May 2023

Dea2uth: A Decentralized Authentication and Authorization Scheme for Secure Private Data Transfer

Phillipe Austria

DEA^{2}UTH: A DECENTRALIZED AUTHENTICATION AND AUTHORIZATION SCHEME FOR SECURE PRIVATE DATA TRANSFER

By

Phillipe S. Austria

Bachelor of Science, Chemical Engineering
University of Washington
2010

Masters of Science, Computer Science
University of Nevada, Las Vegas
2020

A dissertation submitted in partial fulfillment
of the requirements for the

Doctor of Philosophy – Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2023

UNLV | GRADUATE COLLEGE

Dissertation Approval

The Graduate College
The University of Nevada, Las Vegas

April 6, 2023

This dissertation prepared by

Phillipe S. Austria

entitled

Dea^{2}uth: A Decentralized Authentication and Authorization Scheme for Secure Private Data Transfer

is approved in partial fulfillment of the requirements for the degree of

Doctor of Philosophy – Computer Science
Department of Computer Science

Yoohwan Kim, Ph.D.
Examination Committee Chair

Juyeon Jo, Ph.D.
Examination Committee Member

Wolfgang Bein, Ph.D.
Examination Committee Member

Laxmi Gewali, Ph.D.
Examination Committee Member

Tina Vo, Ph.D.
Graduate College Faculty Representative

Alyssa Crittenden, Ph.D.
Vice Provost for Graduate Education & Dean of the Graduate College

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 dissertation, we introduce DeA^{2}uth, a novel decentralized, authentication and authorization scheme for secure private data transfer. DeA^{2}uth 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 third party 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 DeA^{2}uth through use-case studies and metrics such as security, performance, and cost. Our findings indicate DeA^{2}uth to be viable alternative to using centralized services; however, the underlying technologies are still in its infancies and requires more testing before it can supplant traditional services. Overall, this dissertation provides a comprehensive examination of current decentralized technologies and contributes to a possible future where users have complete control over their data.

Acknowledgements

I would like to express my heartfelt gratitude to my esteemed academic advisor, Professor Yoohwan Kim, of the Computer Science department at the University of Nevada, Las Vegas. Dr. Kim has been an unwavering source of guidance and support throughout my Master's and Ph.D. journey. While allowing me to produce original work, he also provided invaluable direction in selecting a stimulating topic and formulating critical research inquiries.

I would also like to extend my appreciation to my committee members: Professor Ju-Yeon Jo, Professor Wolfgang Bein, Professor Laxmi Gewali, and Assistant Professor Tina Vo. I am grateful for the time they invested in assessing and evaluating my dissertation. I would like to offer a special note of thanks to Dr. Vo for the valuable advice and wisdom she has imparted during my graduate assistantship at the Office of Research and Sponsored Projects.

Lastly, but of utmost importance, I would like to acknowledge the unwavering support and love of my family, especially my mother. Their encouragement and motivation kept me focused, and this accomplishment would not have been possible without them.

Phillipe S. Austria

University of Nevada, Las Vegas

May 2023

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 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 such as the one previous mentioned, often use this data to improve their services, target advertising, and develop new products, users must that the storage provider will protect their data or not do anything malicious with it. Centralization of data, more importantly personal identifiable information (PII), attracts hackers to 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. Often times companies 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 risky [10]. For example, many services allow you sign in with a Google Gmail account, however there little to no banks that will allow SSO. 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 modals data owners enforce a policy that defines attributes or roles a data requester must have in order 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 of 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, 14, 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, 18, 19].

Immutability, traceability, transparency, and privacy are a few of the potential benefits blockchain brings and have been proposed in an increasing number of papers to improve identity management

and access control. In a traditional identity management system, a central authority, such as a government agency or a bank, is responsible for storing and verifying identity information. This can be problematic, as it requires individuals to trust the central authority and makes the system vulnerable to hacks and data breaches. Blockchain-based identity management systems aim to address these issues by storing identity information on a decentralized ledger, which can be accessed and verified by authorized parties without the need for a central authority. 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 that has created a scheme to combine decentralized identity, access control and storage to create an ecosystem that is completely private, secure, and ultimately give the user more control of their PII.

