Computer Networks and Communications
https://ojs.wiserpub.com/index.php/CNC

Article

DeAuth: A Decentralized Authentication and Authorization Scheme for Secure Private Data Sharing

Phillipe Austria*, **Yoohwan Kim**, **Ju-Yeon Jo**

Department of Computer Science, Howard R. Hughes College of Engineering, University of Nevada, Las Vegas, Las Vegas, NV, USA
E-mail: austrp1@unlv.nevada.edu

Received: 15 January 2024; **Revised:** 15 April 2024; **Accepted:** 26 April 2024

Abstract: The sharing of private information is a daunting, multifaceted, and expensive undertaking. Furthermore, identity management is an additional challenge that poses significant technological, operational, and legal obstacles. Present solutions and their accompanying infrastructures rely on centralized models that are susceptible to hacking and can hinder data control by the rightful owner. Consequently, blockchain technology has generated interest in the fields of identity and access control. This technology is viewed as a potential solution due to its ability to offer decentralization, transparency, provenance, security, and privacy benefits. Nevertheless, a completely decentralized and private solution that enables data owners to control their private data has yet to be presented. In this research, we introduce DeAuth, a novel decentralized, authentication and authorization scheme for secure private data transfer. DeAuth combines blockchain, smart-contracts, decentralized identity, and distributed peer-to-peer (P2P) storage to give users more control of their private data, and permissioning power to share without centralized services. For this scheme, identity is proven using decentralized identifiers and verifiable credentials, while authorization to share data is performed using the blockchain. A prototype was developed using the Ethereum Blockchain and the InterPlanetary Files System, a P2P file sharing protocol. We evaluated DeAuth through a use-case study and metrics such as security, performance, and cost. Our findings indicate DeAuth to be viable alternative to using centralized services; however, the underlying technologies are still in its infancies and require more testing before it can supplant traditional services.

Keywords: decentralized identity, access control, blockchain, smart contracts, InterPlanetary File System (IPFS)

1. Introduction

Cloud services have become an increasingly popular option for personal users and businesses looking to store, manage, and process data online. By leveraging the power of the internet and remote servers, cloud services enable users to access a wide range of applications and services without the need for local hardware or software. In addition to offering convenience and accessibility, cloud services also provide several benefits to personal users and businesses, including cost savings, scalability, and improved collaboration. As a result, the adoption of cloud services has grown significantly in recent years, with more and more organizations turning to the cloud to meet their computing needs [1].

However, this rapid growth has led centralization of data, data which includes personal and private information. Google, Facebook, and Amazon have access to vast amounts of data through their various products and services [2]. While companies often use this data to improve their services, target advertising, and develop new products, users must agree to

DOI: https://doi.org/10.37256/cnc.2220244281

trust that these same companies will protect their data. Centralization of data, specifically personal identifiable information (PII), attracts hackers who want steal valuable information. In 2018, Facebook suffered a data breach that exposed the personal information of 87 million users [3], attackers were able to gain access to the personal information of the affected users, including their names, email addresses, and phone numbers. Google Cloud Services has been hacked on multiple occasions. In March of 2018, Google was hit with a data theft that affected over 500,000 Google Cloud customers which resulted in names, email address and passwords being stolen [4]. In addition, there have been several more instances of companies being hacked which resulted in private information in the last 5 years including large companies such as: Equifax [5], Uber [6], eBay [7] and more.

Managing large amounts of data and its users is difficult. Companies often rely on trusted third parties which use centralized models such as Public Key Infrastructure (PKI) [8]. Even though PKI been proven to be effective and secure if well managed, it comes at the cost of being complex, challenging to implement, expensive and open to being a single point of failure [9]. Solutions such as Federated Identity and Single Sign-On (SSO) mitigated management complexity for both providers and users; however, building trust between two or more entitles is difficult and comes with risk that some entities cannot afford [10]. For example, eBay allows sign in with a Google Gmail account, however there no banks that we know of that allow SSO into accounts with Gmail. Furthermore, Federated Identity have not been successful in addressing the widespread problem of passwords or mitigating the risks of privacy violations, security breaches, identity theft, and impersonation associated with the use of passwords [11].

Managing data and users in traditional cloud storage systems requires well a fine grain level of access control. Attributed Based Access Control (ABAC) and Role Based Access Control (RBAC) are both effective in achieving such control [12]. Through these models, data owners enforce a policy that defines attributes or roles a data requester must have to access and decrypt the data. However, ABAC and RBAC rely on a centralized model in which trusted third party called a Private Key Generator (PKG) is required to create and distribute private keys. This puts the PKG in vulnerable position. It becomes a single point of failure and must be trusted not to abuse its power in managing private keys. If comprised, it has the ability decrypt all the user's data. One industry that relies on well managed users and strict access control to information is healthcare. There are several challenges that current healthcare systems struggle with, including the difficulty in understanding how decisions are made, lengthy procedures and delays in diagnosis and communication, time-consuming and costly insurance processes, and issues related to privacy, security, data ownership, and control [13–15].

Blockchain is a type of distributed ledger technology (DLT) that consists of a growing list of records, called blocks, which are linked and secured using cryptography [16]. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. The decentralized nature of blockchain technology allows it to operate without a central authority and makes it resistant to modification of the data. This makes it a secure and transparent method for storing and transferring data and has led to its widespread adoption in a variety of industries such as the automotive, construction and pharmaceutical industry [17–19].

Immutability, traceability, transparency, and privacy are potential benefits blockchain brings and have been proposed to improve identity management and access control. Blockchain-based management systems aim to address issues of data centralization and traditional identity management. Furthermore, rather than relying on a centralized authority to grant and revoke access, blockchain-based systems allow users to securely prove their identity and authorization status using cryptographic keys. This decentralized approach has the potential to increase the security and reliability of access control systems, as it reduces the risk of a single point of failure or vulnerability. Current research however has yet proposed a scheme to combine decentralized identity and access control to create an ecosystem that is completely private, secure, and ultimately give the user more control of their PII.

1.1 *Contributions*

This research introduces a framework to unite blockchain, decentralized identity, and distributed peer-to-peer (P2P) storage to give users more control of their PII, and permissioning power to share private data without centralized services. The contributions are as follows:

- We propose DeAuth, a novel scheme for decentralized authentication and authorization that facilitates the secure sharing of private data. Identity is proven using decentralized identifiers and verifiable credentials, while authorization to share data is performed using the blockchain. Through DeAuth, data owners can manage who has access to their data and how it is used, thus mitigating the risks of unauthorized access by malicious third parties.
- We further present detailed description of the components, functions and algorithms that make up DeAuth.
- We demonstrate a use case scenario and implement a working prototype based on that scenario using the Ethereum [20] blockchain to demonstrate DeAuth's viability.
- We present a present a performance and cost evaluation of the protype compared to using a traditional (non-blockchain) cloud database. We additionally discuss the security concerns of using DeAuth.

The remainder of this paper is structured as follows: Section 2 introduces related works in the fields of blockchain-based access control decentralized identity. Section 3 provides essential background knowledge which DeAuth build upon. In Section 4, we detail DeAuth's components, architecture, functions, and algorithms. Section 5 presents the prototype implementation and development details for creating an application to share private health records. Section 6 provides a security analysis, specifically pertaining to the blockchain, digital wallet and encryption. Section 7 reports on the prototype's performance and cost results. Section 8 discusses the limitation of the study and key findings. Lastly, in Section 9 we conclude with main insights and opportunities for future works.

1.2 *Aims and Scope*

The primary aim of this research is to introduce DeAuth, its fundamental concepts, algorithms, and implementation details. Additionally, this study aims to demonstrate the practical applicability and viability of DeAuth through the development of a functional prototype, supplemented by preliminary evaluations. These preliminary evaluations serve as a foundational framework for subsequent testing and experimentation. The scope of this research is delimited to the development and evaluation of DeAuth within the confines of specific technologies, such as the Ethereum blockchain, which helped facilitate prototype development. Nonetheless, the implications of our findings are positioned to resonate on a broader scale.

2. Related Works

In this section, we review pertinent literature about decentralized access control and its potential pivotal role in within industries such as healthcare. Furthermore, we introduce related research about Self-Sovereign Identity (SSI), enhancing our understanding of the broader landscape.

2.1 *Blockchain-Based Access Control*

Blockchain-based access control (BBAC) refers to the use of DLT to manage and verify the permissions of users within a given system. This approach offers several advantages over traditional access control methods, including increased security, transparency, and decentralization. With a decentralized system, there is no single point of failure, making it more resilient to attacks and tampering. Additionally, the use of cryptographic techniques in conjunction with the blockchain ensures that access permissions are verifiable and tamper-evident, providing a high level of trust and accountability. Overall, the adoption of BBAC has the potential to significantly enhance the security and integrity of various systems and processes.

Current cloud services leverage access control models such as RBAC and ABAC to achieve fine-grained access control. RBAC is a model of access control that determines whether a user is granted access to a particular resource based on their role within an organization, while ABAC determines whether a user is granted access to a particular resource based on their attributes, rather than their identity or membership in a particular group. While RBAC and ABAC provide fine-grain access control feature, those models are derived from ID Based Encryption (IBE) [21] and Attribute Based

Encryption (ABE) [22, 23] which require a trusted third party to create and manage private keys. This inherently creates a reliance on a centralized authority. Additionally, both RBAC and ABAC rely on polices that determine which roles or attributers grant access. The polices rely on a generating services or nodes called Policy Decision Point and Policy Administration Point. If these services or nodes fail, then data which users are trying to access become unavailable [24].

As a solution, studies by [25–27] have been proposed. The schema, described in [25], delegates responsibility for private key management to the user or entity that owns the data, thereby eliminating the need for a PKG. In another approach, the authors in [27] have proposed a smart contract-based authentication mechanism known as RBAC-SC, which provides decentralized RBAC for use in trans-organizational operations. This mechanism leverages blockchain to ensure secure and transparent access control within a decentralized setting. Furthermore, in [26], the authors propose a blockchain-based ABAC solution that not only eliminates the need for a central authority, but also reduces or eliminates costs for service providers. This approach is expected to improve the efficiency and effectiveness of access control, thereby enhancing security, and reducing operational costs.

2.1.1 *BBAC in Healthcare*

Blockchain technology has the potential to revolutionize various aspects of the healthcare industry by improving efficiency, security, and interoperability of data handling. One key area where blockchain is being utilized in healthcare is in the management of Electronic Health Records (EHRs) and Electronic Medical Records (EMRs). Additionally, data such as those originating from devices such heartrate and blood pressure monitors should be handled with property authentication and authorization procedures [28]. By using DLT, healthcare providers can securely store and share patient data with authorized parties in a transparent and verifiable manner. This can help to reduce errors and improve the accuracy of patient records, while also enabling better communication and coordination among healthcare professionals.

Recent surveys have shown that blockchain technology is increasingly being proposed as a potential solution to various problems in the healthcare sector [29–31]. A recurring theme across these studies is the management of EHRs, which has been identified as the most targeted area for blockchain research. In a study conducted by [29], it was noted that the healthcare industry faces general challenges such as data fragmentation, security, and privacy, and that blockchain technology can provide a robust solution to address these issues. The papers reviewed in this survey primarily focused on new schemes for improving EHR and EMR systems.

Interoperability between blockchain and legacy data management systems was also identified as an area of interest. In [31], the potential of blockchain technology for patient data and identity management was investigated, with the authors concluding that EHRs and Patient Health Records are at the core of blockchain applications in healthcare. Overall, these surveys indicate that blockchain technology has the potential to address various challenges in the healthcare sector, particularly in the management of EHRs and interoperability between legacy systems and blockchain-based solutions.

2.2 *Self-Sovereign Identity*

SSI is a concept in digital identity management in which individuals have full autonomy over their personal data and identity information [10, 11, 32–34]. This differs from traditional, centralized identity management systems, which a centralized authority, such as a government or corporation, controls and manages an individual's identity. In SSI, individuals are the custodians of their own identity information, and can selectively disclose this information to various parties as needed. This can be achieved by utilizing DLT to secure and tamper-proof the storage of identity information.

One of the key benefits of SSI is that it empowers individuals to have greater control over their personal data and identity information. This can help to protect privacy and enable people to access services and participate in online transactions more easily. Another key benefit of SSI is that it can help to reduce the reliance on centralized authorities and intermediaries, which can improve security and reduce the risk of data breaches. Additionally, SSI can help to promote interoperability and reduce the siloing of identity information, which can make it easier for individuals to access services and transact online.

SSI is confronted with a multitude of challenges, ranging from adoption and interoperability to technical complexity, security, and trust [10]. To overcome these obstacles, scholars have proposed various solutions, including the Self-Sovereign

Identity Based Access Control (SSIBAC) model, as introduced by the authors in [35]. This approach offers an access control framework for cross-organizational identity management by integrating traditional access control models and blockchain technology. Decentralized authentication is followed by centralized authorization, ensuring secure and reliable identity verification. To further enhance privacy, Zero Knowledge Proofs (ZKP) have been utilized [36]. ZKP is a cryptographic technique that enables one party, the prover, to demonstrate the veracity of a statement to another party, the verifier, without revealing any additional information. This feature is especially beneficial in the context of SSI, as it allows individuals to authenticate their identity to others without disclosing their personal information.

3. Background Knowledge

This section provides background knowledge on the foundational technologies upon which DeAuth's components and architectural framework are built on. These encompass: blockchain, smart contracts, decentralized identity and the InterPlanetary File System.

3.1 *Blockchain and Smart Contracts*

The blockchain is a decentralized, immutable, and cryptographically secure digital ledger that was first introduced in a white paper by [37]. In this paper, the blockchain serves as the foundational technology that facilitates the operation of Bitcoin, a type of cryptocurrency, and the corresponding Bitcoin protocol. Nakamoto's motivation for creating Bitcoin and the Blockchain was to develop a secure, electronic peer-to-peer cash system that was not under the control of a single entity. The blockchain maintains a record of transactions, which, in the case of Bitcoin, refer to the sending and receiving of Bitcoin.

A smart contract is a program that runs on the blockchain. The concept of a smart contract was first proposed by Nick Szabo and defined as a computerized contract transaction protocol [38]. Ethereum was the first blockchain to introduce the smart contract to its protocol and allowed users to code and extend the functionality of a blockchain beyond recording transaction of digital currency [20]. Once on the blockchain, smart contracts reside at a specific address and cannot be deleted.

A smart contract is written similar to writing a class in object-oriented programming, see Figure 1. In Ethereum, the contracts are written in a programming language called Solidity. Moreover, the Ethereum Virtual Machine (EVM) [39] is the computation engine that enables smart contract functionality and manages the state of the blockchain. After a contract is written, it is compiled and submitted as a transaction to the blockchain. The transaction is known as contract deployment transaction and like any other transaction, it must be verified by and propagated to all nodes on the network. To interact with the smart contract, users submit a transaction using their wallet, which then calls and executes a specific function within the contract.

```
1. Contract SimpleStorage {
2.   uint storedData = x;
3.
4.   function set(uint x) {
5.     storedData = x;
6.   }
7.
8.   function get()public view returns (uint) {
9.     return storedData;
10.  }
11. }
```

Figure 1. Example smart contract that stores and retrieves an integer to and from the blockchain.

3.2 *Identity Management*

3.2.1 *Centralized and Federated Identity*

Decentralized identity is focused on placing identity control in the hands of its owner. This approach differs from the two existing identity management models: centralized and federated [40]. In a centralized model, shown in Figure 2, an identity provider takes charge of managing and authenticating user identities across an organization. This model provides a centralized control point for user identities, which facilitates the management and security of user access to multiple systems and applications within the organization.

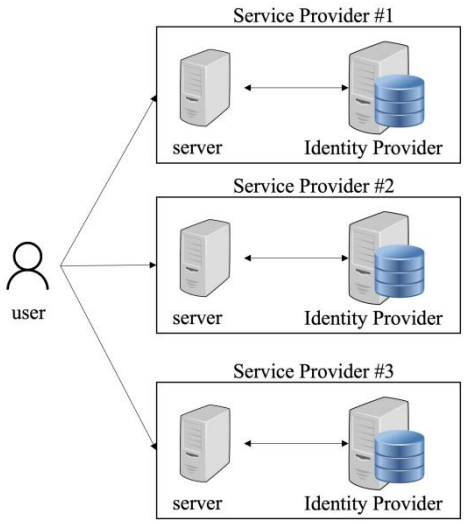


Figure 2. Centralized identity model.

In a federated model, two or more centralized systems establish trust allowing one system to authenticate an entity for another, see Figure 3. This is typically achieved with a third-party identity provider, such as Microsoft Active Directory or Google, that manages and authenticates the user's identity. The user's identity is then shared among the various systems and applications that the user needs to access, eliminating the need for the user to remember and manage multiple sets of login credentials. This approach helps to increase security, reduce administrative overhead, and improve the user experience [41].

SSO is a solution derived from Federated Identity, which allows users to access multiple systems and applications with a single set of login credentials. This process simplifies the user experience by eliminating the need for users to remember and manage multiple usernames and passwords. Additionally, SSO reduces the risk of password-related security breaches. Upon successful verification of the user's identity, the identity provider issues a security token, such as a JSON Web Token, which includes the user's identity and other relevant information. The systems and applications that the user needs to access can then use this token to authenticate the user and grant access.

One of the key benefits of SSO is that it reduces administrative overhead by eliminating the need to manage multiple sets of login credentials [42]. This can be especially useful in organizations with large numbers of users, as it can greatly reduce the burden on IT staff. Additionally, SSO can improve security by reducing the risk of password-related security breaches, as well as making it easier to detect and respond to unauthorized access attempts.

While providing convenience to end users, there are privacy challenges involved in the federated model due to the large volume of exchanging personal identifiable information across organizations [43]. This challenge additionally incurs cost due to the increased security infrastructure needed to share valuable information across domains using loosely coupled network protocols [41].

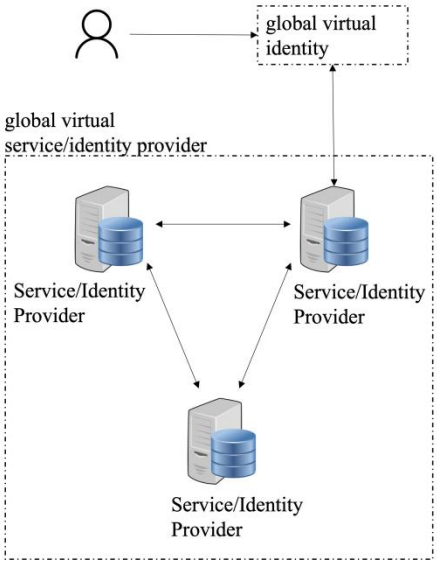


Figure 3. Federated identity model.

3.2.2 *Decentralized Identity*

Decentralized identity is a management paradigm that relies on cryptography and distributed ledger technology to give entities more control over their identity. The entity assumes full responsibility of their personal indefinable information. In a decentralized identity system, identity is proved with digital signatures [44]. The system is comprised of multiple components which we define below:

- **Decentralized Identifier (DID):** a unique identifier, that unlike a traditional identifier that is created by a centralized service (e.g., an email address), is instead created by the owner themself [45]. DIDs are assigned a public/private key pair, thus can be digitally signed. Depending on its purpose, DIDs can either be public or private [46]. Moreover, they can be used for establishing secure communication channels. The DID syntax has been defined by the W3C [47]. There are two main parts the specification method and the identifier. The specific method defines how to read and write a DID and its DID Document. The identifier is a unique data string that represents the DID. An example of a DID is:

  $$did{:}btcr{:}xyv2\text{-}xzpq\text{-}qrst\text{-}n5pk$$

  This DID is associated with a Bitcoin address and is used to identify the owner of the address in a decentralized system. The prefix *did:btcr* indicates that this is a DID on the Bitcoin blockchain, and the string of characters that follows is the unique identifier for the DID owner.

