    forward ("deal", p, k) to $\mathcal{F}$
    wait to receive ("success", kid) from $\mathcal{F}$
    set $\mathcal{L}_{pol}$[kid] := p and $\mathcal{L}_{data}$[kid] = $d_i^{\mathcal{U}}$
    send ("success", kid) to $\mathcal{U}$

// Analyze if the program is compliant
On receive ("analyze", kid, Θ) from $\mathcal{C}$
    notify $\mathcal{A}$ of ("analyze", $l(\Theta)$,$\mathcal{C}$)
    set $p := \mathcal{L}_{pol}$[kid]
    check π_A := **GenerateProof**(Θ, p), else return $\perp$
    forward ("authenticate", π_A, kid, DID$_{\mathcal{CN}}$) to $\mathcal{F}$
    set $\mathcal{L}_{comp}$[kid] := Θ
    send ("success", kid) to $\mathcal{C}$

// Program execution request on some user's data
On receive ("compute", kid, Θ) from $\mathcal{CN}$
    set $\Theta' := \mathcal{L}_{comp}$[kid], if $\Theta' \neq \Theta$ return $\perp$
    forward ("access", DID$_{\mathcal{CN}}$, kid) to $\mathcal{F}$
    wait to receive (k) from $\mathcal{F}$, else return $\perp$
    set $d_i^{\mathcal{U}} := \mathcal{L}_{data}$[kid]
    compute $c_{out}, \pi_C, p' := \Theta(d_i^{\mathcal{U}})$
    set $\mathcal{L}_{pol}$[kid] := p' and $\mathcal{L}_{out}$[kid] := c_{out}, π_C
    send ("computed") to $\mathcal{CN}$

// Claiming program output
On receive ("claim", kid, Θ) from $\mathcal{C}$:
    check if $\mathcal{L}_{out}$[kid] is not empty, else return $\perp$.
    notify $\mathcal{A}$ of ("claim", $l(c_{out})$, $\mathcal{C}$)
    set $c_{out}, \pi_C := \mathcal{L}_{out}$[kid]
    send ("claimed", c_{out}, π_C) to $\mathcal{C}$

On receive ("corrupted") from $\mathcal{CN}$:
    set state to corrupted, return an element randomly sampled
    from $\{0,1\}^\kappa$ when queried for c_{out} with "claim" and
    disclose k to $\mathcal{A}$ when being instructed to "compute"

// Managing all other requests between parties
Any other request from any participant or the adversary is simply relayed to the underlying functionality without any further action and the output is given to the destination specified by the functionality

Figure 8: Wrapper for Policy-compliant decentralized computation $\mathcal{W}_{comp}$.

correctness of the computation. We emphasize that the functionality doesn't support concurrent execution. We assume that the user's data state gets frozen during a program execution request, preventing the acceptance of any concurrent requests. We model the output claiming by querying methodology. The consumer keeps asking for the computation result until the program is executed, and the output is available. Once the program is executed, the policy is updated. In "claim", the consumer submits the kid and Θ for which the computation result should be claimed. The wrapper then returns the program output and the computation proof.