1.1 Dissertation Objectives

This dissertation introduces DeA^{2}uth, a novel schema for decentralized authentication and authorization that facilitates the secure transfer of private data. DeA^{2}uth combines decentralized identity, blockchain technology, and distributed peer-to-peer (P2P) storage to give users more control of their private data, and permissioning power to share without third party services. For this scheme, identity is proven using decentralized identifiers and verifiable credentials, while authorization to share data is performed using the blockchain. DeA^{2}uth leverages the benefits of the blockchain, including enhanced security, decentralization, transparency, and provenance. Through DeA^{2}uth, 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. The objective of this dissertation include:

1. Compare DeA^{2}uth to current methods of identity management and private data transfer.
2. Propose and define the DeA^{2}uth's scheme's architecture, components and interactions between the multiple technologies used.
3. Develop a working prototype using Ethereum Blockchain and P2P data storage.

4. Apply the prototype to three real-world use cases: (1) Transcript Verification, (2) Credit Report Transfer and (3) Healthcare Records Transfer.

5. Evaluate DeA^{2}uth with and without blockchain using metrics such as security, performance, and cost.

The remainder of this dissertation is structured as follows: Chapter 2 presents related works in the fields of decentralized identity and blockchain-based storage access schemes. Chapter 3 provides essential background knowledge to facilitate understanding of DeA^{2}uth. In Chapter 4 , we detail DeA^{2}uth's components, architecture, and sub-schemes. Chapter 5 presents the implementation and development details for creating a prototype. In Chapter 6, we report on the performance and cost evaluation based on the prototype testing. Chapter 7 discusses the results of the evaluation and presents our interpretation of the findings. Lastly, Chapter 8 provides a conclusion and outlines possible future work for this dissertation.

Chapter 2

Literature Review

The primary objective of this chapter is to provide an in-depth review of previous research on decentralized access control. In this regard, we present a comprehensive analysis of the literature, which we have categorized into three distinct sections. Firstly, we examine traditional access control, followed by a review of blockchain-based access control (BBAC). Lastly, we focus on decentralized and self-sovereign identity (SSI)-based access control. By dividing the research into these three sections, we aim to offer a structured and cohesive analysis of the existing literature in this field. Through our systematic review, we aim to identify the main trends, challenges, and future directions of decentralized access control research.

2.1 Blockchain-Based Access Control

BBAC refers to the use of distributed ledger technology 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 service leverage access control models such as Role Based Access Control (RBAC) and Attribute Based Access Control (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 ID Based Encryption (IBE) [20] and Attribute Based Encryption (ABE) [21, 22] which require a trusted third party to create and manage private keys. This inherently creates a need reliance on a centralized authority. Additionally, both RBAC and ABAC rely on polices that determine which roles or attributers grant access. This policy generating services or nodes such as Policy Decision Point and Policy Administration Point; if the nodes fail then data which users are trying to access become unavailable [23].

As a solution, recent works have been proposed [24, 25, 26]. A proposed schema, described in [24], 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 [26] 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 the features of blockchain technology to ensure secure and transparent access control within a decentralized setting. Furthermore, in [25], 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. One key area where blockchain is being utilized in healthcare is in the management of electronic health records (EHRs) and Electronic Medical Records (EMRs). 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 [27, 28, 29]. 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 [27], 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 majority of the papers reviewed in this survey focused on new schemes for improving EHR and EMR systems.

Interoperability between integrating blockchain, a new technology, to a legacy health records and data management system was also identified as an area of interest in both [27] and another survey conducted by [29]. In the latter study, the potential of blockchain technology for patent data and identity management was investigated, with the authors concluding that EHRs and Patient Health Records (PHRs) 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

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, 30, 31, 32]. This differs from traditional centralized identity management systems, in 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 is achieved using decentralized technology such as blockchain, which allows for secure and tamper-proof 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 their privacy and security and can also enable them 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 also 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 [33]. 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 enhance privacy, Zero Knowledge Proofs (ZKP) have been implemented [34]. 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.

Chapter 3

Background Knowledge

The aim of this chapter is to expound upon several fundamental concepts that are essential for comprehending DeA^{2}uth. These concepts include symmetric and asymmetric encryption, digital signatures, blockchain, decentralized identity, and the InterPlanetary File System (IPFS). Through a detailed examination of each of these topics, readers will gain a deeper understanding of the mechanisms that underlie DeA^{2}uth. The subsequent sections will provide a comprehensive exploration of each concept, discussing their principles and functionality.