- **DID Document (DDO):** a DDO is a digital document that is used to verify the identity of an individual or organization in a decentralized system. DDOs contain information about the identity of the individual or organization, including their name, address, and other relevant details. They can also include public keys, which can be used to authenticate the identity of the DID document owner and enable secure communication, see Figure 4 [47]. Every DID has an accompanying DDO. The DDO can be stored on the blockchain itself or another location (i.e., off-chain) and mapped to a DID using a DID Resolver [46].

```
{
  “@context”: https://w3id.org/did/v1,
  “id”: “did:example:123456789abcdefghi”,
  “publicKey”: [{
    “id”: “did:example:123456789abcdefghi#keys-1”,
    “type”: “RsaVerificationKey2018”,
    “owner”: “did:example:123456789abcdefghi”,
    “publicKeyPem”: “-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n”
  }],
  “authentication”: [{
    // this key can be used to authenticate as DID ...9938
    “type”: “RsaSignatureAuthentication2018”,
    “publicKey”: “did:example:123456789abcdefghi#keys-1”,
  }],
  “service”: [{
    “type”: “ExampleService”,
    “serviceEndpoint”: https://example.com/endpoint/8377464
  }]
}
```


Figure 4. Example DID document.

- **Distributed Ledger:** while a blockchain is not required for decentralized identity, they provide a ready-made infrastructure for managing data in a decentralized way [10]. Additionally, transactions related to DIDs, DDOs and credentials can be notarized. This in turn provides proof when data was created and provides an electronic seal to reveal when tamping has occurred. An example DLT platform for managing a decentralized identity network is Hyperledger Indy [48].
- **DID Resolver:** a DID resolver is a piece of software that is responsible for resolving DIDs to their corresponding DDO. It allows users to look up and retrieve DID documents using the corresponding DID. This is accomplished by querying the ledger on which the DID is stored and returning the DID document that is associated with the DID. The DID resolver is designed to work in a decentralized environment and is typically implemented as a software library or API that can be integrated into other applications or systems. Works such as [35, 49] have used the blockchain and smart contracts to implement a DID resolver.
- **Verifiable Credential (VC):** A VC is a digital document that contains a set of claims about an individual or organization that can be independently verified by a third party. VCs are designed to be self-sovereign, i.e., that the individual or organization that holds the credential has control over it and can use it to prove their identity or other attributes without relying on a centralized authority. They can be used to prove a wide range of attributes, including personal information (e.g., name, date of birth), professional qualifications, and other relevant details, see Figure 5 [47]. VCs are often issued by trusted organizations, such as schools, government agencies, or professional associations, and can be verified by anyone who has access to the credential and the necessary tools to check its validity. Due to personal and private information being stored within the documents claims, VCs are not stored on public database or ledger, but rather on the owner’s storage device and managed by a software known as a digital wallet [50].

```
{
  "@context": https://w3id.org/did/v1,
  "id": "did:example:123456789abcdefghi",
  "type": ["Credential", "ProofOfAgeCredential"],
  "issuer": "https://dmv.example.gov",
  "issued": "2010-01-01",
  "claim": {
    "id": "did:example:ebfeb1f712eb6f1c276e12ec21",
    "ageOver": 21
  },
  "revocation": {
    "id": http://example.gov./revocations/738,
    "type": SimpleRevicationList2017"
  },
 "service": {
    "type": "LinkedDataSignature2015",
    "created": "2016-06-18T21:19:10Z",
    "creator": https://example.come/jdoe/keys/1,
   "domain" "json-ld.org",
   "nonce": "598c63d6",
   "signatureValue": "BavEll0/I1zpYw8XNi1bgVg/sCne04Jugez8RwDg/+MC
   RVpj0boDoe4SxxKjkC0vKiCHGDvc4krqi6Z1n0UfqzxGfmatCuFibcC1wps
   PRdW+gGsutPTLzvueMWmFhwYmfIFpbBu95t501+rSLHIEuujM/+PXr9Ck
   y6Ed+W3JT24="
  }
}
```


Figure 5. Example verifiable credential.

- **Verifiable Presentation (VP):** a VP is a digital document that contains a set of claims about an individual or organization that has been attested to by a trusted third party. It is the document that is generated and presented to a verifier upon their request to validate claims about a subject. One main difference between VCs and VPs is that claims within VPs are derived from one or more credential. Another difference is that a VP contain both the cryptographic signature from the issuer who issued the VC and additional signature from subject who generated the VP. This is to prevent a replay attack [47]. The additional signature contains a challenge the verifier must solve upon verification. An important aspect of a VP is that the subject is given the choice to selectively reveal only the information that is necessary for a specific purpose, while still maintaining control over their own identity and personal information.

- **Claims:** a claim is a statement about an individual or organization that can be proven or disproven. Claims are a key component of VCs and VPs and are used to prove the identity of an individual or organization, known as the subject, in a decentralized system. Examples of claims are their name, date of birth, or professional qualifications. Claims can be proven or dis-proven using cryptographic techniques, which allows for the creation of a tamper-evident records, the verifiable credential.

- **Credential Schema:** a credential schema is a standardized set of rules and guidelines that defines the structure and format of a verifiable credential. They include information about the types of claims that can be made in a verifiable credential, as well as the format and structure of the credential. Credential schemes are used to ensure that verifiable credentials are interoperable, thus allowing for the easy exchange and verification of verifiable credentials across different systems and contexts.

Decentralized identity is a rapidly evolving field that is gaining increasing attention from both academia and industry. It can be implemented with or without DLT if it supports the necessary features and capabilities. The following are list networks and platforms being developed and used today: uPort (https://uport.me/), Sovrin (https://sovrin.org/), SelfKey (https://selfkey.org/) and Civic (https://www.civic.com/). The authors in [10] provide an evaluation of each of the mentioned platforms along with the challenges and limitations.

3.3 *InterPlanetary File System*

The InterPlanetary File System (IPFS) is a distributed file system protocol that aims to make the internet more resilient and scalable [51]. It is based on a P2P network architecture, in which nodes communicate with each other to store and retrieve data. In contrast to traditional client-server architectures, in which data is stored on centralized servers and accessed by clients through network requests, IPFS allows users to access data directly from other nodes in the network.

An integral component of IPFS is the content identifier (CID), which serves as a unique identifier for files or content within the network. A CID is a unique identifier that is used to identify and locate a file or piece of content within the IPFS network. CIDs are generated by hashing the file and result in a fixed-length string of characters that are unique to the specific content, see Figure 6. The resulting CID is used to locate and retrieve the content. An example CID is *QmV8RgHXhv7n68EoyYG5N8rGZn3vZ5z5LcN5Z8uAVhX9y7*. Users can then use the CID to retrieve the content from the IPFS network.

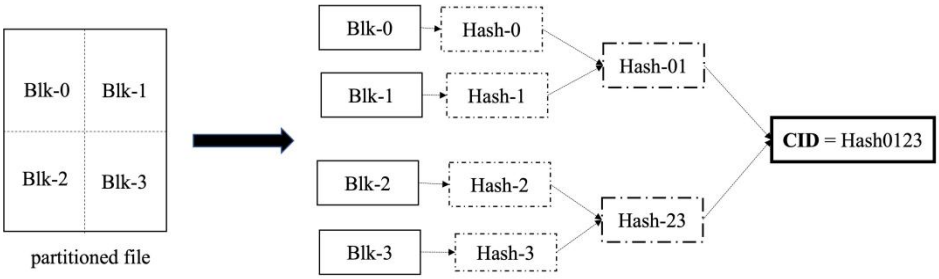


Figure 6. Illustrating the process of creating a CID. A file is first partitioned into smaller sized data chunks. Then hashed into a Merkle Tree with the root hash becoming the CID.

IPFS allows for the decentralization of data storage and retrieval. This means that data is not stored on a single central server, but rather is distributed across a network of nodes. This reduces the risk of data loss or downtime, as data can still be accessed even if one or more nodes go offline. Additionally, the decentralized nature of IPFS can make it more efficient and faster to retrieve data, as it allows users to access data from the node that is closest to them, rather than having to make a request to a central server that may be located far away. Overall, IPFS has the potential to significantly improve the speed, resilience, and scalability of the internet.

4. Proposed Scheme

In this section, we give a comprehensive description of DeAuth's concepts, components, and architecture. We explain their purposes, inner workings and how they are interconnected. Moreover, we explain the programmatic functions and algorithms underpinning DeAuth's functionality, offering a thorough examination of its operations.

4.1 *Concepts and Components*

DeAuth integrates three concepts within a decentralized system: identity management, authentication, and authorization (i.e., access control) in order to share private data. Users are identified by private keys, public keys, blockchain addresses, DIDs, VPs and VCs. An identity management system is necessary to store and manage those identifies. Authentication is necessary such it allows users to verify who they say they are. In DeAuth, authentication proves a user owns (i.e., has the private keys to) a DID and signatures in a VC or VP. Authorization is necessary to dictate who is allowed to share and obtain another user's private data. Now we introduce the components necessary to support the aforementioned concepts.

DeAuth is comprised of three main components: the wallet, smart contracts, and IPFS all of which are unified by the blockchain, depicted in Figure 7. The wallet's main responsibilities are identity and blockchain address management.

All users must have a wallet to hold an identity and interact with smart contracts. The smart contracts contain DeAuth's business logic. There are two smart contracts, one to manage the creation and lookup of DIDs and the other to submit authorization messages; both of which are discussed upcoming in Section 4.1.2. The last component is IPFS. Authorization messages and private user data are not stored on the blockchain to minimize the memory growth rate and uphold privacy. Rather they uploaded and stored to a IPFS node that is hosted by third-party service, making the data available for sharing. Users have the option to self-host an IPFS node, providing benefits latter discussed in Sections 6.3 and 8.2. We elaborate on each the components the following sections.

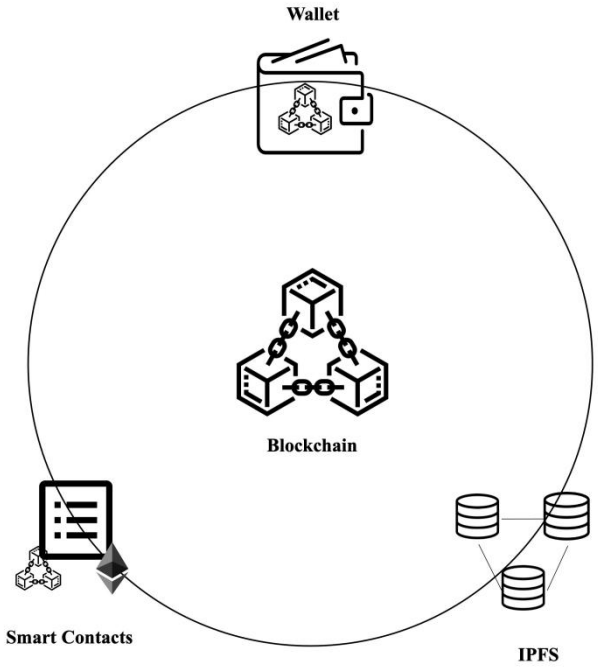


Figure 7. DeAuth components.

4.1.1 *Wallet*

The (digital) wallet enables users to oversee their identity and engage with the blockchain through smart contracts. Wallets are predominantly deployed as desktop, mobile, or web applications, each tailored to meet specific requirements contingent upon the system in which they are integrated. For instance, wallets incorporated into decentralized identity systems are designed to manage DIDs, whereas wallets integrated into blockchain systems primarily oversee blockchain addresses and facilitate the submission of transactions to the blockchain. Given that DeAuth constitutes a hybrid application integrating both decentralized identity and blockchain functionalities, the wallet requires a multifaceted feature set to accommodate these dual responsibilities. The features of DeAuth's wallet encompass the following:

- Generation of public and private keys.
- Creation, retrieval, and revocation of DIDs.
- Storage of addresses, DIDs, and VCs.
- Verification of VCs and VPs.
- Creation of blockchain addresses.
- Communication with the blockchain.

- Interaction with smart contracts.

4.1.1.1 *Key Management*

The DeAuth wallet operates as a deterministic wallet, employing a hierarchical deterministic (HD) key generation method [52]. In HD key generation, both private and public keys originate from a common secret or seed. Importantly, as long as the seed remains unchanged, an identical set of keys can be reliably regenerated. Conversely, non-deterministic wallets adopt an alternative approach, generating each key from distinct, randomly generated seeds, thereby precluding the replication of the same keys. This characteristic affords both deterministic and non-deterministic wallets the capability to generate a virtually limitless number of keys. To ensure compatibility across multiple blockchain platforms, our wallet adheres to standardized wallet implementation protocols, including those specified in BIP32 [53] and BIP44 [54].

A deterministic wallet proves advantageous for DeAuth and similar blockchain applications for several compelling reasons. In the event a user inadvertently deletes their wallet along with its associated keys, the sole possession of their seed facilitates the seamless recovery of the same set of keys. Furthermore, keys are systematically organized within deterministic wallets, akin to the hierarchical structure of files and folders on a hard drive, depicted in Figure 8 [55]. Lastly, deterministic wallets possess the unique ability to generate public keys independently of private keys, rendering them suitable for deployment on potentially insecure server environments.

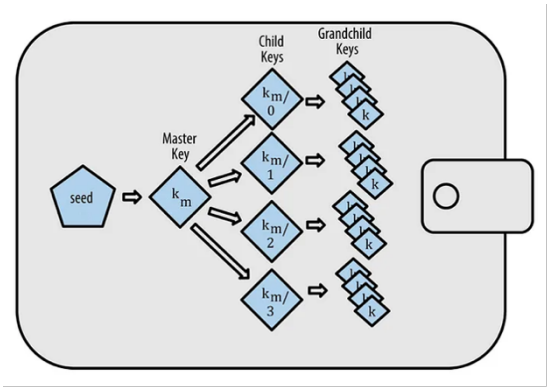


Figure 8. Example of HD key generation in a blockchain wallet.

4.1.1.2 *Blockchain Address and Identity Management*

Blockchain addresses (we refer to blockchain address(es) as just address(es) for the rest of the paper for brevity) and DIDs are both derived from public keys and are stored in the digital wallet. For the wallet to accommodate DIDs and distinguish whether a public key was used to create an address or DID, we utilized the BIP44 standard. BIP44 defines a standard framework for organizing keys in an HD wallet [54]. This adaptation allows the wallet to be used with multiple different cryptocurrencies and to support different use cases. The BIP44 levels are:

m / purpose / coin type / account / change / address index

The "m" stands for "master seed" and is constant; all other levels are represented with an index value (0, 1, 2, ..., etc.) and together form a *path*. The path is necessary when generating a public/private key pair. An example path is m/44/60′/0/0/0, the typical path of the first key pair. The 44 represents the BIP44 standard and the 60 is the constant value for the Ethereum Blockchain. More coin types can be found in [54].

In DeAuth, we employed the *account* and *change* levels to accommodate DID management. First, we placed a rule at the account level such that even values represent accounts used for blockchain address and odd values represent accounts used for DIDs. Furthermore, we added an additional rule at the change level. Originally value 0 is used for external chain and 1 for internal chain; external chain is used for addresses that are meant to be visible outside of the wallet (e.g., for

receiving payments) and internal chain is used for addresses which are not meant to be visible outside of the wallet and is used for return transaction change. In DeAuth, 0 is now used to create public addresses and DIDs while 1 is used to create private addresses and DIDs. Recall private DIDs are not published on the blockchain and shared only between participants to establish private communications between wallets. Example paths for addresses and DIDs are shown below:

- m/ 44′/ 60′/0 /0/0—public blockchain address
- m/ 44′/ 60′/0 /1/0—public blockchain address
- m/ 44′/ 60′/1 /0/0—public DID
- m/ 44′/ 60′/1 /1/0—private DID

4.1.1.3 *Agents and Services*

Agents are used to facilitate interactions between users and perform a range of services. They function as intermediators, aiding users with identity management and enabling identity verification to external parties. In DeAuth, these agents are seamlessly integrated into the wallet software and are made accessible through Application Programming Interfaces (APIs). Services are generally public and integrated into a DDO, see Figure 9. Services that agents perform in DeAuth are:

- Message transmission.
- Message and file encryption/decryption.
- Authenticate and verify DIDs, VCs and VPs.
- Wallet information management.

```
{
  …other DID Document properties,
  "service": [{
    "id": "did:example:bob",
    "type": "Verification",
    "serviceEndpoint": "https://walletagent.io.verifyDID"
  }]
}
```

Figure 9. Example of a service defined in a DDO.

4.1.2 *Smart Contracts and Authorization Messages*

DeAuth utilizes two smart contracts, the identity contract and authorization contract. The source code is shown in Appendixes Figures A1 and A2. The Identity Contract is responsible for creating and looking up DIDs, while the authorization contract is responsible for submitting authorization messages that control data sharing permissions. Furthermore, only one Identity Contract is required to be deployed; however, there can be many authorization contracts deployed. The smart contract's address is included as a service in a DDO so that wallet holders know where to send transactions to, see Figure 10.

```
{
  …other DID Document properties,
  "service": [{
    "id": "did:example:datamanager",
    "type": "SmartContract",
    "serviceEndpoint": "0x6ac7ea33f8831ea9dcc53393aaa88b25a785dbf0"
  }]
}
```

Figure 10. Example of smart contract definition in a DDO.

4.1.2.1 *Authorization Messages*

Authorization messages regulate data sharing permission in a secure and transparent manner. Additionally, authorization messages enable a comprehensive record of all the permissions requested and granted for data sharing between users. They are stored and shared using IPFS. Their CID is recorded and submitted as a transaction to the blockchain via the Authorization Contract; we discuss the authorization message submission sequence in the Section 4.2. Each authorization message is labeled with a type identifier that describes the functionality of the message. Currently, there are five message types; however, can be extended in future works:

1. ***RequestData***: request data not owned by the requesting party.
2. ***AskPermission***: ask permission from the owner to share data with the requesting party.
3. ***AllowPermission***: owner grants approval for the transfer of the user's private data.
4. ***RejectPermission***: owner declines the transfer of a user's private data.
5. ***ShareData***: share data with requesting party.

Messages are divided into two categories: non-data (Figure 11) and data (Figure 12). We use JavaScript Object Notation (JSON) [56] to define the schema. All types except the ShareData message type uses the non-data schema.

```
{
  to: the receiver's public DID,
  from: the sender's public DID,
  type: the authorization message type
}
```

Figure 11. Non-data authorization message schema.

The data schema uses the same property fields as the non-data schema and has additional fields for the CID of the encrypted data being shared, and the CID of the encrypted session key; we discuss the session key next in Section 4.1.3.

```
{
  to: receiver's public DID,
  from: the sender's public DID,
  type: the authorization message type,
  encryptedData: CID of the encrypted data,
  encryptionKey: CID of the encrypted session key
}
```

Figure 12. Data authorization message schema.

4.1.3 *Data Storage and Encryption*

DeAuth uses IPFS for storing and sharing data such as authorization messages and files. Data is costly to store on the blockchain. Given the substantial cost associated with storing large volumes of data on the blockchain, IPFS serves as an off-chain storage layer. Only the DIDs and CIDs are retained on the blockchain. Moreover, the requested private data is shared only when an *AllowPermission* message is submitted by the data's owner. The nature of this data can encompass a variety of file types, including documents, images, audio, and videos. Prior to sharing, the data is assumed to be stored on the user's hard drive or other storage mediums—e.g., cloud, external flash drive, etc.

In the interest of safeguarding user privacy and anonymity, authorization messages undergo encryption using the recipient's public key prior to submission. This public key is derived from a private DID specifically created to facilitate private communication between two designated users. Furthermore, it is imperative to note that shared private data undergoes encryption as well. Within the IPFS framework, all data is inherently public and thereby accessible to all users within the network. Non-encrypted data is particularly susceptible since it remains in plaintext form. To address this security concern, DeAuth employs AES encryption using a randomly generated symmetric key, denoted as k—i.e., a session key [57]. Key k itself is encrypted with the requester's public key and is then stored on the IPFS network, thus allowing the requester to retrieve and use k for decryption. We further discuss the storage and encryption algorithms in Section 4.2.2.

4.2 *Architecture*

DeAuth's architecture is comprised of two main schemes, authentication and authorization. The authentication scheme involves two sequences, DID authentication and VC verification, illustrated in Figure 13. In DID authentication, a subject proves to verifier they are the owner of a DID, while in VC verification, a subject proves to a verifier that the claims in a VC have been digitally signed by an issuer.

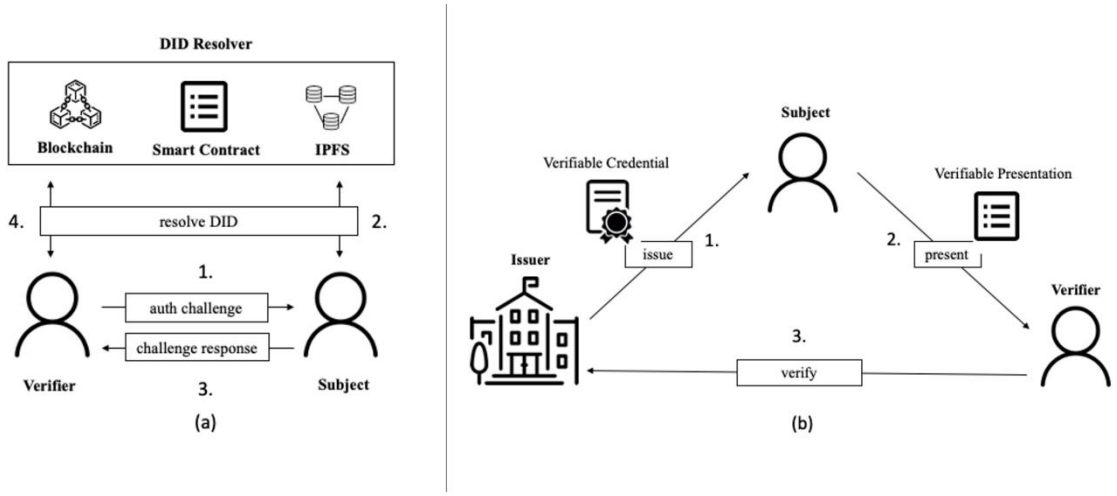


Figure 13. Two authentication services in DeAuth: (**a**) DID authentication and (**b**) VC verification. DID authentication has four main steps: (1) Verifier sending a challenge, (2) Subject resolves the Verifier's DID, (3) Subject sends back challenge response and (4) Verifier resolves Subject's DID. VC verification has three main steps: (1) Issuer sends Subject a VC, (2) Subjects sends Verifier a VP, and (3) Verifier verifies the Issuers digital signature within the VP.

Figure 13a illustrates DIDs are authenticated through a challenge-response process and DID resolution. Both sequences are shown in Figure 14. To pass the challenge, the verifier sends the subject a random string (i.e., challenge) and requires the subject to encrypt it with both its private key and the verifier's public key. The encrypted challenge is sent back to the verifier to decrypt the response with their secret key and the subject's public key. The subject passes the challenge if the string remains unchanged. The verify and subject obtain the others public keys through DID resolution, where the

blockchain and IPFS serve as a DID Resolver (Section 3.2.2). In DID resolution, a DID is looked up on the blockchain via the Identity Contract (Section 4.1.2) to get DDO's CID. The DDO can then be retrieved from IPFS using the CID.

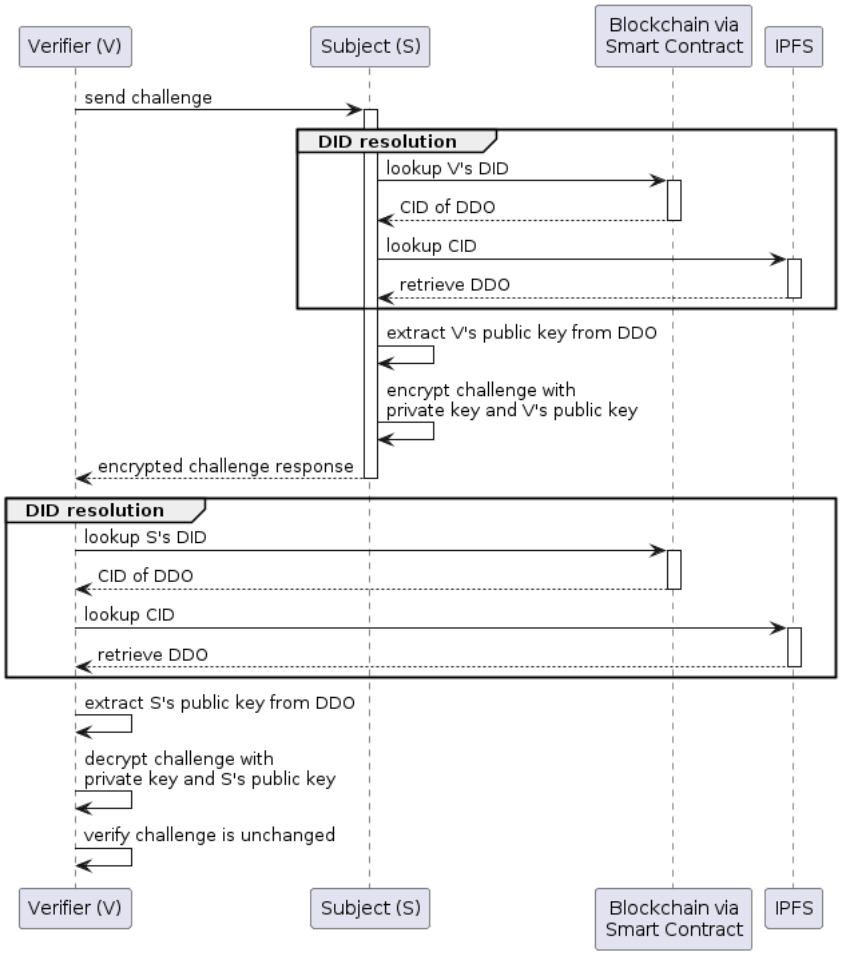


Figure 14. DID authentication and resolution sequence diagram.

DID authentication is necessary for VC verification. Figure 13b shows there are three main steps: (1) issue, (2) present, and (3) verify. The entire sequence is illustrated in Figure 15. When a verifier requires a VC (e.g., to prove one meets a specific age requirement), the subject must first request a VC from a trusted issuer. The subject and issuer perform DID authentication on each other before the issuer can add claims, sign, and send a VC. It is assumed the VC is sent over a secure connection using DIDs; we discuss this feature later in Section 5.1.2. The VC is saved in a wallet and the subject can now request access to the verifier's service. In response, the verifier requests a VP. The subject generates a VP, derived from a VC, and adds a signature using their private key—refer to Section 3.2.2 about reply attack. Before sending the VP, the subject and verifier authentication each other's DID. Lastly, the VP is sent to the verifier and verifies the signatures from both the subject and issuer.

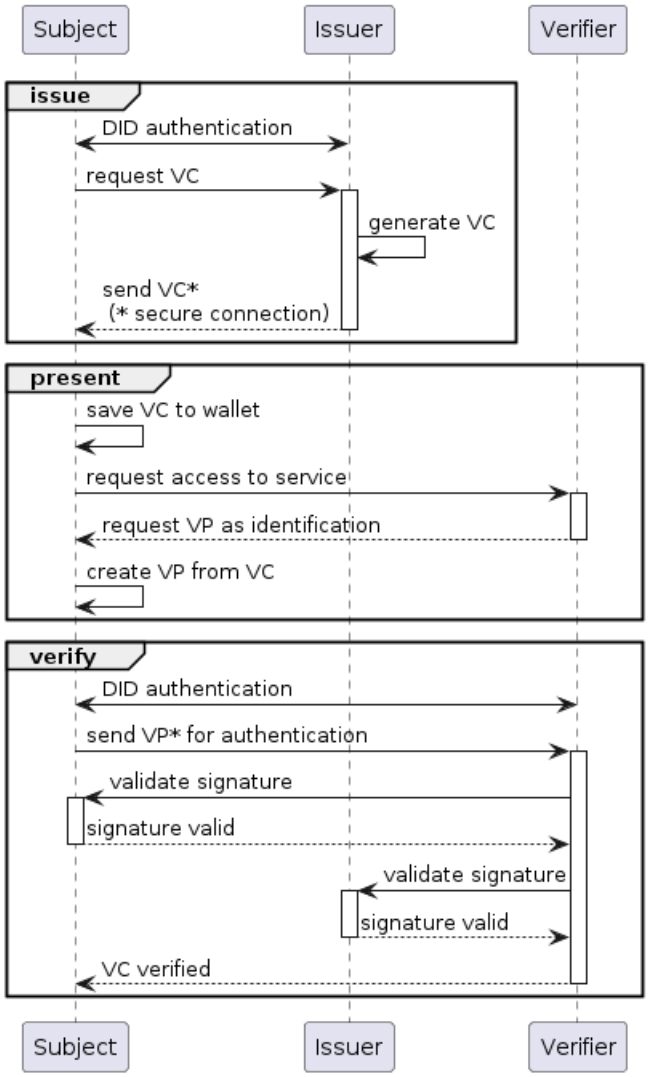


Figure 15. VC verification sequence diagram.

Figure 16 illustrates the authorization scheme and how private data is shared after users are authenticated. In this scheme, are three types of users the Data Manager (DM), Data Owner (DO) and Data Requester (DR). The DM is an entity that manages and stores private data about a DO. DMs cannot share this data without a DO's permission. The DR is the user that is requesting data that belongs to DO and is managed by a DM. Users create and submit authorization messages request private data, allow permission, and share that data to one another.

Authorization messages are created by the users, saved to IPFS, and submitted to the blockchain via the Authorization Contract. All messages are encrypted before being saved to IPFS. Figure 17 shows the sequence of this process between a sender and receiver. The sender first creates a message—e.g., *RequestData*, *AllowPermission*, etc. Next the sender and receiver perform DID resolution which allows both to obtain the others' public key. The message is then encrypted with the receiver's public key. The sender saves the encrypted message to IPFS, obtains the CID, and submits the CID to the blockchain. A TID is generated from the submission and is sent to the receiver. The receiver looks up the TID on the blockchain to retrieve the CID, then uses the CID to retrieve the encrypted messages. Last, the receiver uses their private key to decrypt the message to reveal its contents in plain text.

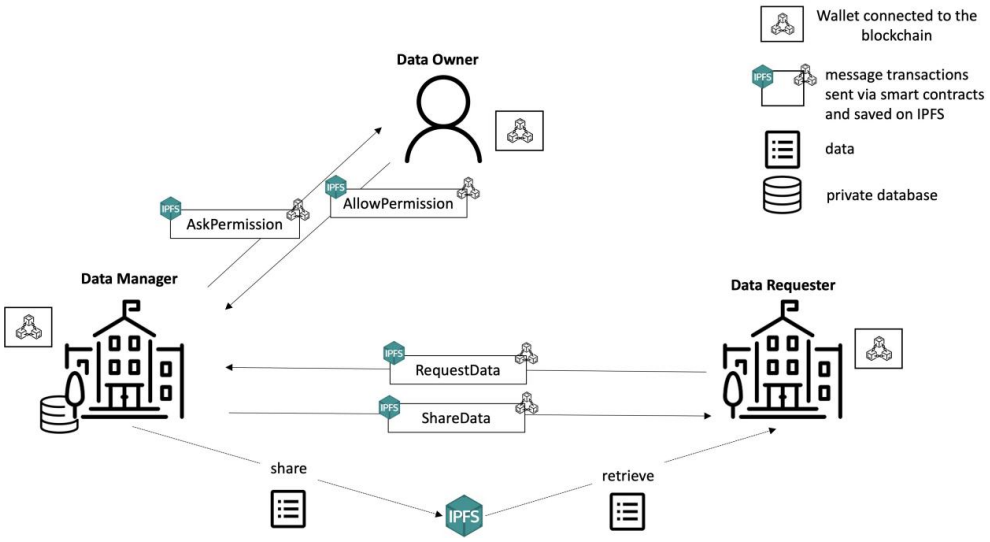


Figure 16. DeAuth authorization scheme. The scheme is made of six main steps: (1) the DR submits a *RequestData* message, (2) the DM submits a *AskPermission* message, (3) DO submits a *AllowPermission* message, (4) DM shares DO's private data through IPFS, (5) DM shares a *ShareData* message and (6) DR retrieves DO's private data through IPFS.

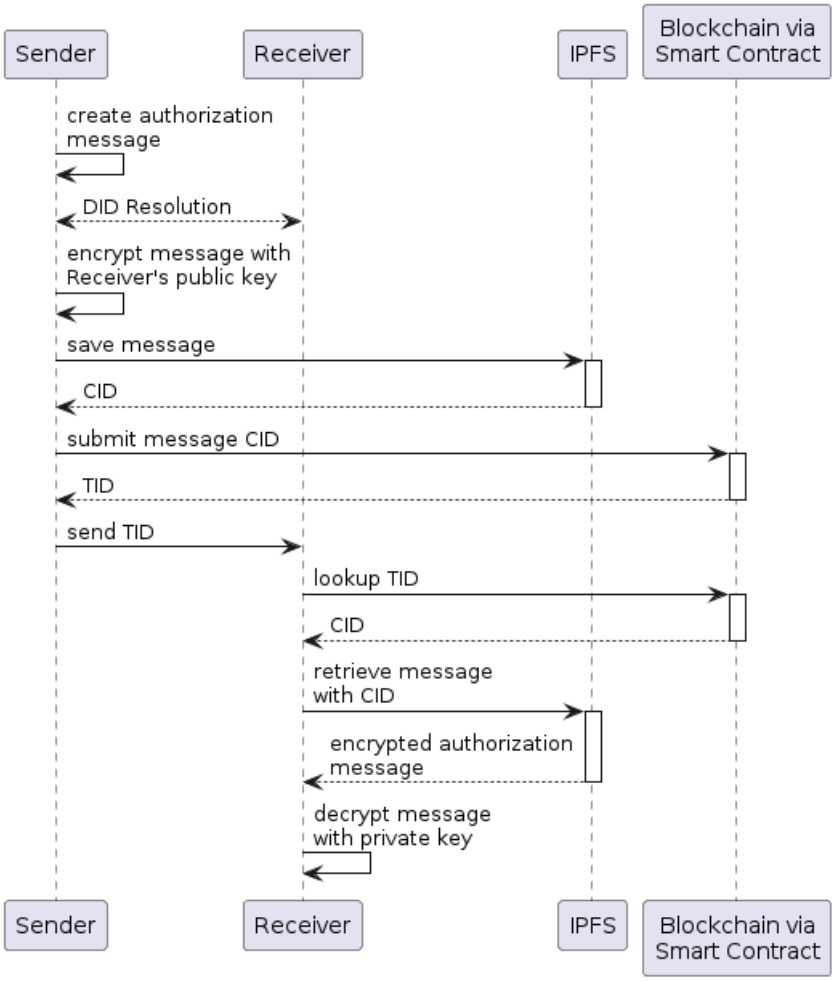


Figure 17. Sequence diagram for submitting an authorization message within the authorization scheme.

4.2.1 *Functions*

In this section we define all major functions used in authentication and authorization schemes. They are categorized into six categories: (1) general, (2) wallet, (3) smart contract, (4) identity, (5) authorization message and (6) data.

1. General Functions:

- ***sk, pk*** **←*genKeyPair* (*seed,path*)**—generates a private (i.e., secret) key (*sk*) and public key (*pk*) pair.
- ***k*** **←*genSessionKey*()**—generates a session key (*k*) for data encryption.
- CID_m **←*lookupTID*(*TID*)**—looks up a blockchain transaction identifier (TID) on the blockchain and returns the IPFS CID for an authorization message (*m*).

2. Wallet Functions:

- ***createPublicConn*(*DID*$_1$, *DID*$_2$)**—creates a public connection between two wallets using two distinct public DIDs.
- ***createPrivateConn*(*DID*$_1^*$,*DID*$_2^*$)**—creates a private connection between two wallets using two distinct private DIDs.
- ***true* | *false*** **← *authenticateDID*(*DID* | *DID**)**—authenticates a public or private DID by sending the owner a challenge using the *pk* associated with the DID; the *pk* is found in the DDO. Returns true if the owner has the associated *sk* to solve the challenge; false otherwise.
- ***true* | *false*** **← *authenticateVP*(*VP*)**—authenticates a VP. Return true if VP*'s* owner and issuer can verify the signatures attached with the VP; false otherwise.
- ***send*(*DID*$_{sender}$, *DID*$_{receiver}$, *VP* | *DID* | *TID*)**—sends a VP, DID, or TID to a user's wallet.

3. Smart Contract Functions:

- ***TID*** **←*publishDID*(*DID,CID*$_{DDO}$)**—calls the method in the Identity Contract to store both a DID and CID_{DDO} on the blockchain, which results in making the DID public.
- CID_{DDO}**← *lookupDID*(*DID*)**—calls the method in the Identity Contract to look up a DID and return the CID_{DDO} associated with a DID.
- ***TID*** **← *submitMessage*(*t,CID*$_{mCT}$)**—calls the method in the Authorization Contract to submit an encrypted authorization message **(*CID*$_{mCT}$)** to the blockchain labeled with type *t*.

4. Identity Functions:

- ***DID*** **←*createDID*(*pk*)**—creates a DID from a public key.
- ***DDO*** **←*createDDO*(*DID*)**—creates a DDO from a DID.
- ***signature*** **←*extractProof*(*DDO* | *VP*)**—extracts and returns the proof (i.e., digital signature) from a DDO or VP.
- ***true* | *false*** **←*verifyProof*(*signature, pk*)**—verifies a signature using a public key. Returns true if verified; false otherwise.
- ***pk*** **←*extractPublicKey*(*DDO*)**—extracts a public key from a DDO.
- ***pk*** **←*retrievePublicKey*(*DID*)**—returns a public key associated with a DID. This function is comprised of multiple functions; its details are found within Algorithm 3 in the next section.

- $DID_i^{*CT} \leftarrow encryptPrivateDID(pk, DID^*)$—encrypts a private DID with a public key so that the private DID can be sent securely.