3.1 Symmetric Encryption

In Symmetric Encryption, the same key is used to encrypt and decrypt messages.

$$CT = Enc(k, m) \tag{3.1}$$

$$m = Dec(k, CT) \tag{3.2}$$

where:

$$CT = \text{ciphertext}$$
$$m = \text{plaintext message}$$
$$k = \text{symmetric key}$$
$$Enc = \text{Encryption Algorithm}$$
$$Dec = \text{Decryption Algorithm}$$

In Figure 3.1, Alice wants to send an encrypted message to Bob. Alice first encrypts the plaintext message m with a symmetric key k which results in a ciphertext CT. Alice sends CT and shares k with Bob. Bob then uses k to decrypt CT and reveal m.

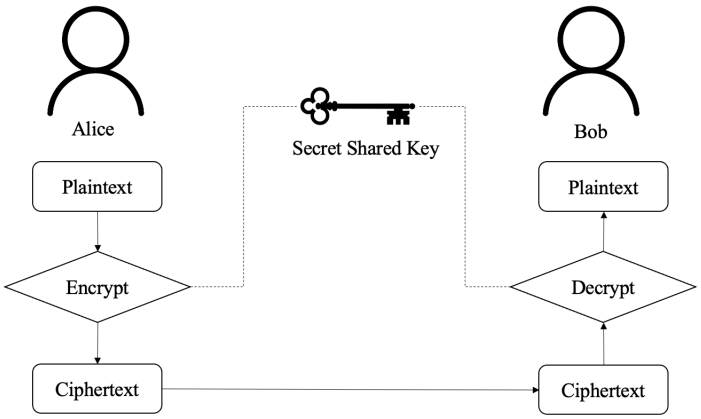


Figure 3.1: Encrypting a message with symmetric encryption.

The confidentiality of CT in a symmetric encryption scheme relies upon the secrecy of the shared secret key, denoted by k. Consequently, this key must remain confidential and not be disclosed to unauthorized parties. However, a challenge arises in distributing the key to all participating parties in the communication. If a malicious adversary were to obtain the key, they would gain access to the encrypted data. A possible solution to this limitation is to use asymmetric encryption techniques to encrypt the shared secret key prior to its distribution.

Within the realm of symmetric encryption algorithms, there exist two distinct categories: block and stream algorithms. The former operates on fixed-length blocks of plaintext, while the latter processes messages character by character in what is commonly referred to as a stream [35]. The Advanced Encryption Standard (AES) currently represents the National Institute of Technology Standard (NIST) and is considered the most secure option among symmetric encryption algorithms [36]. In terms of computational power and efficiency, stream algorithms have demonstrated superior performance compared to block algorithms [35].

3.2 Asymmetric Encryption

Asymmetric encryption, commonly referred to as Public Key Cryptography, relies upon the utilization of a pair of keys, namely a public key (pk) and a private key (sk), also known as a secret key. Both of these keys have the capability to encrypt a plaintext message to produce a ciphertext.

However, only the opposite key of the pair can decrypt the resulting ciphertext, as expressed in Equations 3.3 and 3.4. Essentially, if a public key encrypts a message, solely the corresponding private key can decrypt the message, and vice versa. Public Key Cryptography was first proposed in 1976 by W. Diffie and E. Hellman at Stanford University [37]. Beyond encryption, Public Key Cryptography serves multiple purposes, including shared key establishment, non-repudiation, identification, and message integrity.

$$CT = Enc(pk, m) \tag{3.3}$$

$$m = Dec(sk, CT) \tag{3.4}$$

where:

$$pk = \text{public key}$$

$$sk = \text{secret key}$$

Figure 3.2 illustrates how Alice can send a private message to Bob using asymmetric encryption. First, Alice encrypts a plain text message m with Bob's public key and results in a ciphertext CT. Alice then sends CT to Bob. Lastly, Bob decrypts CT with sk to obtain m.

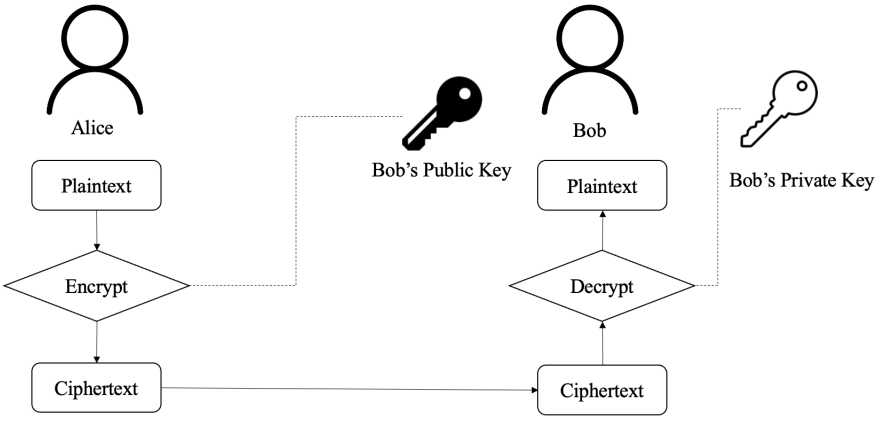


Figure 3.2: Encrypting a message with asymmetric encryption.

The fundamental principle underlying asymmetric encryption algorithms is the utilization of one-way functions, which rely upon the complexity of mathematical problems. One-way functions possess the property that given an input, they are easy to compute, but the reverse process is

significantly more challenging. This mathematical property serves as the basis for generating the public-private key pair utilized in asymmetric encryption schemes. Two of the most popular one-way problems upon which asymmetric encryption schemes rely are the integer factorization problem [38] and the discrete logarithm problem [39]. The Rivest-Shamir-Adleman (RSA) scheme [40] is based on the former, while the Diffie-Hellman Key Exchange (DHKE) scheme [37] is based on the latter. Both of these schemes are widely used and have become industry standards.

When considering the relative merits of symmetric and asymmetric encryption schemes, it is widely acknowledged in the literature that asymmetric schemes offer greater security [35, 41], albeit at the cost of requiring more time, processing power, and memory. As a result, in practice, symmetric keys are generally used for encrypting data, while asymmetric encryption is employed to encrypt the symmetric key to facilitate secure delivery to each participant.

3.3 Digital Signatures

A digital signature is a mathematical scheme to verify the authenticity and integrity of a message [42]. Using asymmetric encryption, the scheme gives the receiver high confidence that it was sent by a known sender and was not altered. Digital signatures can also provide non-repudiation, that is a signer cannot deny they have signed a message while claiming their private key which signed the message, is still secret.

Digital Signatures was first introduced in 1976 by W. Diffie and M. Hellman in the same paper which introduced Public Key Cryptography [37]. Soon after, RSA was invented and could also produce digital signatures, although only as a proof of concept and cryptographically insecure at the time [39]. Common digital signature schemes used today include: Digital Signature Algorithms (DSA), Elliptic Curve Digital Signature Algorithm, Edwards-curve Digital Signature Algorithm (EdDSA), RSA, ElGamal Signature Scheme and Schnorr Signature scheme [43].

Figure 3.3 shows in illustration of Alice wanting to prove that she was the sender of a private message and that it was not tampered with using a digital signature. The sequence is described below:

1. Alice hashes a message to produce a hash also known as a digest.

2. Alice encrypts the digest with her private key and produces a verifiable ciphertext, the digital signature.

3. Alice includes the signature with the message and encrypts then combined message with Bob's public key. The message is then sent to Bob.

4. Bob decrypts the message with his private key and separates the message from the signature

5. Bob decrypts signature with Alice's public key, verifying the signature.

6. Bob hashes the original plain text message and produces a digest.

7. Bob compares the digest from verifying the signature and the digest from hashing the message. If they match, then Bob knows that the message was sent by Alice and not tampered with.