5. Authorization Message Functions:

- $m_t \leftarrow createMessage(t, DID_{sender}, DID_{receiver})$—creates a non-data authorization message of type *t*.
- $m_{ShareData} \leftarrow createDataMessage('ShareData', DID_{sender}, DID_{receiver}, CID_{f^{CT}}, CID_{k^{CT}})$—creates a data authorization message of type *ShareData*.
- $m^{CT} \leftarrow encryptMessage(pk, m)$—encrypts an authorization message with the receiver's public key.
- $m \leftarrow decryptMessage(sk, m)$—decrypts an authorization message with a receiver's private key.

6. Data Functions:

- $f^{CT} \leftarrow encryptFile(k, f)$—encrypts file(s) with a session key.
- $f \leftarrow decryptFile(k, f^{CT})$—decrypts file(s) with the corresponding session key.
- $CID \leftarrow saveToIPFS(m^{CT} \mid DDO \mid f_k^{CT})$—saves data to IPFS and returns a CID. Authorization messages and files are encrypted before saving.
- $m^{CT} \mid DDO \mid f_k^{CT} \leftarrow retrieveFromIPFS(CID)$—retrieves data from IPFS using a CID.

4.2.2 *Algorithms*

In this section, we describe the algorithms used within DID authentication, VC verification and authorization. The algorithms are composed of the functions defined previously in Section 4.2.1. There are nine algorithms and are summarized Table 1.

Table 1. Table of algorithms used with the authentication and authorization schemes.

Algorithm Name	Scheme	Description
1: Create Public DID	DID Authentication	Create DIDs and DDOs for participants in the network.
2: Authenticate DID	DID Authentication	Verifier sends Subject a challenge to authenticate DID.
3: Create Private DID	VC Verification	Create a private DID for encrypting authorization messages and establishing private connections.
4: Request VC	VC Verification	Subject requests a VC from Issuer.
5: Verify VP	VC Verification	Verifier verifies signatures in VP.
6: Request Data	Authorization	DR requests private data from DM, owned by DO.
7: Ask Permission	Authorization	DM asks permission from DO to share private data with DR.
8: Allow Permission	Authorization	DO gives permission to DM to share private data with DR.
9: Share Data	Authorization	DM encrypts DO's private data and shares it with DR.
10: Retrieve Data	Authorization	DR retrieves private data and decrypts it.

Algorithm 1 creates DIDs and DDOs for all users n in the network N, then publishes them to the blockchain, which results in publicizing those DIDs. The first key pair and DID represents the wallet. First, sk and pk pairs are generated (line 2). Next, the DIDs and DDOs are created with each user's pk (line 3). The DDO is stored on IPFS and the CID_{DDO} is returned (line 4). Last, each DID and CID_{DDO} is published to the blockchain via the Identity contract (line 5).

Algorithm 1 Create Public DIDs

1: **for** $\boldsymbol{n} \in N$ **do**
2: $\boldsymbol{sk_n, pk_n} \leftarrow$ ***genKeyPair*(*seed, path*)**
3: $\boldsymbol{DID_n, DDO_n} \leftarrow$ ***createDID*($\boldsymbol{pk_n}$)**
4: $\boldsymbol{CID_{DDO_n}} \leftarrow$ ***saveToIPFS*($\boldsymbol{DDO_n}$)**
5: publishDID(**$\boldsymbol{DID_n, CID_{DDO_n}}$**)
6: **end for**

Algorithm 2 authenticates a DID using a challenge-response process (refer to Figure 14). Verifier n_V generates a random string, $challenge_{n_s}$, and sends it to Subject n_S (line 2–3). In response, $challenge_{n_s}$ must be encrypted by both n_V's public key and n_S's private key, then sent back to n_V. To do this, n_s looks up n_V's DID, DID_{n_V}, on the blockchain to get the CID to its document, $CID_{DDO_{n_V}}$ (line 4). With $CID_{DDO_{n_V}}$, n_V can retrieve DDO_{n_V} from IPFS and extract the public key to obtain pk_{n_V} (line 5–6). The challenge is encrypted by pk_{n_V} and sk_{n_S} to form $challenge_{n_S}^{CT}$, then sent back to n_s (line 7–8).

For decryption, $challenge_{n_S}^{CT}$ must be decrypted by both pk_{n_s} and sk_{n_V}. Thus, similar to lines 4–6, n_V performs the same functions to obtain pk_{n_S} (lines 9–11). The challenge can now be decrypted and verified that its content matches the original challenge that was sent to n_S.

Algorithm 2 Authenticate DID

1: Let $\boldsymbol{n_S, n_V} \in \boldsymbol{N}$
2: $\boldsymbol{challenge_{n_S}} \leftarrow$ ***generateChallenge*()** >Verifier
3: ***send*($\boldsymbol{DID_{n_V}, DID_{n_S}, challenge_{n_V}}$)**
4: $\boldsymbol{CID_{DDO_{n_V}}} \leftarrow$ ***lookupDID*($\boldsymbol{DDO_{n_V}}$)** > Subject
5: $\boldsymbol{DDO_{n_V}} \leftarrow$ ***retrieveFromIPFS*($\boldsymbol{CID_{DDO_{n_V}}}$)**
6: $\boldsymbol{pk_{n_V}} \leftarrow$ ***extractPublicKey*($\boldsymbol{DDO_{n_V}}$)**
7: $\boldsymbol{challenge_{n_S}^{CT}} \leftarrow$ ***encryptChallenge*($\boldsymbol{sk_{n_S}, pk_{n_V}, challenge_{n_S}}$)**
8: ***send*($\boldsymbol{DID_{n_S}, DID_{n_V}, challenge_{n_S}^{CT}}$)**
9: $\boldsymbol{CID_{DDO_{n_S}}} \leftarrow$ ***lookupDID*($\boldsymbol{DID_{n_S}}$)** > Verifier
10: $\boldsymbol{DDO_{n_S}} \leftarrow$ ***retrieveFromIPFS*($\boldsymbol{CID_{DDO_{n_S}}}$)**
11: $\boldsymbol{pk_{n_S}} \leftarrow$ ***extractPublicKey*($\boldsymbol{DDO_{n_S}}$)**
12: $\boldsymbol{challenge_{n_V}} \leftarrow$ ***decryptChallenge*($\boldsymbol{sk_{n_V}, pk_{n_S}, challenge_{n_S}^{CT}}$)**
13: verifyChallenge(**$\boldsymbol{challenge_{n_S}, challenge_{n_V}}$**)

Algorithm 3 sends a private DID to another user. Recall private DIDs are used to establish private communication and are not published on the blockchain (refer to Section 3.2.2). First, a key pair is generated by user n_i and will only by known to user n_j; the key pair represented as $sk^*_{n_i,\, n_j}$ and $pk^*_{n_i,\, n_j}$ (line 2). Private DID, $DID^*_{n_i,n_j}$ and $DDO^*_{n_i,n_j}$, are created from $pk^*_{n_i,\, n_j}$ (line 3). Before $DID^*_{n_i,n_j}$ can be sent to n_j, it must be encrypted with n_j's public key pk_{n_j}, which obtained by looking up DID_{n_j}, retrieving DDO_{n_j} from IPFS and extracting pk_{n_j} (lines 4–6). We substitute lines 4 to 6 with function ***retrievePublicKey*(*DID*)** for the rest of the algorithms. Once pk_{n_j} has been extracted from DDO_{n_j}, it is used to encrypt $DID^*_{n_i,n_j}$ and sent to n_j (line 6–8).

Algorithm 3 Send Private DID

1: Let $\boldsymbol{n_i, n_j \in N}$ where $\boldsymbol{i \neq j}$
2: $\boldsymbol{sk^*_{n_i,n_j}, pk^*_{n_i,n_j} \leftarrow genKeyPair(seed, path)}$
3: $\boldsymbol{DID^*_{n_i,n_j}, DDO^*_{n_i,n_j} \leftarrow createDID(pk^*_{n_i,n_j})}$
4: $\boldsymbol{CID_{DDO_{n_j}} \leftarrow lookupDID(DID_{n_j})}$
5: $\boldsymbol{DDO_{n_j} \leftarrow retrieveFromIPFS(CID_{DDO_{n_j}})}$
6: $\boldsymbol{pk_{n_j} \leftarrow extractPublicKey(DDO_{n_j})}$
7: $\boldsymbol{DID^{*CT}_{n_i} \leftarrow encryptPrivDID(pk_{n_j}, DID^*_{n_i})}$
8: $\boldsymbol{send(DID_{n_j}, DID^{*CT}_{n_i})}$

Algorithm 4 sends a VC to another user. Before Issuer n_I sends Subject n_S their verifiable credential VC_{n_S}, a private connection is initiated between two private DIDs, $DID^*_{n_S,n_I}$ and $DID^*_{n_I,n_S}$ (line 2). Once established, n_I sends VC_{n_S} to n_S's wallet (line 3). VC_{n_S} does not need to be encrypted since the private connection encrypts all communication between n_S and n_I.

Algorithm 4 Send Verifiable Credentials

1: Let $\boldsymbol{n_S, n_I \in N}$
2: $\boldsymbol{createPrivateConnection(DID^*_{n_S,n_I}, DID^*_{n_V,n_I})}$
3: $\boldsymbol{send(DID_{n_S}, VC_{n_I})}$

Algorithm 5 verifies the digital signatures (proofs) in a VP. For this algorithm, Subject n_S is having their VP verified by Verifier n_V. Recall from Section 3.2.2 a VP is derived from a VC and contains signatures from both n_S and Issuer n_I. First n_S creates VP_{n_S} from one of their credentials VC_{n_S} (line 2). A private connection is established between n_S and n_V before the VP is sent (line 3). Two signatures are then extracted from VP_{n_S}, one from n_S and the other from n_I (line 5). Last, $signature_{n_S}$ and $signature_{n_I}$ are verified using n_S's and n_I's public key (lines 6–8).

Algorithm 5 Verify Verifiable Presentation

1: Let $\boldsymbol{n_S, n_V, n_I \in N}$
2: $\boldsymbol{VP_{n_S} \leftarrow createVerifiablePresentation(VC_{n_S})}$ > Subject
3: $\boldsymbol{createPrivateConnection(DID^*_{n_S,n_V}, DID^*_{n_V,n_S})}$
4: $\boldsymbol{send(VP_{n_S}, DID_{n_V})}$
5: $\boldsymbol{signature_{n_S}, signature_{n_I} \leftarrow extractProof(VP_{n_S})}$
6: **for** $\boldsymbol{n \in [n_S, n_I]}$ **do**
7: $\quad \boldsymbol{pk_n \leftarrow retrievePublicKey(DID_n)}$
8: $\quad \boldsymbol{true \mid false \leftarrow verifyProof(signature_n, pk_n)}$
9: **end for**

Algorithm 6 is used by the DR to request private data managed by the DM and owned by the DO. DR, DM, and DO is represented as n_{DR}, n_{DM} and n_{DO} respectively. First, n_{DR} creates a *RequestData* authorization message $m_{RequestData}$ with $DID_{n_{DR}}$ and $DID_{n_{DO}}$ (line 2). n_{DR} then retrieves $pk_{n_{DM}}$ associated with $DID_{n_{DM}}$ in order encrypt $m_{RequestData}$ (line 3). $m^{CT}_{RequestData}$ is saved to IPFS and n_{DR} obtains $CID_{m^{CT}_{RequestData}}$. In the final step, n_{DR} submits $CID_{m^{CT}_{RequestData}}$ to the blockchain (line 6).

Algorithm 6 Request Data

1: Let $n_{DO}, n_{DR}, n_{DM} \in N$
2: $m_{RequestData} \leftarrow createMessage('RequestData', DID_{n_{DR}}, DID_{n_{DO}})$
3: $pk_{n_{DM}} \leftarrow retrievePublicKey(DID_{n_{DM}})$
4: $m^{CT}_{RequestData} \leftarrow encryptMessage(pk_{n_{DM}}, m_{RequestData})$
5: $CID_{m^{CT}_{RequestData}} \leftarrow saveToIPFS(m^{CT}_{RequestData})$
6: $TID_{RequestData} \leftarrow submitMessage('RequestDat', CID_{m^{CT}_{RequestData}})$

Algorithm 7 is used by the DM to ask the DO's permission to share private data with the DR. First, n_{DM} looks up $TID_{RequestData}$ to obtain $CID_{m^{CT}_{RequestData}}$ from the blockchain (line 2). n_{DM} then uses $CID_{m^{CT}_{RequestData}}$ to retrieve $m^{CT}_{RequestData}$ from IPFS and decrypts it with their secret key sk_{n_k}. Next, the n_{DO}'s and n_{DR}'s public DIDs are extracted from $m_{RequestData}$ (line 5) and n_{DM} uses the DIDs to create $m_{AskPermission}$ (line 6). The authorization message is encrypted with $pk^{*}_{n_{DO},n_{DM}}$, extracted from $DID^{*}_{n_{DO},n_{DM}}$ (line 7). The encrypted message, $m^{CT}_{AskPermission}$, is saved to IPFS and $CID_{m^{CT}_{AskPermission}}$ is returned (line 8). Last, n_{DM} submits $CID_{m^{CT}_{AskPermission}}$ to the blockchain and obtains $TID_{AskPermission}$ (line 9).

Algorithm 7 Ask Permission

1: Let $n_{DO}, n_{DR}, n_{DM} \in N$
2: $CID_{m^{CT}_{RequestData}} \leftarrow lookUpTID(TID_{RequestData})$
3: $m^{CT}_{RequestData} \leftarrow retrieveMessageFromIPFS(CID_{m^{CT}_{RequestData}})$
4: $m_{RequestData} \leftarrow decryptMessage(sk_{n_{DM}}, m^{CT}_{RequestData})$
5: $DID_{n_{DO}}, DID_{n_{DR}} \leftarrow extractDID(m_{RequestData})$
6: $m_{AskPermission} \leftarrow createMessage('AskPermission', DID_{n_{DR}}, DID_{n_{DO}})$
7: $m^{CT}_{AskPermission} \leftarrow encryptMessage(pk^{*}_{n_{DO},n_{DM}}, m_{AskPermission})$
8: $CID_{m^{CT}_{AskPermission}} \leftarrow saveToIPFS(m^{CT}_{AskPermission})$
9: $TID_{AskPermission} \leftarrow submitMessage('AskPermission', CID_{m^{CT}_{AskPermission}})$

Algorithm 8 is used by the DO to give the DM permission to share their private data with a DR. First, n_{DO} looks up the transaction receipt $TID_{RequestData}$ sent from n_{DM} and obtains $CID_{m^{CT}_{RequestData}}$ (line 2). n_{DO} uses the CID to retrieve $m^{CT}_{RequestData}$ from IPFS (line 3) and decrypts it with $sk^{*}_{n_{DO},n_{DM}}$ (line 4). Next, n_{DO}'s and n_{DR}'s public DID are extracted from $m_{AskPermission}$; n_{DO} now knows who is requesting their private data. In this algorithm, we assume n_{DO} allows permission, as opposed to rejects permission, and creates the *AllowPermission* authorization message $m_{AllowPermission}$ (line 6). n_{DO} then retrieves pk_{DM} (line 7) and uses it to encrypt $m_{AllowPermission}$ (line 8). The encrypted messaged $m^{CT}_{AllowPermission}$ is saved to IPFS, and n_{DO} obtains $CID_{m^{CT}_{AllowPermission}}$. The last line, $CID_{m^{CT}_{AllowPermission}}$ is submitted to the blockchain.

Algorithm 8 Allow Permission

1: Let $n_{DO}, n_{DR}, n_{DM} \in N$
2: $CID_{m^{CT}_{AskPermission}} \leftarrow lookUpTID(TID_{AskPermission})$
3: $m^{CT}_{AskPermission} \leftarrow retrieveMessageFromIPFS(CID_{m^{CT}_{AskPermission}})$
4: $m_{AskPermission} \leftarrow decryptMessage(sk^{*}_{n_{DO},n_{DM}}, m^{CT}_{AskPermission})$
5: $DID_{n_{DO}}, DID_{n_{DR}} \leftarrow extractDID(m_{AskPermission})$
6: $m_{AllowPermission} \leftarrow createMessage('AllowPermission', DID_{n_{DO}}, DID_{n_{DR}})$
7: $pk_{n_{DM}} \leftarrow retrievePublicKey(DID_{n_{DM}})$
8: $m^{CT}_{AllowPermission} \leftarrow encryptMessage(pk_{n_{DM}}, m_{AllowPermission})$
9: $CID_{m^{CT}_{AllowPermission}} \leftarrow saveToIPFS(m^{CT}_{AllowPermission})$
10: $TID_{AllowPermission} \leftarrow submitMessage('AllowPermission', CID_{m^{CT}_{AllowPermission}})$

Algorithm 9 is used by the DM to share the DO's private data with the DR. First, n_{DM} looks $TID_{AllowPermission}$ on the blockchain to obtain $CID_{m^{CT}_{AllowPermission}}$ (line 2). n_{DM} then uses the CID to retrieve $m^{CT}_{AllowPermission}$ (line 3) and decrypts the authorization message with $sk_{n_{DM}}$ (line 4). n_{DO}'s and n_{DR}'s public DID is extracted from $m_{AllowPermission}$ (line 5). Next, n_{DM} generates a random session key k (line 6) and encrypts n_{DO}'s data file(s) f_{DO} (line 7). n_{DM} then retrieves $pk^{*}_{n_{DR},n_{DM}}$ from $DID^{*}_{n_{DR},n_{DM}}$ and encrypts k (line 8). f^{CT}_{DO} and k are saved to IPFS and n_{DM} obtains $CID_{f^{CT}}$ and $CID_{k^{CT}}$ respectively (lines 9 and 10). With all the required information, n_{DM} creates a *ShareData* authorization message (line 11) and encrypts the message with $pk^{*}_{n_{DR},\, n_{DM}}$ (line 12). n_{DM} saves the message to IPFS (line 13) and submits $CID_{m^{CT}_{ShareData}}$ to the blockchain (line 14). In the last line, n_{DM} sends blockchain transaction $TID_{AllowPermission}$ to DR using DID_{DR} (line 15).

Algorithm 9 Share Data

1: Let $n_{DO}, n_{DR}, n_{DM} \in N$
2: $CID_{m^{CT}_{AllowPermission}} \leftarrow lookUpTID(TID_{AllowPermission})$
3: $m^{CT}_{AskPermission} \leftarrow retrieveMessageFromIPFS(CID_{m^{CT}_{AllowPermission}})$
4: $m_{AllowPermission} \leftarrow decryptMessage(sk_{DM}, m^{CT}_{AllowPermission})$
5: $DID_{n_{DO}}, DID_{n_{DR}} \leftarrow extractDID(m_{AllowPermission})$
6: $k \leftarrow generateEncryptionKey()$
7: $f^{CT} \leftarrow encryptData(k, f)$
8: $k^{CT} \leftarrow encryptKey(pk^{*}_{n_{DR},n_{DM}}, k)$
9: $CID_{k^{CT}} \leftarrow saveToIPFS(k^{CT})$
10: $CID_{f^{CT}} \leftarrow saveToIPFS(f^{CT})$
11: $m_{ShareData} \leftarrow createMessage('ShareData', DID_{n_{DO}}, DID_{n_{DR}}, DID_{k^{CT}}, DID_{f^{CT}})$
12: $m^{CT}_{ShareData} \leftarrow encryptMessage(pk_{DID^{*}_{DR,DM}}, m_{ShareData})$
13: $CID_{m^{CT}_{ShareData}} \leftarrow saveToIPFS(m^{CT}_{ShareData})$
14: $TID_{ShareData} \leftarrow submitMessage('SendData', m^{CT}_{ShareData})$
15: $send(DID_{DM}, DID_{DR}, TID_{ShareData})$

Algorithm 10 is used by the DR to retrieve DO's data from IPFS. First, n_{DR} look ups $TID_{ShareData}$ from the blockchain to obtain $CID_{m^{CT}_{ShareData}}$ (line 2), then uses the CID to retrieve m^{CT}_{Share} from IPFS (line 3). Next, n_{DR} decrypts m^{CT}_{Share} with $sk^{*}_{n_{DR},n_{DM}}$ (line 4). The CIDs are extracted from $m_{ShareData}$ (line 5) and used to retrieve the encrypted session key k^{CT} and file(s) f^{CT} (lines 6 and 7). Before f^{CT} can be decrypted, k^{CT} must be decrypted first using $sk^{*}_{n_{DR},n_{DM}}$ to reveal k (line 8). Last, f^{CT} is decrypted with k and n_{DR} is now in possession of n_{DO}'s private data (line 9).