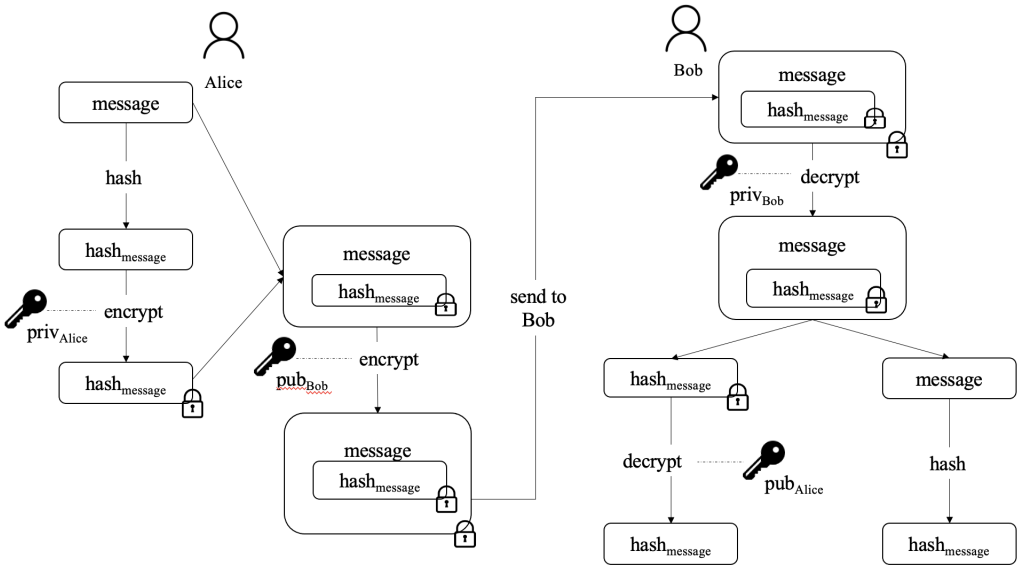


Figure 3.3: Authenticating a message with a digital signature.

3.4 Blockchain

The Blockchain is a decentralized, immutable, and cryptographically secure digital ledger that was first introduced in a white paper by [44]. 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. This section will focus on the workings of the Blockchain specifically with respect to Bitcoin, as it was the first application of this technology.

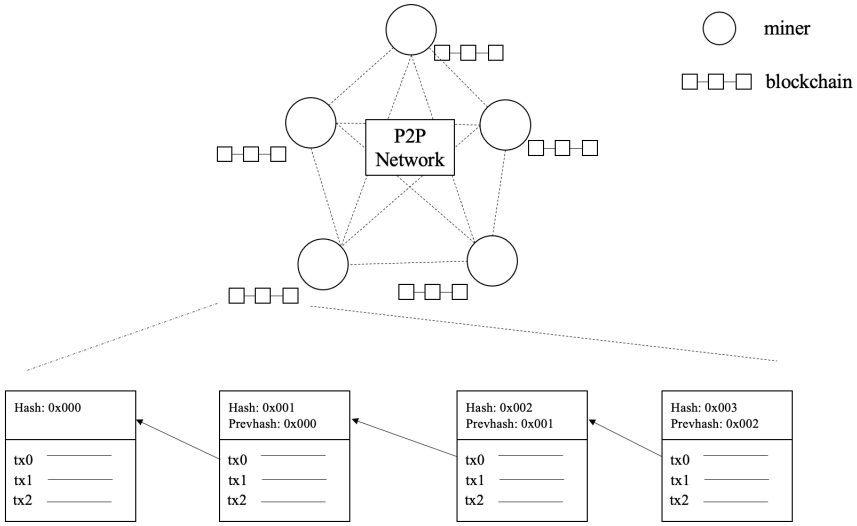


Figure 3.4: Authenticating a message with a digital signature.

3.4.1 Architecture

The blockchain is comprised of the following components:

- **transactions**: signed data structure defining an action on the blockchain.
- **blocks**: data structure that contains a group of transactions, information about the block itself such as the timestamp and a hash of the previous block.
- **blockchain**: data structure of linked blocks that have been validated.
- **node**: a participant in the network help to maintain the integrity of the blockchain. Nodes store a copy of the blockchain, validates transactions and helps to propagate transactions to the rest of the network.
- **miner**: a machine that can do anything a Node can, but also participates in the mining process to add a new block to the blockchain.
- **mining**: the sequence of steps miners performs to add the next block to the blockchain chain and generally are rewarded for it.

3.4.2 Transactions

The blockchain is underpinned by the transactions that take place within it. These transactions are created, validated, and added to the blockchain via the blockchain protocols. A transaction

represents a record of an action, which in the case of the Bitcoin Protocol, refers to the transfer of ownership of Bitcoin from one entity to another. To provide evidence that a transaction is authorized by the initiating entity, a digital signature is included. Transactions are public and can be verified by anyone within the network.

3.4.3 Merkle Trees

A Merkle Tree is a binary hash tree data structure [45]. They are used to efficiently summarize and verify the integrity of large sets of data. In blockchain, Merkle Trees are used to summary transactions in a block. To create a Merkle Tree, transactions are hashed in pairs until there remains only one hash, known as the Merkle Root (Figure 3.5). The Merkle Root is then included in the Block header. Another benefit of using Merkle Trees, is that is allows nodes to easily verify if a transaction is included in a block.

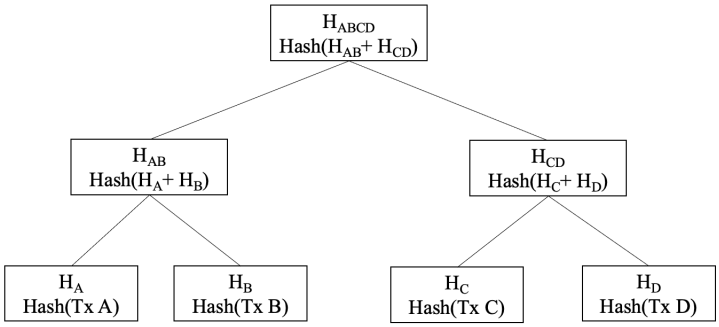


Figure 3.5: Building a Merkle Tree.

3.4.4 Keys and Addresses

In blockchain, keys and addresses play a fundamental role in establishing ownership and providing a secure and unique identity for users. They enable the transfer and storage of assets on the blockchain. Two types of keys are utilized: a private key and a public key. The private key is kept confidential and is employed to sign transactions, while the public key is distributed to others and is used to validate the authenticity of a signed transaction. An address is a string of alphanumeric characters that designates a specific location on the blockchain where assets can be sent and stored. A user's address is generated from their public key and is used to receive assets on the blockchain. Notably, private keys and addresses are not stored on the blockchain or any public network but are instead stored by the user in a file or database known as a wallet.

3.4.5 Blocks

A block is a data structure that stores transactions for inclusion in the blockchain. It is generally made up of two main parts: the header and transactions. The header describes information about the blockchain and includes data such as: a reference to the previous block in the form of a hash, metadata, and the transactions Merkle Root. The metadata itself contains more information about the block such as the timestamp, difficulty and nonce, a value we discuss in the next section.

A block can be identified in two ways. First is the block hash, which is generated by hashing the block header, thus creating a unique identifier. Second is the block height, a value that describes the sequence order in which the block was added to the blockchain. The block height; however, is not guaranteed to be unique like the block hash. This is due to the scenario when two or more blocks are at same position on the blockchain. The first block is commonly known as the Genesis Block.

The blockchain is extended with a new block as new incoming blocks are received by a node. When a node receives a new block, it first validates the block, then looks at the previous block hash and verifies it matches the current block's block hash. If so, the node stores the incoming block and is now linked, becoming the current block.

3.4.6 Mining and Consensus

The term "mining" is the process by which transactions are verified and permanently recorded on the blockchain in a decentralized manner. This process involves miners validating transactions, bundling them into a block, and competing to link the block to the existing blockchain. This competition is referred to as "mining" and the successful miner(s) are rewarded by minting new coins that are subsequently circulated.

The mining process entails solving a difficult mathematical problem based on hashing, which results in the creation of a Proof-of-Work (POW) [46]. This solution is then included in the block header for other miners to verify. In addition to the mining reward, the winning miner(s) also receive all of the transaction fees associated with the transactions in the block.

The ultimate goal of mining is to achieve consensus among nodes in the network, wherein each node has a local copy of the blockchain and through mining, arrives at the same outcome. Thus, the blockchain represents a single truth agreed upon by all participating nodes.

Consensus can be summarized into four processes, done by each node in the network indepen-

dently:

1. Verification of transactions.
2. Aggregation of transactions into blocks, included with the POW solution.
3. Verification of new blocks.
4. Choosing the chain with the most work when faced with blockchain forking.

In a blockchain network, every node is responsible for verifying transactions upon receipt. The verification process entails various checks such as validating the data structure and size, ensuring the reference to the previous transaction exists in the block, and adhering to specific criteria unique to the blockchain. Verified transactions that are not yet added to the blockchain are stored in a transaction pool, commonly referred to as a mempool.