Algorithm 10 Retrieve Data

Let $n_{DO}, n_{DR}, n_{DM} \in N$
$CID_{m^{CT}_{AllowPermission}} \leftarrow lookUpTID(TID_{ShareData})$
$m^{CT}_{ShareData} \leftarrow retrieveFromIPFS(CID_{m^{CT}_{ShareData}})$
$m_{ShareData} \leftarrow decryptMessage(sk_{DID^{*}_{DR,DM}}, m^{CT}_{ShareData})$
$CID_{k^{CT}}, CID_{f^{CT}} \leftarrow extractDID(m_{ShareData})$
$k^{CT} \leftarrow retrieveFromIPFS(CID_{k^{CT}})$
$f^{CT} \leftarrow retrieveFromIPFS(CID_{f^{CT}})$
$k \leftarrow decryptKey(sk^{*}_{n_{DR}, n_{DM}}, k^{CT})$
$f \leftarrow decryptData(k, f^{CT})$

4.3 *Real World Use Case Scenario*

In this scenario, we have Alice who is experiencing chest pain and decides to make an appointment to see a physician. In the past she has gone to Hospital A (H_A) but has recently enrolled in medical insurance at her workplace which offers a different provider. Prompting Alice to register as new patient at Hospital B (H_B). To register, she uses a driver's license VC, obtained from the Department of Motor Vehicles (DMV), to prove her identify. Alice presents a VP, and H_B verifies the signatures belong to both Alice and the DMV. In Section 5.1.4, we later show how this process was implemented in the prototype.

During her visit, the physician at H_B recommends a new treatment for Alice's chest pain, but first needs to review historic medical history to ensure the treatment will not have ill side-effects. H_B requests Alice's health records from H_A using a DeAuth application. H_A can only share her private EHR upon Alice's permission. In this scenario, Alice is the DO, H_A is the DM and H_B is the DR, thus H_A can only share Alice's EHR with H_B with her permission. The scenario is represented in Figures 18 and 19. We make the following assumptions prior to the scenario taking place:

- All users have a digital wallet that is able to perform all functions described in Section 4.2.2.
- Users have an internet connection that allows the wallet to interact with smart contracts on the blockchain, IPFS and other users' wallet.
- All users have a created a public DID published to the blockchain.
- Identity authorities, such the DMV, participate in decentralized identity and have infrastructure to issue and verify VCs.
- H_A, being the DM, has deployed an Authorization Contract and is monitoring the contract for transactions.

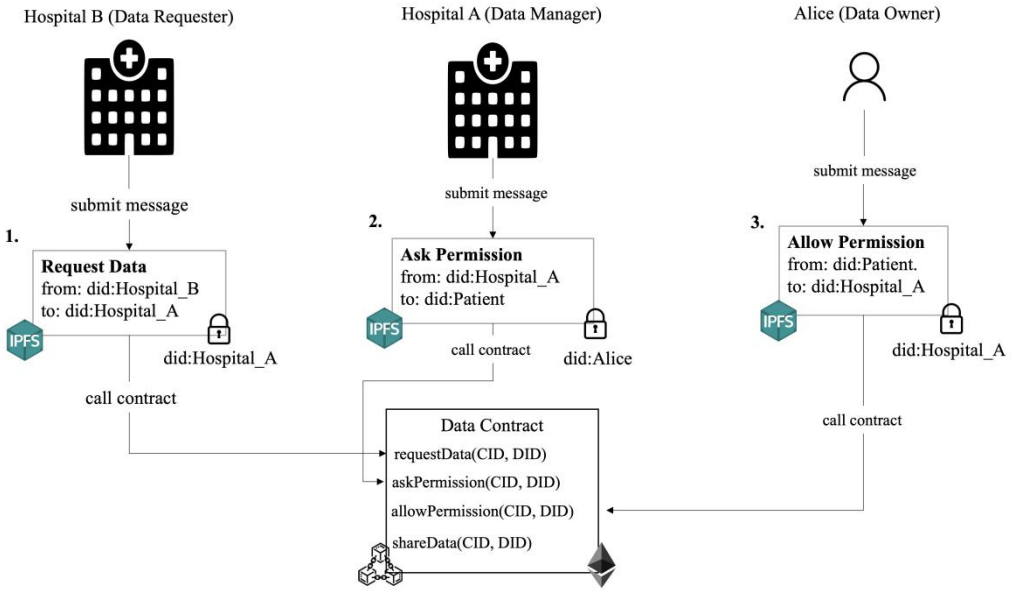


Figure 18. First three steps of EHR sharing use case scenario: (1) Request Data, (2) Ask Permission and (3) Allow Permission.

In Figure 18, step 1, H_B submits a *RequestData* authorization message to obtain Alice's EHR (refer to Algorithm 6). To do so, H_B first creates $m_{RequestData}$. The authorization message is encrypted with pk_{H_A}, which is extracted from DID_{H_A}, then saved to IPFS. H_B completes the request by submitting $CID_{m^{CT}_{RequestData}}$ to the blockchain.

In Figure 18, step 2, H_A submits a *AskPermission* authorization message to get Alice's permission to share her EHR with H_B (refer to Algorithm 7). H_A first looks ups $TID_{ReqeuestData}$ to retrieve $CID_{m^{CT}_{RequestDATA}}$, then uses the CID to retrieve $m^{CT}_{RequestData}$. H_A decrypts $m^{CT}_{RequestData}$ with sk_{H_A} and now knows H_B is requesting Alice's EHR. H_A creates $m_{AskPermission}$ and encrypts with a pk^*_{Alice,H_A}, which was extracted from $DID^*_{Alice,\ H_A}$; recall using $DID^*_{Alice,\ H_A}$ is to ensure Alice's anonymity on the network (refer to Section 4.1.3). The encrypted message, $m^{CT}_{AskPermission}$, is saved to IPFS. H_A submits the CID of the encrypted message, $CID_{m^{CT}_{AskPermission}}$, to the blockchain and obtains $TID_{AskPermission}$. To complete the step, H_A creates a private connection with Alice and sends $TID_{AskPermission}$ to her.

In Figure 18, step 3, Alice submits an *AllowPermission* authorization message to give H_A permission to share her EHR with H_B (refer to Algorithm 8). Alice first looks up $TID_{AskPermission}$ on the blockchain and obtains $CID_{m^{CT}_{AskPermission}}$. Next, she retrieves $m^{CT}_{AskPermission}$ from IPFS and decrypts the message with sk^*_{Alice,H_A}. Alice now becomes aware that H_A is requesting permission to share her EHR with H_B. Alice then creates $m_{AllowPermission}$ and encrypts it with pk_{H_A}. Alice saves $m^{CT}_{AllowPermission}$ to IPFS and obtains $CID_{m^{CT}_{AllowPermission}}$. Alice then submits the CID to the blockchain.

In Figure 19, step 4, H_A submits a *ShareData* authorization message to share Alice's EHR H_B (refer to Algorithm 9). H_A looks up $TID_{AllowPermision}$ from the blockchain to obtain $CID_{m^{CT}_{AllowPermission}}$. H_A uses the CID to retrieve $m^{CT}_{AllowPermission}$. H_A decrypts the message with sk_{H_A} and now knows that Alice has granted them permission to share her EHR with H_B. H_A prepares to encrypt Alice's EHR files f by generating a session key k. H_A uses k to encrypt f, and after encrypts k with $pk^*_{H_B,H_A}$. H_A uploads f^{CT} and k^{CT} to IPFS and obtains $CID_{f^{CT}}$ and $CID_{k^{CT}}$. With all the necessary information, H_A can create $m_{ShareData}$ and encrypt it with $pk^*_{H_B,H_A}$. H_A saves $m^{CT}_{ShareDATA}$ to IPFS and obtains $CID_{m^{CT}_{ShareData}}$. H_A submits $CID_{m^{CT}_{ShareData}}$ to the blockchain and obtains $TID_{ShareData}$. To end this step, H_A creates a private connection with H_B and sends $TID_{ShareData}$ to H_B.

In Figure 19, the 5th and final step, H_B retrieves the encrypted session key and files from IPFS (refer to Algorithm 10). H_B first looks up $TID_{ShareData}$ from the blockchain to obtain $CID_{m^{CT}_{ShareData}}$. H_B then uses $CID_{m^{CT}_{ShareData}}$ to retrieve $m^{CT}_{ShareDATA}$ from IPFS. Next, H_B decrypts $m^{CT}_{ShareData}$ with $sk^*_{H_B,H_A}$ to reveal $m_{ShareData}$. $CID_{f^{CT}}$ and $CID_{k^{CT}}$ are extracted from $m_{ShareData}$.

The CIDs are used to retrieve f^{CT} and k^{CT} from IPFS. H_B then decrypts k^{CT} with $sk^*_{H_B,H_A}$ to reveal k. Last, H_B uses k to decrypt f^{CT} and can now read Alice's EHR.

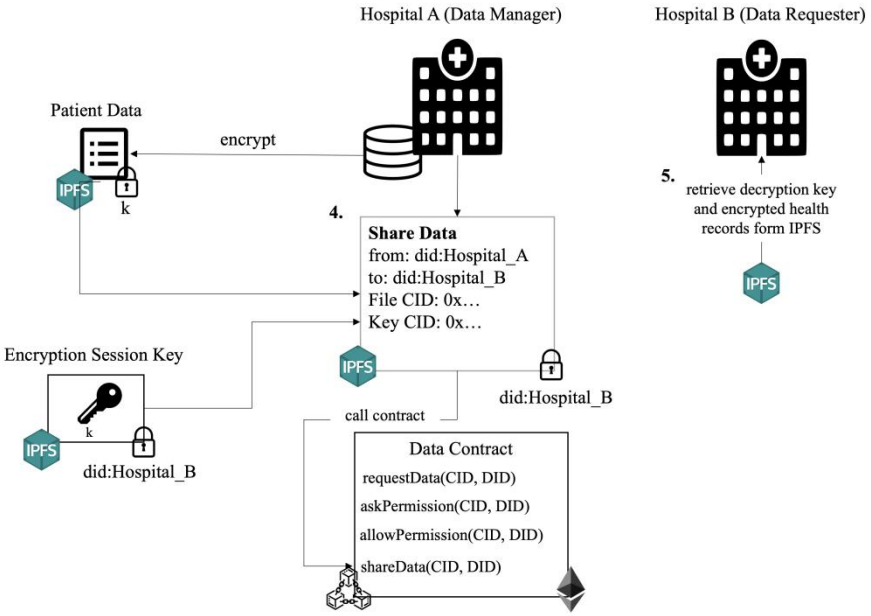


Figure 19. Last two steps of EHR sharing use case: (4) Share Data and (5) Retrieve Data.

5. Prototype Implementation

In this section, we provide an overview of the tools and methods that were used in the development of a prototype DeAuth application. The first iteration relied on a centralized service to simulate the functionality of a blockchain, as well as data storage. In the second iteration, decentralized services were integrated to enable the application to operate in a decentralized manner. The following technologies were used to develop the application:

- Next.js (https://nextjs.org/)—a React framework to create to web applications.
- Firebase (https://firebase.google.com/)—Google's back end as a service and cloud storage service.
- Web3.Storage (https://web3.storage/)—a service to store files accessible via IPFS.
- Ganache (https://archive.trufflesuite.com/ganache/)—a local Ethereum blockchain node.

5.1 *Application Development*

5.1.1 *Wallet User Interface*

The bulk of the development efforts were dedicated to designing and implementing the wallet interface and its associated functionalities. The wallet user interface (UI) was built with Next.js and segmented into three display sections: messaging, DIDs and VCs, and a dedicated section to display details such as VC signatures, see Figure 20a–c.

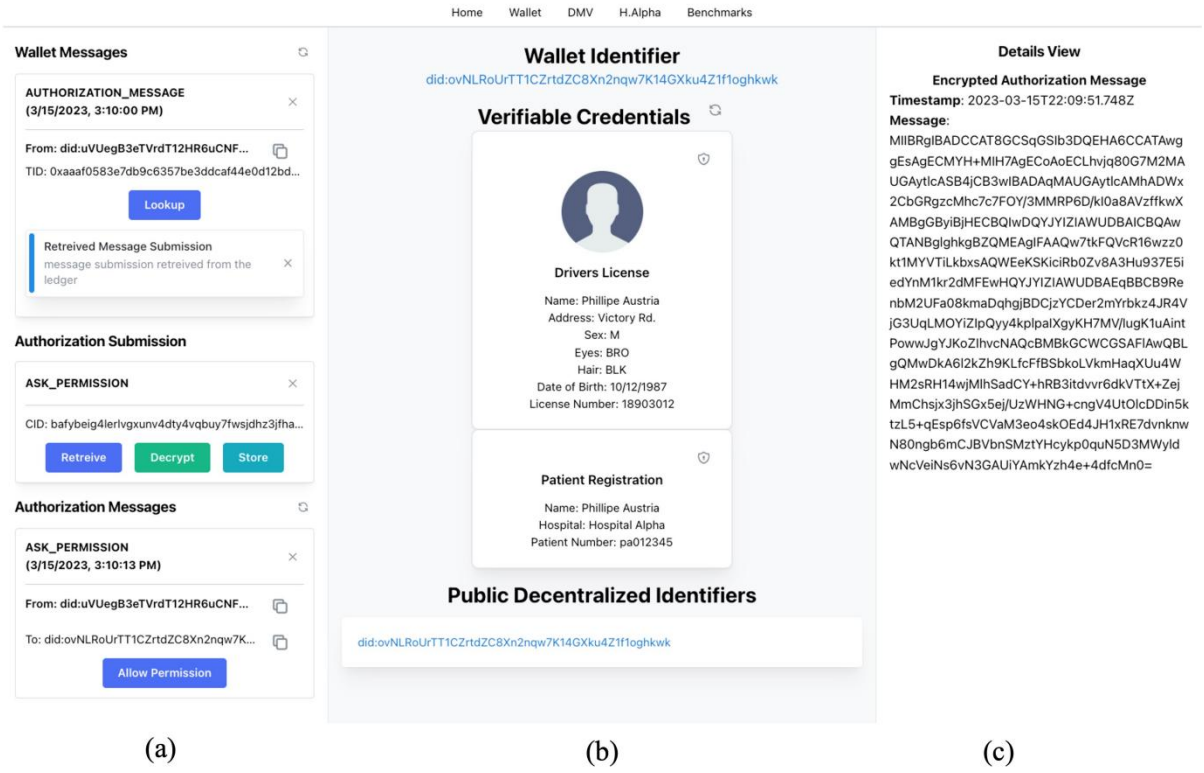


Figure 20. Wallet UI with three sections: (**a**) wallet, authorization submissions and authorization messages, (**b**) VCs and DIDs and (**c**) details view.

The messaging section of the UI displays messages sent from other wallets (i.e., wallet messages), authorization message submissions to the blockchain and the authorization messages themselves. The wallet messages serve to alert the owner that another user's wallet has requested DID or VC verification; they are not to be confused with authorization messages. Furthermore, wallet messages notify the wallet owner when an authorization message has been submitted to the blockchain and are the intended recipient. Each message is associated with specific actions that can be undertaken in response. For instance, in Figure 21a the user can either accept or reject a DID authentication request message. The complete catalogue of wallet messages created in the prototype is itemized in Table 2.

Table 2. List of wallet messages, their descriptions, and their follow up actions.

Wallet Message Type	Description	Follow-Up Actions
DID_AUTH_REQUEST	another wallet requested DID authentication request	accept or reject
ACCEPT_DID_AUTH_REQUEST	DID authentication request accepted	send challenge
DID_CHALLENGE	DID authentication challenge sent	respond to challenge
DID_CHALLENGE_RESPONSE	DID challenge decrypted	verify response
ID_CREDENTIAL_REQUEST	another wallet requested a VC	select a VC
ID_PRESENTATION	VP generated and signed	verify VP
AUTHORIZATION_MESSAGE	authorization message submitted to blockchain	look up TID_m

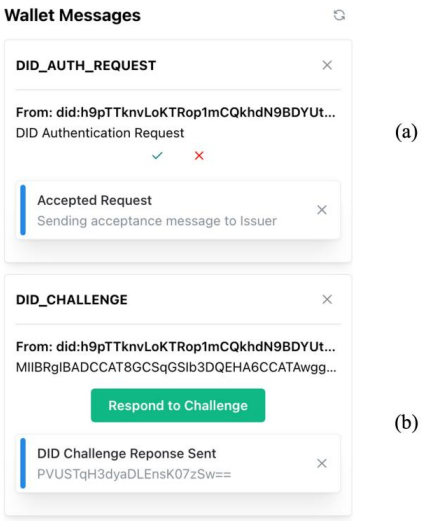


Figure 21. Example wallet messages: (**a**) DID_AUTH_RQUEST—the wallet was sent a DID authentication request and accepted the request. (**b**) DID_CHALLENGE—the wallet was sent an authentication challenge and responds to the requester to validate the DID.

The authorization submissions section of the UI displays the authorization message submissions that have been looked up from the blockchain with a TID and are waiting to be retrieved from IPFS. Upon being designated as the receiver of an authorization message, the wallet is notified with a wallet message. The TID associated with the authorization message submission is looked up on the blockchain (Figure 22a) and the CID is shown in authorization submission section (Figure 22b). For prototype and demonstration purposes, we manually trigger each step with a button click to retrieve the authorization message. Those steps are: (1) Retrieve the authorization message from IPFS, (2) Decrypt the message with a private key, and (3) Store the decrypted message within the wallet. In a production application, these steps would presumably automatically when a submission is made.

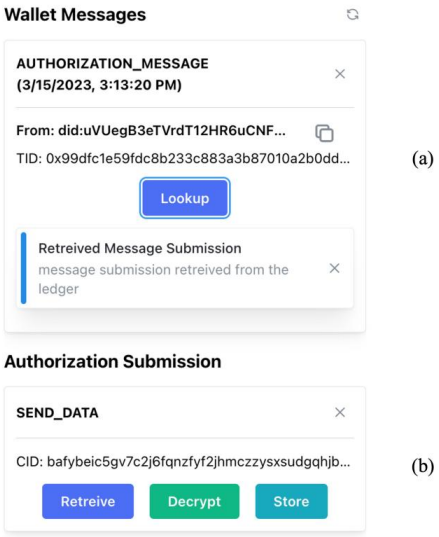


Figure 22. Screenshot of a *SendData* authorization message. (**a**) The wallet message shows the submission TID, which is used to lookup the IPFS CID on the blockchain. (**b**) Using a CID, the wallet can retrieve the message from IPFS. After, the wallet decrypts and stores the message.

The authorization messages section in the wallet UI displays plain text authorization messages after they have been retrieved from IPFS and decrypted. Each authorization message type has a specific set of actions that can be taken in

response. For example, when a *RequestData* is received, a button is displayed asking user, the DM, to ask permission from the DO to share their data. Figure 23 comprehensively catalogs the various types of authorization messages, along with their associated actions.

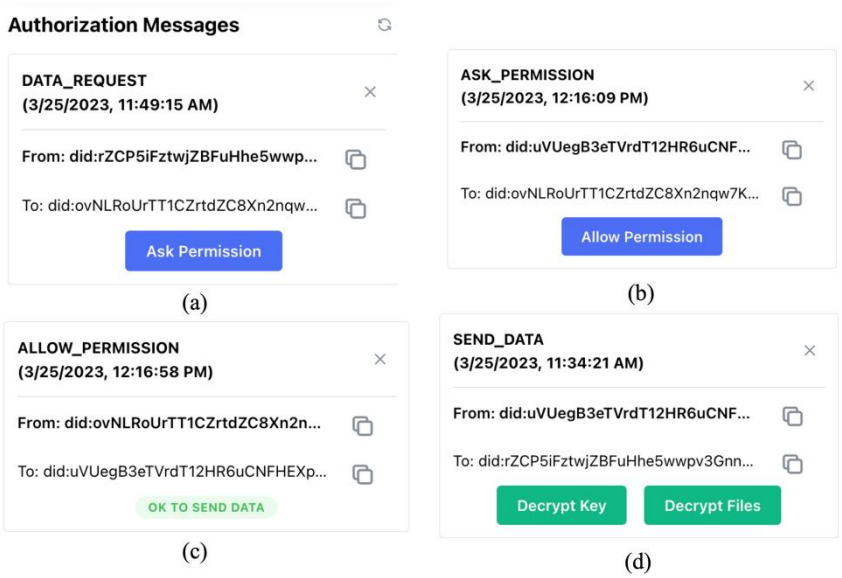


Figure 23. Various authorization messages and follow up actions displayed in the UI: (**a**) Data Request, (**b**) Ask Permission, (**c**) Allow Permission and (**d**) Share Data.

The DID/VC section of the wallet UI (Figure 20b) displays the DIDs and VCs that are associated with the wallet. Upon the creation of a new wallet, a key pair is generated, and the public key is encoded in base58 to yield the wallet's public DID. Our prototype did not include the capability to create new public or private DIDs, but would be a necessary feature in a production application.

5.1.2 *Wallet Messages*

Our prototype did not encompass the development of a fully functional P2P wallet messaging system, as this aspect was not the central focus of the research. Nevertheless, we save this feature for future iterations of the application, drawing upon the specifications and protocols outlined in the DIDComm (https://identity.foundation/didcomm-messaging/spec/) framework to provide a secure, private communication methodology built atop the decentralized design of DIDs. In lieu of a functional messaging system, we simulated messaging activity by storing messages in a JSON file. These wallet messages are exchanged between wallets for the purpose of DID authentication, as well as for transmitting TIDs associated with authorization message submissions.

5.1.3 *Blockchain and Data Storage*

Initially, we developed a prototype leveraging Google Firebase, which served a dual role in the capacity of both the blockchain and the storage mechanism for files. Within this context, DDOs and authorization messages were also saved to Firebase. To facilitate these operations, a Firebase API (https://firebase.google.com/docs/storage) was used to implement of calls to store and retrieve data from Firebase. In essence, these calls simulated smart contract calls to the blockchain.

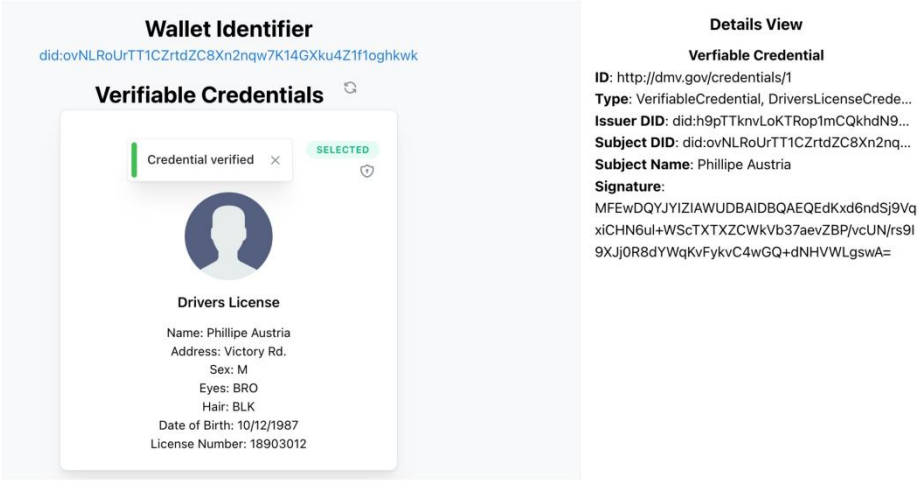
In the second iteration of our prototype, we eliminated Firebase and replaced its calls with smart contracts functions, while employing Web3.Storage for storing data to IPFS. The Identity and Authorization contracts were designed and compiled using the Remix (https://remix.ethereum.org/) editor. For initial smart contract testing we used Ganache, a local Ethereum node. Thereafter, we conducted testing on the live Goerli Ethereum test network.

5.1.4 *Obtaining a Verifiable Credential*

Upon wallet creation, it does not initially contain a VC. To acquire a VC, a driver's license VC was simulated to be obtained from the DMV. In this scenario, it is assumed that after a subject applies for a driver's license, they register their DID with the DMV. The DMV then allows users to obtain a digital version of their driver's license, which is the VC. Figure 24 shows a screen of the prototype where the subject enters their wallet DID to receive VC. After the DID is submitted in the form, the DID owner will receive a wallet message to begin the DID authentication process.

Figure 24. Screenshot of the DMV form to obtain a driver's license VC.

Upon obtaining the VC, it is displayed in the central portion of the wallet UI as seen in Figure 25a. The user can view specific details of the VC by clicking on it, which will be displayed on the right-hand side of the wallet UI, see Figure 25b. These details include the digital signature that was generated by the DMV's private key. Additionally, we implement the functionality to request VC verification by the Issuer. By clicking the lock symbol on the top right of the credential in Figure 25a, sends a request to the DMV's wallet to validate the VC's signature.



(a) (b)

Figure 25. Screenshot of: (**a**) the verified driver's license VC from the DMV and (**b**) the VC detail which includes the issuer's signature.

5.1.5 *EHR Implementation*

In the prototype, we implemented the use case scenario for sharing EHRs (Section 4.3). This implementation comprises three primary functionalities, which are illustrated in Figure 26: (a) patient registration, (b) patient data request and (c) sharing patient EHR to another hospital.

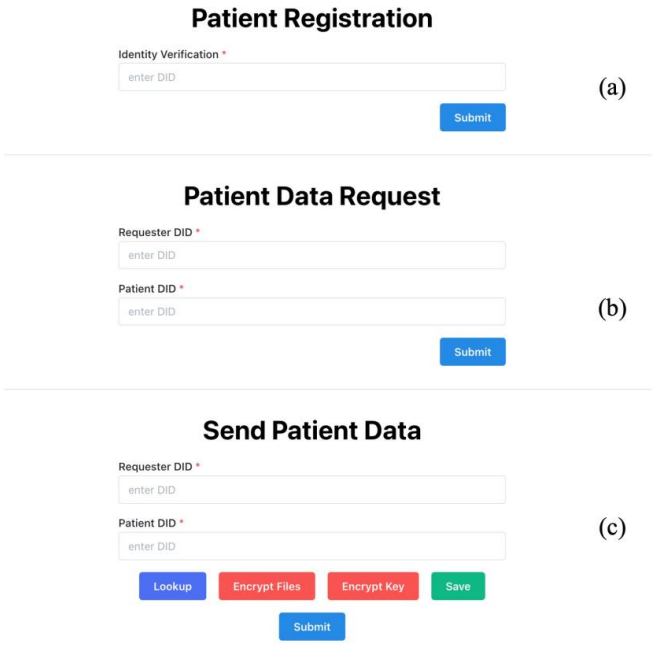


Figure 26. Screenshot of the various forms: (**a**) patient registration, (**b**) patient data request and (**c**) sharing patient EHRs.

Like Alice in Section 4.3, a user (subject) registers with H_A as a patient and then H_B requests the patient's EHR from H_A. The user first registers as a patient with H_A and obtains a second VC as proof of being a patient. While real-life registration generally requires the patient to provide detailed medical information, in our prototype, only the identity needs to be verified. After submitting their DID, the patient receives a message to prove their identity with a VC. The patient selects their driver's license VC, generates a VP, and sends the VP to H_A, see Figure 27. H_A validates both patient's signature and the DMV's signature. Once the VP is validated, H_A sends the patient a new VC.

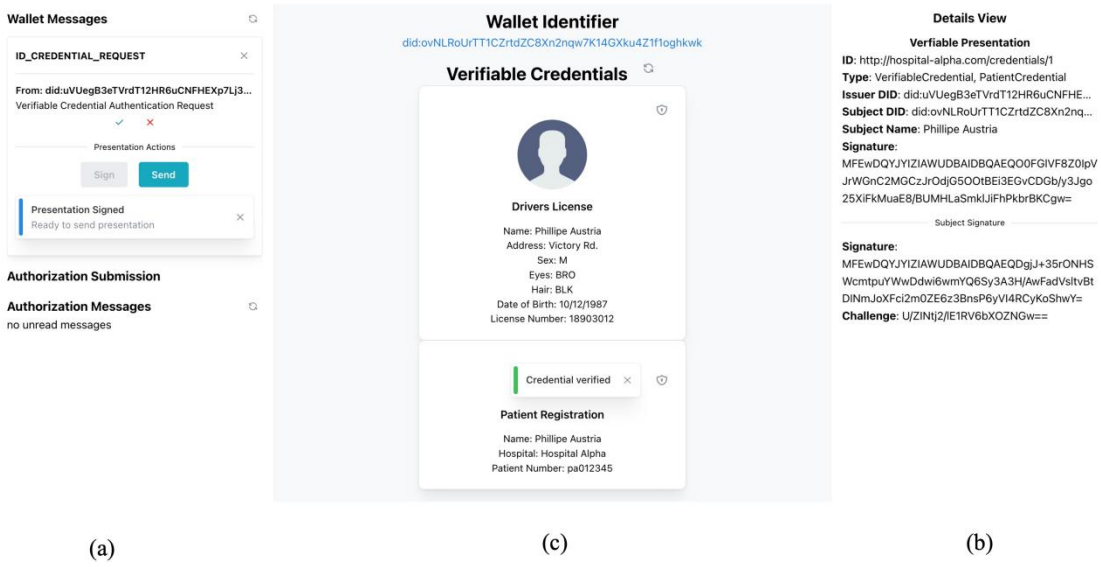


Figure 27. Screenshot of views when receiving a VC. (**a**) Patient signs and sends a driver's license VP. (**b**) The Details View shows the patient's digital signature along with the original Issuer's signature. (**c**) After signature validation, H_A sends the patent a VC which confirms they are a now a registered patient.

5.1.6 *Libraries and Source Code*

Table 3 provides an overview of the important libraries and their corresponding versions utilized in the development of the DeAuth prototype. These libraries played a crucial role in the functionality of the application, such as the UI design and the integration of blockchain and storage mechanisms. Additionally, numerous other libraries were utilized for the infrastructure and organization of the application. All these libraries can be searched and accessed through npmjs.com (https://www.npmjs.com/). The source code for the smart contracts are in the Appendixes Figures A1 and A2; the rest of the protype source code is hosted on our Gitlab repository (https://gitlab.com/paustria/unlv-dissertation/).

Table 3. Main libraries for the DeAuth prototype.

Library Name	Version	Description
abi-decoder	2.4.0	helper library to decode smart contract call input
base-58	0.0.1	provides helper functions to encode and encode in base58
firebase	9.1.16	api package to interact with Google Firebase
ganache	7.7.7	api package to interact with Ganache Ethereum local node
hdkey	2.0.1	hierarchical deterministic key library to generate BIP32 compliant wallet keys
truffle/hdwallet-provider	2.1.8	library used to create a Ethereum provider that auto-signs transactions
virgo-crypto	4.2.2	a cryptography library used for performing encryption
web3.js	1.8.2	a collection of libraries to interact with an Ethereum node

6. Security Analysis

In this section, we examine the security risks and concerns associated with the primary components of DeAuth, focusing specifically on the blockchain and wallet. Additionally, we offer insights into data encryption and storage reliability when using IPFS.

6.1 *Blockchain Consensus Mechanism*

DeAuth relies on the blockchain as the fundamental component and security layer of the system. As such, it inherits the security concerns that are inherent in blockchain technology. There has been extensive research conducted on the security of blockchain [58, 59]. The 51% attack is one of the most significant security threats to blockchains and is also relevant to DeAuth [60]. The approach taken to carry out such an attack depends on the consensus mechanism employed by the blockchain.

The DeAuth prototype was developed using the Ethereum blockchain, which has recently transitioned from the PoW consensus mechanism to the Proof of Stake (PoS) mechanism. However, the risk of a 51% attack persists as it does with PoW, albeit with higher economic risks for the attacker(s) [61]. To control 51% of the staked ETH, the attackers would need to possess a considerable amount of Ether (ETH), which is held by validator nodes. As of September 2023, there were 806,759 validator nodes with a combined 24,619,309 ETH staked. To conduct a 51% attack, an attacker would need own 51% of the staked ETH, which is valued more than $20 billion based on an ETH price of $1636 per ETH.

Moreover, the PoS consensus mechanism provides an opportunity for the community to mount a counterattack against a 51% attack [61]. In the event of such an attack, honest validator nodes can choose to continue building on the minority chain and disregard the attacker's fork while encouraging other nodes to do the same. They can also forcibly remove the attacker from the network, resulting in the destruction of their stacked ETH, which creates an even stronger economic disincentive for an attacker to launch a 51% attack.

6.2 *Wallet Vulnerabilities*

In DeAuth, wallets and their corresponding private keys are stored on the device owned by the user, giving them full control over their identity. However, this approach entails both security and privacy benefits, as well as the responsibility of protecting the private keys from theft. Losing the private keys could lead to the unauthorized control of DIDs and VCs,

a situation akin to identity theft. If an attacker steals the mnemonic phrase used to generate the private keys, they would be able to access all the private keys. Additionally, since DeAuth's wallet contains blockchain private keys for signing transactions, an attacker with access to the keys would have the ability to spend any currency associated with those keys.

The immutability of DIDs on the blockchain implies that revocation of a DID is a challenging task, and for this reason, revocation mechanisms have been proposed [47]. These mechanisms allow issuers to publish a revocation status for a particular DID, which is verified prior to validating its proof. However, this approach presents certain drawbacks, such as the potential for the revocation list to become unwieldy and have negative impacts on system performance. To address these limitations, current research efforts have proposed space-efficient and high-performance implementation solutions [32].

The theft of a pneumonic seed phrase can have more severe consequences than the theft of a few keys, as the attacker would be able to generate all the private keys of the wallet. Thus, it is imperative to safeguard the seed and only share it with trusted peers for backup purposes. Additionally, forgetting the seed can be equally detrimental; if the wallet is deleted and the seed is forgotten, the owner will lose complete access to the private keys. To mitigate this risk, wallet providers today encourage seed phrase owners to write down the phrase and make it an explicit step before using the wallet. Recent research has also proposed using the Shamir Secret Sharing algorithm to recover forgotten keys [62, 63].

6.3 *Encryption and Reliability*

Data is not encrypted on IPFS by default, which was intentional by its creator to prevent application developers from being limited due to a lack of modularity, flexibility, and futureproofing [64]. However, in DeAuth, authorization messages are encrypted using RSA encryption, and files are shared with AES encryption. Both RSA and AES are widely used and secure encryption algorithms that are approved by NIST [65, 66]. The strength of the encryption depends on the length of the key used, but the security of the system ultimately relies on the proper management of the keys themselves [67]. In DeAuth, when a new DID is created by the wallet, the private key of the DID is used as a seed to generate an RSA key pair. Therefore, it is crucial to keep the wallet's private keys secure.

There is an additional consideration that relates to storage network reliability in DeAuth. Although IPFS serves as the protocol for sharing files between nodes, the protocol alone does not determine metrics such as data redundancy, durability, and availability. These are common storage metrics included in Service Level Agreements of storage providers such as Amazon S3, Dropbox, and Google Drive. A study conducted in [68] found that Sia, one of the earliest blockchain-based storage services, exhibited similar redundancy, durability, and availability compared to centralized cloud services. That said, these metrics are primarily dependent on the storage network architecture and implementation, and not the blockchain itself.

For our prototype, we used Web3.Storage, a service that uses both IPFS and Filecoin, a blockchain based storage network [69]. Web3.Storage provides public gateway and hosts multiple IPFS nodes to ensure availability and redundancy, then utilizes the Filecoin network for storage durability. The public gateway simply allows users to IPFS without having to install IPFS software and self-host a node. Information regarding the number of IPFS nodes hosted by Web3.Storage were not disclosed in their documentation and thus storage metrics are unknown. To achieve a level of redundancy that is comparable to Google Cloud, Web3.Storage would need to replicate data a minimum of two times across their nodes.

In DeAuth, it is recommended files retrieved from IPFS are stored locally or to personal storage network. That way the user has their own copy and do not need to continuously fetch the same files from IPFS. Files saved to local storage would be decrypted, but remain encrypted on IPFS. Furthermore, while the authorization messages are stored on IPFS, it is also recommended to store them locally for quick access when needed.

7. Prototype Results and Analysis

The purpose of this section is to offer a comparative analysis of the primary operations involved in DeAuth, as implemented in two distinct prototype versions. Specifically, we compare a centralized version that employs Firebase, with a decentralized version that utilizes Ethereum and Web3.Storage. To achieve this goal, we present a detailed examination

of both the time performance and cost results for each prototype. For additional performance testing, we hosted a local IPFS. Table 4 shows the operations that were measured and analyzed. By comparing the performance metrics of the two prototypes, we aim to provide valuable insights into the relative advantages and disadvantages of centralized versus decentralized implementations of DeAuth.

Table 4. Operations and their services used.

Operations	Firebase	Ethereum	Web3.Storage
Saving an authorization message	x		x
Retrieving an authorization message	x		x
Submitting an authorization message transaction	x	x	
Retrieving an authorization message transaction	x	x	
Storing various sized data	x		x
Retrieving various sized data	x		x

7.1 *Time Performance*

Figure 28 shows the average time to save and retrieve authorization messages from storage. Saving to Web3.Storage took approximately 1.4 s longer than saving to Firebase. Retrieving messages were comparable between the two.

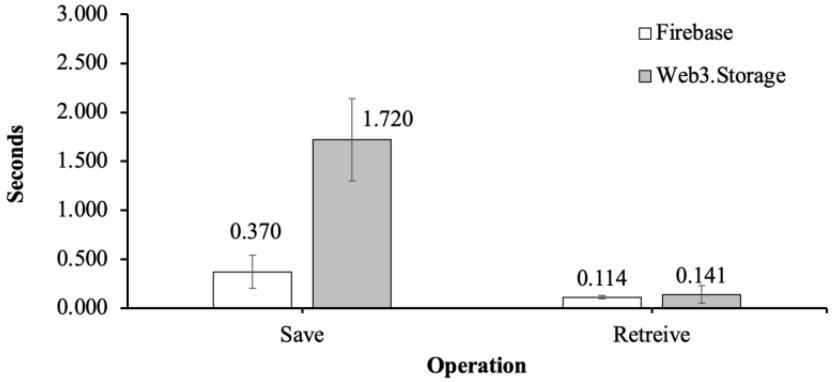


Figure 28. Average time to save and retrieve authorization messages from storage.

The results comparing the average time for submitting and retrieving an authorization message transaction are presented in Figure 29. Recall that authorization messages are first saved to storage, followed by a blockchain submission via the Authorization Contract. Using Firebase resulted in a significantly faster submission time of 0.370 s, whereas Ethereum took 15.101 s. On the other hand, retrieving a transaction from Ethereum was faster with 0.283 s, but it was still slower than Firebase, which took 0.114 s.

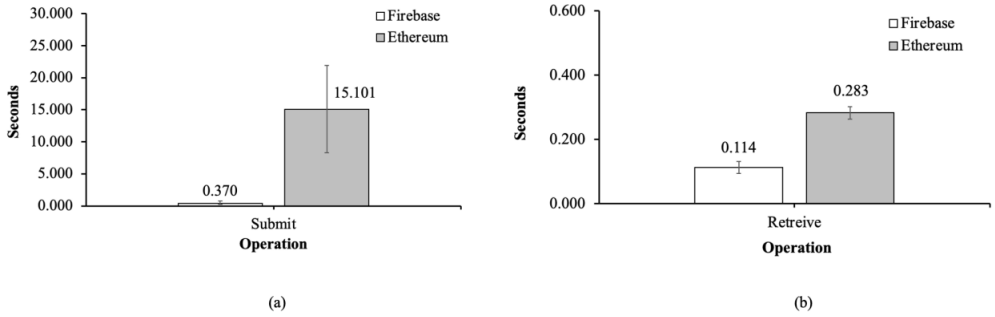


Figure 29. Measured time to (**a**) submit and (**b**) retrieve authorization message transaction.

Figure 30 shows the average times to upload different sized files to Firebase and Web3.Storage. Image files and binary files were used for this test. It can be observed that, for files of size 1 MB, 10 MB, and 1000 MB, the time taken to upload to Firebase was faster than to Web3.Storage. The difference in time increased as the file size increased. Notably, for a 1000 MB file, Firebase was on average approximately 59 s faster than Web3.Storage. However, for the 100 MB file, Firebase surprisingly took longer to upload than Web3.Storage. We speculate reasons this unexpected observation in Section 8.

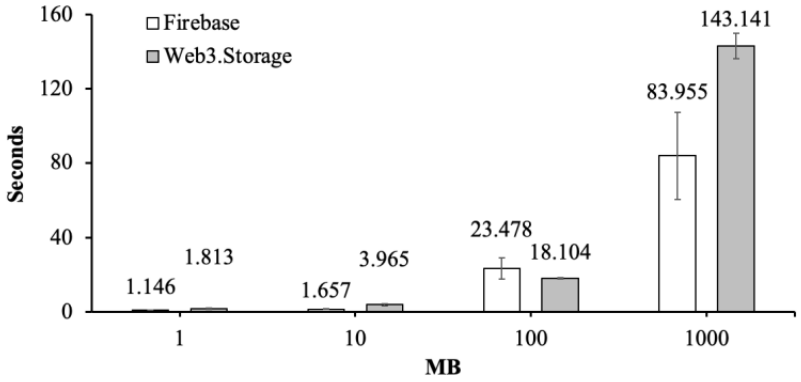


Figure 30. Upload times of various file sizes to storage.

Figure 31 shows average time to download various sized files to local storage. We measured the time to download the 1 MB, 10 MB, 100 MB and 1000 MB files to local storage. For this metric, we included our local IPFS node when downloading the 100 MB and 1000 MB files. The results indicate that the average download times for Firebase and Web3.Storage were similar for all file sizes. Using our own IPFS node however, showed a significant 90% download time improvement for both the 100 MB and 1000 MB files.

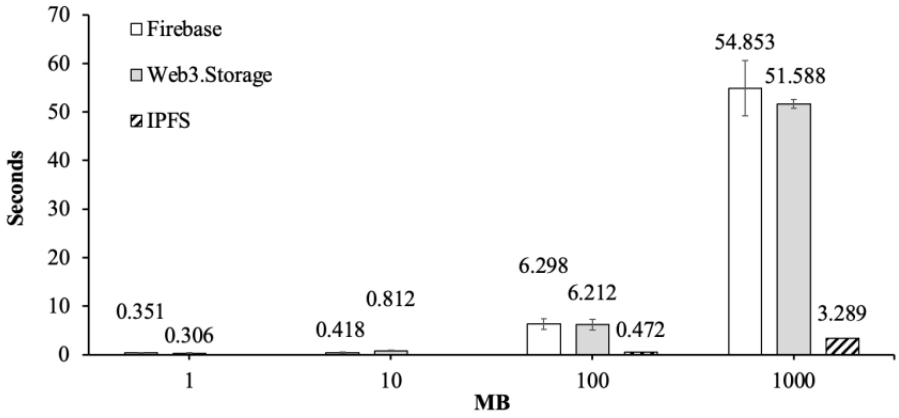


Figure 31. Download time of various files sizes from storage.

7.2 *Costs*

In this section, we report on the cost of DeAuth's main operation transactions, which fall into two categories: deploying contracts and smart contract function calls. The reported ETH amounts include miner fees. The ETH price at the time of testing was $1719.74/ETH. Table 5 presents the cost of performing DeAuth's main operations on the to the Ethereum Blockchain, Goerli Network. The cost to deploy the Authorization and Identity contract was $4.46 and $4.97 respectively. The cost to submit an authorization message and publish a DID was $0.094 and $1.13 respectively.

Table 5. Cost of operations on Ethereum.

Operation	Ether (ETH)	USD
deploy authorization contract	0.002701	$4.46
deploy identity contract	0.002786	$4.97
submit authorization message	0.000568	$0.94
publish DID	0.000632	$1.13

Table 6 summarizes the costs associated with storing one GB of data per month on these platforms, including the download bandwidth cost. The prices listed reflect the current "pay as you go" rates, as both services offer promotions that provide free initial storage. Firebase charges a rate of $0.03/GB*mo for storage and $0.12 for bandwidth, which results in a total cost of $0.15/GB*mo. It is assumed a DR downloads files to their local machine after being shared by a DM, thus the reason we total the storage and bandwidth. In contrast, Web3.Storage charges a higher rate of $0.08/GB*mo for storage but does not charge for bandwidth.

Table 6. Cost of Storage.

Operation	Firebase	Web3.Storage
storage ($/GB*mo)	$0.03	$0.08
bandwidth ($/GB*mo)	$0.12	-
total	$0.15	$0.08

8. Discussions

In this section, we analyzed the performance and cost results presented in the preceding section, offering our interpretation of significant findings. Prior to delving into the discussions, we address the limitations identified within this study that could potentially impact our interpretations.

8.1 *Research Limitations*

While this primary aim of this paper is to introduce a scheme to authenticate users and authorizing permissions in decentralized system, it is essential to recognize limitations within the study. Firstly, constraints are inherent in the services and technologies employed to construct both iterations of the prototype, notably Firebase, Ethereum, and Web3.Storage. We acknowledge that these choices can impact the performance and cost outcomes, which may exhibit variance if alternative providers were selected. Despite the comparison provided between centralized and decentralized implementations, the absence of additional testing does not detract from the study's core objectives, which focus on demonstrating DeAuth's feasibility within a decentralized framework, rather than emphasizing its performance or cost-effectiveness. The comprehensive examination and benchmarking of DeAuth against other centralized providers are reserved for future investigations.

Furthermore, while other cloud database providers were available, Google Firebase was selected for its simplicity and ease of prototyping with minimal overhead. A prior comparison conducted by authors [70] evaluated several cloud database providers, concluding that Amazon Relational Database Service (https://aws.amazon.com/rds/) offered superior performance, while Firebase was deemed suitable for personal use, such as prototypes, aligning with the intended purpose of this study. Similarly, various blockchain were considered. Although several blockchain options existed, Ethereum's development tools eased prototype development. Section 8.2 mentions potential alternatives to enhance performance and reduce costs, yet a comprehensive evaluation comparing various blockchains is deferred to future studies.

One notable limitation pertains to the assumptions underlying the practical application of DeAuth in real-world settings. Specifically, our study operates under the assumption that existing identity authorities, such as the DMV issuing driver's licenses and the Department of State issuing passports, already possess requisite technical infrastructure to facilitate

the implementation of DeAuth and its associated functionalities. This assumption further extends to entities responsible for issuing other VCs. For instance, in the scenario outlined in Section 4.3, the hospital would have DeAuth integrated into its existing systems to effectively issue VCs and share EHRs. Indeed, there is a substantial development and maintenance costs of necessary infrastructure and software. A comprehensive investigation into the precise expenses involved in the implementation of DeAuth is deferred to future research endeavors.

Last limitation is related to the usage of Web3.Storage. The results in Section 7.1 included self-hosting an IPFS node. Self-hosting an IPFS is not a prerequire in DeAuth, users may lack the desire or inability to self-host an IPFS node. Therefore, an alternative is to pay a provider to access an IPFS network and distribute the data across multiple nodes. It is important to note that using a third party for an IPFS gateway, such as Web3.Storage, does not designate it as a centralized data custodian, as it does not control who can access and retrieve data. Section 8.3 provides some cost comparisons using alternative IPFS services.

8.2 *Performance*

The research community is aware of the performance limitations of the blockchain [71, 72]. Unlike normal databases, for each transaction, miners are required to validate the transaction, mine a new block and propagate it to every node in the network. Furthermore, there is a limit on the number of transactions per second (TPS) that each block can accommodate. The Ethereum (2.0) blockchain, for example, has an average block time of approximately 12 s and can process 27–30 TPS. The impact of these limitations is reflected in Figure 26, where it takes approximately 15 s to submit an authorization message transaction on the Ethereum blockchain, whereas it occurs nearly instantaneously using Firebase.

In the context of DeAuth, optimizing performance is crucial, as users may not be willing to wait for 12–20 s for authorization message transactions to be confirmed. Moreover, a data manager, such as the hospital in our use case, may receive numerous requests to share data, requiring efficient processing. To address these concerns, we recommend the utilization of high-performance blockchains, such as Polygon and Avalanche, for the future DeAuth implementations [72].

In terms of file uploading, including images, Firebase demonstrated a faster upload time compared to Web3.Storage. Although there was no significant difference for small-sized files, uploading a 1 GB file was completed approximately one minute faster on Firebase (Figure 27). Web3.Storage hosts its own IPFS nodes and distributes a copy of the uploaded file(s) to each node. Unfortunately, implementation details of Web3.Storage were not disclosed and may have an impact on upload time. It is important to note that the upload time of Web3.Storage does not reflect the performance of the IPFS protocol itself. For future research, we plan to evaluate other IPFS services such as Filebase (https://filebase.com/) and Infura (https://www.infura.io/) and compare their upload times to those of Firebase.

We examined the performance of Firebase and Web3.Storage with regards to file downloading times. Our findings indicate that both services exhibit comparable download times. It is likely that this is because both services require the files to be downloaded through a single connection in a gateway server. To explore the potential benefits of leveraging the IPFS network, we self-hosted an IPFS node and measured the time required to retrieve (i.e., download) files. Results in Figure 31 demonstrated that downloading files using a self-hosted node was significantly faster than both Firebase and Web3.Storage. IPFS operates in a P2P fashion and can download pieces of files from multiple nodes simultaneously. Thus, it is recommended that users host their own IPFS node if they can.

8.3 *Costs*

In DeAuth, there are two main expenses that need to be considered, namely the cost of using the blockchain and the cost of storing data. In the centralized iteration, there are no costs associated with deploying contracts or submitting authorization message. However, data storage costs are more with Firebase (Table 6).

For the decentralized iteration, one DID contract is required for the entire scheme and one authorization contract is required per data manager. Additionally, each participant needs to publish a DID to make it public. At this point, costs are still relatively low. However, costs become a concern when submitting authorization messages to the blockchain, as submitting a message on Ethereum costs approximately $1 (Table 5). An entire authorization sequence consisting of

RequestData, *AskPermission*, *AllowPermission*, and *ShareData*, requires four messages, which amounts to $4. This could be a cost concern for DMs who receive many requests to share files.

One solution to address the cost issue is to divide the costs between DR and DM, with the former paying for the *RequestData* and *ShareData* messages, and the latter paying for the *AskPermission* and *AllowPermission* messages. However, the cost of submitting authorization messages remains unchanged. Another approach to address this issue is to use a blockchain with lower miner fees. We tested deploying the contract and submitting authorization messages on the Polygon [73] blockchain. Table 7 shows the cost of deploying a contract and submitting a message is approximately $0.03 and $0.002, respectively. This solution reduces authorization messaging costs significantly.

Table 7. Cost of operations on Polygon.

Operation	Polygon (Matic)	USD
deploy authorization contract	0.025	$0.03
deploy identity contract	0.0268	$0.03
submit authorization message	0.001713	$0.002
publish DID	0.00147	$0.002

Table 8 presents alternative hosting services such as Infura, Filebase, and Fleek (https://fleek.co/), which are other IPFS file hosting services. The prices offered by these hosting services are relatively similar. It is important to note that DeAuth's scheme is to enable data sharing among users, rather than provide a data storage solution. Thus, in DeAuth, IPFS is only necessary to facilitate file exchanges between users. Generally, DMs will manage and store files on their own device or network before being requested by a DR, and similarly DRs will download the files from IPFS to their own storage device or network. As a result, IPFS hosting services are not essential if users are willing to host their own IPFS node. In an ideal scenario, both DRs and DMs would host their own IPFS node, which eliminates the cost of using IPFS hosting services.

Table 8. Cost to store data on alternative hosting services.

Item	Amazon S3	Infura	Filebase	Fleek
storage ($/GB*mo)	$0.023	$0.08	$0.10	$0.12
bandwidth ($/GB*mo)	$0.09	$0.12	$0.05	$0.05
total	$0.11	$0.20	$0.15	$0.17

9. Conclusions

In this research, we introduced DeAuth, a novel decentralized authentication and authorization scheme for secure private data sharing. DeAuth employs blockchain technology, decentralized identity, and P2P distributed storage to enable users to have more control over their PII and allow uses to have permissioning power to share their private data without using centralized services. Authentication is achieved by verifying DIDs and VCs, while authorization is managed by submitting authorization messages to the blockchain via smart contracts. We demonstrated DeAuth's viability by developing a prototype, with the Ethereum blockchain and IPFS, to authenticate hospital patients and authorize sharing of private health records. Furthermore, we presented preliminary performance and cost evaluation of the protype compared to using a traditional cloud service. Our work has shown that DeAuth is a promising solution for secure private data sharing in a decentralized system.

Testing between our centralized and decentralized protype versions showed that decentralization comes at a performance cost. However, using a self-hosted IPFS node when retrieving large files from IPFS took significantly less time than downloading from Firebase, demonstrating an advantage of IPFS's P2P network. Furthermore, the cost of implementing DeAuth largely depends on the blockchain used to deploy smart contracts and submit authorization messages.

Using Ethereum incurred higher costs compared to using Firebase. Although, using an alternative blockchain such as Polygon can greatly reduce costs due to lower miner gas fees.

9.1 *Future Works*

This research represents an exciting contribution to the decentralized technology field. However, several crucial features omitted from our prototype warrant attention. These include the capability to generate multiple public and private DIDs, facilitate wallet-to-wallet messaging, and establish secure connections between wallets using private DIDs for data transfer, such as VCs. Additionally, the integration of attributes and roles into the schema could enable the implementation of fine-grained access control.

Expanding the study's scope to encompass performance optimization, cost effectiveness and improved security remains a prospect for future endeavors. This entails improving the architecture and conducting a more comprehensive analysis and evaluation, comparing various centralized cloud services and decentralized technologies. Furthermore, a meticulous cost and security analysis is warranted for the development of a production ready DeAuth application.

Additionally, further use-case studies and analysis are required to strengthen the practical application of DeAuth in real-world scenarios. Potential application domains include supply chain management, pharmaceuticals, education credentialing, legal services, and finance.

Nomenclature

Encryption Nomenclature

Term	Description
sk	secret or private key
pk	public key
CT	cipher text
m	plain-text authorization message
m^{CT}	encrypted authorization message
t	authorization message type
k	session key
k^{CT}	encrypted session key
f	plain-text file(s)
f^{CT}	encrypted file(s)

Decentralized Identity and Systems Nomenclature

Term	Description
S	subject
I	issuer
V	verifier
VC	verifiable credential
VP	verifiable presentation
DID	public decentralized identifier
DID^{CT}	Encrypted DID
DID*	private DID
DID^{*CT}	Encrypted DID*
DDO	DID document
DO	data owner
DM	data manager
DR	Data requester
TID	blockchain transaction identifier
CID	IPFS content identifier

Acknowledgments

This research was sponsored by the UNLV Top Tier Doctoral Graduate Research Assistantship (TTDGRA) Grant Program.

Conflict of Interest

There is no conflict of interest for this study.

Appendix

Smart Contract Source Code

```
pragma solidity >=0.8.2 <0.9.0;

/**
 * @title Identity
 * @dev Publish and lookup DIDs from the the blockchain.
 */
contract Identity {

   mapping(string => string) documents;
   string[] dids;

    /**
     * @dev Publish did to ledger
     * @param did decentralzied identifier
     * @param ddo CID of the did document
     */
    function publish(string calldata did, string calldata ddo) public {
        dids.push(did);
        documents[did] = ddo;
    }

    /**
     * @dev Lookup did document from the ledger
     * @param did decentralzied identifier
     */
    function lookup(string calldata did) public view returns (string memory ddo)
{         return documents[did];
    }

    /**
     * @dev Return all dids in the ledger. This function was not used in the
     * prototype and only used for benchmark testing.
     */
    function getAllDIDs() public view returns (string[] memory DIDs){
        return dids;
    }
}
```

Figure A1. Identity smart contract in the DeAuth prototype.

```
pragma solidity >=0.8.2 <0.9.0;

/**
 * @title Authorization
 * @dev Submit & retrieve authorization message transactions from the blockchain.
 */
contract Authorization {

    struct Message {
        string messageType;
        string cid;
    }

    Message[] messages;

    /**
     * @dev Store value in variable
     * @param messageType type of authorization message
     * @param cid IPFS content identifier
     */
    function submit(string calldata messageType, string calldata cid) public {
        messages.push(Message(messageType, cid));
    }

    /**
     * @dev Return all messages. This function was not used in the prototype
     * since the message transactions were retreived used the transaction id.
     */
    function all() public view returns (Message[] memory message){
        return messages;
    }
}
```

Figure A2. Authorization smart contract in the DeAuth prototype.

References

[1] M. R. DeLisi, "Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach Nearly $600 Billion in 2023," Gartner, Accessed: Jan. 14, 2023. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2022-10-31-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-600-billion-in-2023#:~:text=Worldwide%20end-user%20spending%20on,18.8%25%20growth%20forecast%20for%202022.

[2] A. Vigderman and G. Turner, "The Data Big Tech Companies Have On You," security.org, Accessed: Jan. 14, 2023. [Online]. Available: https://www.security.org/resources/data-tech-companies-have/.

[3] J. Isaak and M. J. Hanna, "User data privacy: Facebook, Cambridge Analytica, and privacy protection," *Computer*, vol. 51, pp. 56–59, 2018, doi: 10.1109/MC.2018.3191268.

[4] K. O'Flaherty, "Google+ Security Bug—What Happened, Who Was Impacted And How To Delete Your Account," Forbes, Accessed: Jan. 14, 2023. [Online]. Available: https://www.forbes.com/sites/kateoflahertyuk/2018/10/09/google-plus-breach-what-happened-who-was-impacted-and-how-to-delete-your-account/?sh=459b34526491.

[5] Y. Zou, A. H. Mhaidli, A. McCall, and F. Schaub, ""I've Got Nothing to Lose" : Consumers' Risk Perceptions and Protective Actions after the Equifax Data Breach," in *Proc. 14th Symp. Usable Priv. Secur. (SOUPS 2018)*, Baltimore, MD, USA, Aug. 12–14, 2018, pp. 197–216.

[6] J. Rasalam and R. J. Elson, "Cybersecurity And Management's Ethical Responsibilities: The Case Of Equifax And Uber," *Glob. J. Bus. Pedagog.*, vol. 3, pp. 8–15, 2019.

[7] N. Tihanyi, A. Kovács, G. Vargha, and Á. Lénárt, "Unrevealed patterns in password databases Part one: analyses of cleartext passwords," in *Technology and Practice of Passwords*. Cham, Switzerland: Springer, 2014, doi: 10.1007/978-3-319-24192-0_6.

[8] J. Linn, "Trust models and management in public-key infrastructures," *RSA Lab.*, pp. 1–13, 2000.

[9] B. Rajendran, "Evolution of PKI ecosystem," in *Proc. Int. Conf. Public Key Infrastruct. Appl. (PKIA)*, Bangalore, India, Nov. 14–15, 2017, doi: 10.1109/PKIA.2017.8278951.

[10] O. Dib and K. Toumi, "Decentralized identity systems: Architecture, challenges, solutions and future directions," *Ann. Emerg. Technol. Comput.*, vol. 4, no. 5, pp. 19–40, 2020, doi: 10.33166/AETiC.2020.05.002.

[11] K. C. Toth and A. Anderson-Priddy, "Self-Sovereign Digital Identity: A Paradigm Shift for Identity," *IEEE Secur. Priv.*, vol. 17, pp. 17–27, 2019, doi: 10.1109/MSEC.2018.2888782.

[12] Y. Wang, Y. Ma, K. Xiang, Z. Liu, and M. Li, "A role-based access control system using attribute-based encryption," in *Proc. 2018 Int. Conf. Big Data Artif. Intell. (BDAI)*, Beijing, China, Jun. 22–24, 2018, doi: 10.1109/BDAI.2018.8547200.

[13] J. Adler-Milstein, et al., "Electronic health record adoption in US hospitals: the emergence of a digital " advanced use" divide," *J. Am. Med. Inform. Assoc.*, vol. 24, no. 6, pp. 1142–1148, 2017, doi: 10.1093/jamia/ocx080.

[14] S. Angraal, H. M. Krumholz, and W. L. Schulz, "Blockchain technology: applications in health care," *Circ. Cardiovasc. Qual. Outcomes*, vol. 10, no. 9, p. e003800, 2017, doi: 10.1161/CIRCOUTCOMES.117.003800.

[15] I. Yaqoob, K. Salah, R. Jayaraman, and Y. Al-Hammadi, "Blockchain for healthcare data management: opportunities, challenges, and future recommendations," *Neural Comput. Appl.*, vol. 34, no. 11, pp. 11475–11490, 2021, doi: 10.1007/s00521-020-05519-w.

[16] R. Kuhn, D. Yaga, and J. Voas, "Rethinking distributed ledger technology," *Computer*, vol. 52, no. 2, pp. 68–72, 2019, doi: 10.1109/MC.2019.2898162.

[17] H. L. H. S. Warnars and E. Abdurachman, "Blockchain technology open problems and impact to supply chain management in automotive component industry," in *Proc. 2020 6th Int. Conf. Comput. Eng. Des. (ICCED)*, Sukabumi, Indonesia, Oct. 15–16, 2020, doi: 10.1109/ICCED51276.2020.9415836.

[18] J. Kang, "Convergence analysis of BIM & blockchain technology in construction industry informatization," in *Proc. 2022 4th Int. Conf. Smart Syst. Inventive Technol. (ICSSIT)*, Tirunelveli, India, Jan. 20–22, 2022, doi: 10.1109/ICSSIT53264.2022.9716352.

[19] E. Fernando, "The Business Process of Good Manufacturing Practice Based on Blockchain Technology in the Pharmaceutical Industry," in *Proc. 2021 Fifth Int. Conf. Inf. Retr. Knowl. Manag. (CAMP)*, Kuala Lumpur, Malaysia, Jun. 15–16, 2021, doi: 10.1109/CAMP51653.2021.9498104.

[20] "A Next-Generation Smart Contract and Decentralized Application Platform," Ethereum.org, Accessed: Nov. 12, 2020. [Online]. Available: https://ethereum.org/en/whitepaper/.

[21] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Advances in Cryptology. CRYPTO 1984. Lecture Notes in Computer Science*. Berlin, Germany: Springer, doi: 10.1007/3-540-39568-7_5.

[22] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology—EUROCRYPT 2005. EUROCRYPT 2005. Lecture Notes in Computer Science*. Berlin, Germany: Springer, doi: 10.1007/11426639_27.

[23] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, Alexandria, VA, USA, Oct. 30–Nov. 3, 2006, doi: 10.1145/1180405.1180418.

[24] R. Wang, X. Wang, W. Yang, S. Yuan, and Z. Guan, "Achieving fine-grained and flexible access control on blockchain-based data sharing for the Internet of Things," *China Commun.*, vol. 19, pp. 22–34, 2022, doi: 10.23919/JCC.2022.06.003.

[25] S. Wang, Y. Zhang, and Y. Zhang, "A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems," *IEEE Access*, vol. 6, pp. 38437–38450, 2018, doi: 10.1109/ACCESS.2018.2851611.

[26] F. Ghaffari, E. Bertin, N. Crespi, S. Behrad, and J. Hatin, "A Novel Access Control Method Via Smart Contracts for Internet-Based Service Provisioning," *IEEE Access*, vol. 9, pp. 81253–81273, 2021, doi: 10.1109/ACCESS.2021.3085831.

[27] J. P. Cruz, Y. Kaji, and N. Yanai, "RBAC-SC: Role-Based Access Control Using Smart Contract," *IEEE Access*, vol. 6, pp. 12240–12251, 2018, doi: 10.1109/ACCESS.2018.2812844.

[28] M. Tanveer, A. U. Khan, M. Ahmad, T. N. Nguyen, and A. A. A. El-Latif, "Resource-Efficient Authenticated Data Sharing Mechanism for Smart Wearable Systems," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, pp. 2525–2536, 2022, doi: 10.1109/TNSE.2022.3203927.

[29] L. Soltanisehat, R. Alizadeh, H. Hao, and K.-K. R. Choo, "Technical, Temporal, and Spatial Research Challenges and Opportunities in Blockchain-Based Healthcare: A Systematic Literature Review," *IEEE Trans. Eng. Manag.*, vol. 70, pp. 353–368, 2020, doi: 10.1109/TEM.2020.3013507.

[30] A. Hasselgren, K. Kralevska, D. Gligoroski, S. A. Pedersen, and A. Faxvaag, "Blockchain in healthcare and health sciences—A scoping review," *Int. J. Med. Inform.*, vol. 134, p. 104040, 2019, doi: 10.1016/j.ijmedinf.2019.104040.

[31] B. Houtan, A. S. Hafid, and D. Makrakis, "A Survey on Blockchain-Based Self-Sovereign Patient Identity in Healthcare," *IEEE Access*, vol. 8, pp. 90478–90494, 2020, doi: 10.1109/ACCESS.2020.2994090.

[32] R. Soltani, U. T. Nguyen, and A. An, "A new approach to client onboarding using self-sovereign identity and distributed ledger," in *Proc. 2018 IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Soc. Comput. (CPSCom) IEEE Smart Data (SmartData)*, Halifax, NS, Canada, Jul. 30–Aug. 3, 2018, doi: 10.1109/Cybermatics_2018.2018.00205.

[33] O. Avellaneda et al., "Decentralized identity: Where did it come from and where is it going?" *IEEE Commun. Stand. Mag.*, vol. 3, pp. 10–13, 2019, doi: 10.1109/MCOMSTD.2019.9031542.

[34] M. Korir, S. Parkin, and P. Dunphy, "An empirical study of a decentralized identity wallet: Usability, security, and perspectives on user control," in *Proc. Eighteenth Symp. Usable Priv. Secur. (SOUPS 2022)*, Boston, MA, USA, Aug. 7–9, 2022, pp. 195–211.

[35] R. Belchior, B. Putz, G. Pernul, M. Correia, A. Vasconcelos, and S. Guerreiro, "SSIBAC: self-sovereign identity based access control," in *Proc. 2020 IEEE 19th Int. Conf. Trust, Secur. Priv. Comput. Commun. (TrustCom)*, Guangzhou, China, Dec. 29, 2020–Jan. 1, 2021, doi: 10.1109/TrustCom50675.2020.00264.

[36] N. V. Kulabukhova, "Zero-knowledge proof in self-sovereign identity," in *Proc. 27th Int. Symp. Nucl. Electron. Comput. (NEC'2019)*, Budva, Montenegro, Sep. 30–Oct. 4, pp. 381–385.

[37] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," Accessed: Sep. 20, 2020. [Online]. Available: http://dx.doi.org/10.2139/ssrn.3440802.

[38] B. K. Mohanta, S. S. Panda, and D. Jena, "An Overview of Smart Contract and Use Cases in Blockchain Technology," in *Proc. 2018 9th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, Bengaluru, India, Jul. 10–12, 2018, doi: 10.1109/ICCCNT.2018.8494045.

[39] C. S. Ke and Y. R. Chen, "Instruction Verification of Ethereum Virtual Machine by Formal Method," in *Proc. 2020 Indo–Taiwan 2nd Int. Conf. Comput. Anal. Netw. (Indo-Taiwan ICAN)*, Rajpura, India, Feb. 7–15, 2020, doi: 10.1109/Indo-TaiwanICAN48429.2020.9181334.

[40] Y. Cao and L. Yang, "A survey of identity management technology," in *Proc. 2010 IEEE Int. Conf. Inf. Theory Inf. Secur.*, Beijing, China, Dec. 17–19, 2010, doi: 10.1109/ICITIS.2010.5689468.

[41] E. Maler and D. Reed, "The Venn of Identity: Options and Issues in Federated Identity Management," *IEEE Secur. Priv.*, vol. 6, pp. 16–23, 2008, doi: 10.1109/MSP.2008.50.
[42] T. Huang and F. Guo, "Research on Single Sign-on Technology for Educational Administration Information Service Platform," in *Proc. 2021 3rd Int. Conf. Comput. Commun. Internet (ICCCI)*, Nagoya, Japan, Jun. 25–27, 2021, doi: 10.1109/ICCCI51764.2021.9486813.
[43] J. Jensen, "Federated identity management challenges," in *Proc. 2012 Seventh Int. Conf. Avail. Rel. Secur.*, Prague, Czech Republic, Aug. 20–24, 2012, doi: 10.1109/ARES.2012.68.
[44] S. M. Matyas, "Digital signatures—An overview," *Comput. Netw.*, vol. 3, pp. 87–94, 1976, doi: 10.1016/0376-5075(79)90007-2.
[45] B. A. Alzahrani, "An Information-Centric Networking Based Registry for Decentralized Identifiers and Verifiable Credentials," *IEEE Access*, vol. 8, pp. 137198–137208, 2020, doi: 10.1109/ACCESS.2020.3011656.
[46] A. Tobin, "*Sovrin: What goes on the ledger*?" Salt Lake City, UT, USA: Evernym/Sovrin Foundation, 2018, pp. 1–11.
[47] "Verifiable Credentials Data Model," W3C, Accessed: Jan. 15, 2023. [Online]. Available: https://www.w3.org/TR/vc-data-model/#presentations-0.
[48] M. P. Bhattacharya, P. Zavarsky, and S. Butakov, "Enhancing the security and privacy of self-sovereign identities on hyperledger indy blockchain," in *Proc. 2020 Int. Symp. Netw. Comput. Commun. (ISNCC)*, Montreal, QC, Canada, Oct. 20–22, 2020, doi: 10.1109/ISNCC49221.2020.9297357.
[49] H. Saidi, N. Labraoui, A. A. A. Ari, L. A. Maglaras, and J. H. M. Emati, "DSMAC: Privacy-Aware Decentralized Self-Management of Data Access Control Based on Blockchain for Health Data," *IEEE Access*, vol. 10, pp. 101011–101028, 2021, doi: 10.1109/ACCESS.2022.3207803.
[50] Y. Jing, J. Li, Y. Wang, and H. Li, "The introduction of digital identity evolution and the industry of decentralized identity," in *Proc. 2021 3rd Int. Acad. Exch. Conf. Sci. Technol. Innov. (IAECST)*, Guangzhou, China, Dec. 10–12, 2021, doi: 10.1109/IAECST54258.2021.9695553.
[51] J. Benet, "IPFS-content addressed, versioned, P2P file system," *arXiv preprint*, 2014, doi: 10.48550/arXiv.1407.3561.
[52] D. Khovratovich and J. Law, "BIP32-Ed25519: Hierarchical Deterministic Keys over a Non-Linear Keyspace," in *Proc. 2017 IEEE Eur. Symp. Secur. Priv. Workshops (EuroS&PW)*, Paris, France, Apr. 26–28, 2017, doi: 10.1109/EuroSPW.2017.47.
[53] P. Wuille, "Hierarchical Deterministic Wallets," GitHub, Accessed: Jan. 15, 2023. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki.
[54] M. Palatinus and P. Rusnak, "Multi-Account Hierarchy for Deterministic Wallets," GitHub, Accessed: Jan. 15, 2023. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki.
[55] A. Antonopoulos, "*Mastering Bitcoin*", 1st ed. Sebastopol, CA, USA: O'Reilly Media Inc., 2014.
[56] A. A. Frozza, R. dos Santos Mello, and F. D. S. da Costa, "An approach for schema extraction of JSON and extended JSON document collections," in *Proc. 2018 IEEE Int. Conf. Inf. Reuse Integr. (IRI)*, Salt Lake City, UT, USA, Jul. 6–9, 2018, doi: 10.1109/IRI.2018.00060.
[57] M. Abdalla and D. Pointcheval, "Simple password-based encrypted key exchange protocols," in *Topics in Cryptology—CT-RSA 2005. Lecture Notes in Computer Science*, Berlin, Germany: Springer, pp. 191–208, doi: 10.1007/978-3-540-30574-3_14.
[58] S. Sharma and K. Shah, "Exploring Security Threats on Blockchain Technology along with possible Remedies," in *Proc. 2022 IEEE 7th Int. Conf. Convergence Technol. (I2CT)*, Mumbai, India, Apr. 7–9, 2022, doi: 10.1109/I2CT54291.2022.9825123.
[59] A. AlFaw, W. Elmedany, and M. S. Sharif, "Blockchain vulnerabilities and recent security challenges: A review paper," in *Proc. 2022 Int. Conf. Data Anal. Bus. Ind. (ICDABI)*, Sakhir, Bahrain, Oct. 25–26, 2022, doi: 10.1109/ICDABI56818.2022.10041611.
[60] X. Yang, Y. Chen, and X. Chen, "Effective Scheme against 51% Attack on Proof-of-Work Blockchain with History Weighted Information," in *Proc. 2019 IEEE Int. Conf. Blockchain (Blockchain)*, Atlanta, GA, USA, Jul. 14–17, 2019, pp. 261–265, doi: 10.1109/Blockchain.2019.00041.
[61] L. Pennella, "Proof-of-Stake (PoS)," Ethereum.org, Accessed: Mar. 26, 2023. [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/.
[62] M. Aydar, S. C. Cetin, S. Ayvaz, and B. Aygun, "Private Key Encryption and Recovery in Blockchain," *arXiv preprint*, 2019, doi: 10.48550/arXiv.1907.04156.

[63] H. P. Singh, K. Stefanidis, and F. Kirstein, "A Private Key Recovery Scheme Using Partial Knowledge," in *Proc. 2021 11th IFIP Int. Conf. New Technol. Mobility Secur. (NTMS)*, Paris, France, Apr. 19–21, 2021, doi: 10.1109/NTMS49979.2021.9432642.

[64] "Privacy and Encryption," IPFS Docs, Accessed: Mar. 26, 2023. [Online]. Available: https://docs.ipfs.tech/concepts/privacy-and-encryption/.

[65] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini, and Y. Khamayseh, "Comprehensive Study of Symmetric Key and Asymmetric Key Encryption Algorithms," in *Proc. 2017 Int. Conf. Eng. Technol. (ICET)*, Antalya, Turkey, Aug. 21–23, 2017, doi: 10.1109/ICEngTechnol.2017.8308215.

[66] S. S. Ghosh, H. Parmar, P. Shah, and K. Samdani, "A Comprehensive Analysis Between Popular Symmetric Encryption Algorithms," in *Proc. 2018 IEEE Punecon*, Pune, India, Nov. 30–Dec. 2, 2018, doi: 10.1109/PUNECON.2018.8745324.

[67] Z. Hercigonja, "Comparative Analysis of Cryptographic Algorithms," *Int. J. Digit. Technol. Econ.*, vol. 1, pp. 127–134, 2016.

[68] P. Austria, "Analysis of Blockchain-Based Storage Systems," Master Thesis, Univ. Nevada, Las Vegas, Paradise, NV, USA, Jan. 2020, doi: 10.34917/22085740.

[69] J. Benet and N. Greco, "Filecoin: A Decentralized Storage Network," Res. Protocol AI, Accessed: Jun. 20, 2020. [Online]. Available: https://research-protocol-ai.ipns.dweb.link/publications/filecoin-a-decentralized-storage-network/.

[70] T. H. Shareef, K. H. Sharif, and B. N. Rashid, "A Survey of Comparison Different Cloud Database Performance: SQL and NoSQL," *Passer J. Basic Appl. Sci.*, vol. 4, pp. 45–57, 2022, doi: 10.24271/PSR.2022.301247.1104.

[71] C. Fan, S. Ghaemi, H. Khazaei, and P. Musilek, "Performance Evaluation of Blockchain Systems: A Systematic Survey," *IEEE Access*, vol. 8, pp. 126927–126950, 2020, doi: 10.1109/ACCESS.2020.3006078.

[72] A. W. d. S. Abreu, E. F. Coutinho, and C. I. M. Bezerra, "Performance Evaluation of Data Transactions in Blockchain," *IEEE Lat. Am. Trans.*, vol. 20, pp. 409–416, 2021, doi: 10.1109/TLA.2022.9667139.

[73] "Ethereum's Internet of Blockchains," LeewayHertz, Accessed: Feb. 15, 2023. [Online]. Available: https://www.leewayhertz.com/polygon-ethereums-internet-of-blockchains/.