During the mining process, a miner simultaneously receives transactions from the network and validates them as described above. Valid transactions that are included in the current block are removed from the mempool. Once the miner finds the solution for the current block, or when the miner receives the current block from the network, indicating another miner has found the solution, the miner begins to aggregate the transactions in the mempool into a candidate block. A candidate block is a block that has not been validated yet, meaning it does not have a valid POW proof.

The POW algorithm relies on hashing to obtain a hash value that is less than the target value. The miner achieves this by repeatedly appending a nonce to the block header hash and hashing it until a valid POW is found. To prevent cheating, each node in the network independently validates the new block against a set of criteria unique to the blockchain. The criteria include ensuring the block structure is correct, the block size is within limits, and the POW is valid. These criteria differ for each blockchain. The decentralized validation process is essential for achieving a consensus on the state of the blockchain.

In the Bitcoin blockchain, it takes approximately 10 minutes for miners to find a valid POW. Once a miner finds a valid POW, it becomes the block ID and is broadcasted to the miner's peers for validation. Any invalid block is discarded, and the miner's effort to create that block is wasted. Therefore, independent validation of new blocks is crucial for maintaining the integrity of the blockchain.

3.4.7 Smart Contracts

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 [47]. Ethereum [48] 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. Once on the blockchain, smart contracts reside at a specific address, are not controlled by any owner, and cannot be deleted.

Sequence of Smart Contract execution:

1. The Smart Contract is programmed and compiled into byte code by the compiler.
2. The contract is deployed as a transaction.
3. Miners validate the mined the transaction.
4. The transaction accepted into the blockchain, and the contract is assigned an address.
5. To use the contract, users submit transactions to the contract's address specifying which function they would like to call.
6. Once the transactions are validated and mined, each node in the network executes the smart contract function.

Smart contracts are written in a similar style to writing a class in object-oriented programming. In Ethereum, the contracts are written in a programming language called Solidity, though there are libraries available to use JavaScript and Python. Ethereum Virtual Machine (EVM) [49] based blockchains are compatible with Solidity. After a contract is written, it is compiled, then submitted as a transaction to the blockchain. The specific transaction is sometimes referred to as a contract deployment transaction and like any other transaction it must be verified by and propagated to all nodes on the network. Users interact with the program much like they would with public Application Programming Interface (API). Users submit a transaction using their wallet, which then calls and executes a specific function within the smart contract.

```
contract SimpleStorage {
  uint storedData = x;

  function set(uint x) public {
    storedData = x;
  }

  function get() public view returns (uint) {
    return storedData;
  }
}
```

Figure 3.6: Example of a smart contact.

3.5 Identity Management

3.5.1 Centralized and Federated Identity

The concept of decentralized identity revolves around placing the control of identity in the hands of its owner. This approach differs from the two existing identity management models: centralized and federated [50]. In a centralized model, 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.

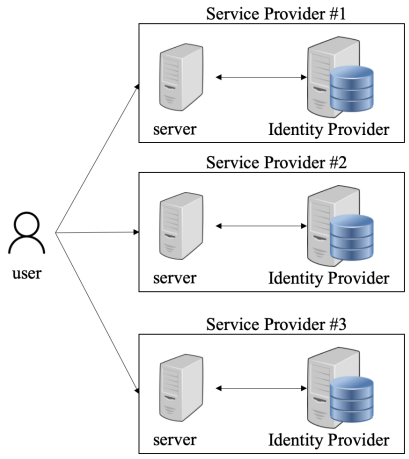


Figure 3.7: Centralized identity model.

In a federated model, two or more centralized systems establish trust allowing one system to authenticate an entity for another. 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 [51].

Single Sign-On (SSO) is a solution derived from Federated Identity, which allows users to access multiple systems and applications with a single set of login credentials [52]. 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 (JWT), 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 [52]. 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 [53]. This challenge additionally incurs cost due to the increased security infrastructure needed to share valuable information across domains using loosely coupled network protocols [51].

3.5.2 Decentralized Identity

Decentralized Identity is a new 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. The system is comprised of multiple components which we define below:

- **Decentralized Identifier (DID)**: a unique identifier that unlike traditional identifies that are created by a centralized service (i.e. an email), is created by the entity themselves [54]. DIDs are assigned a public/private key pair, thus can be digitally signed. Depending on the

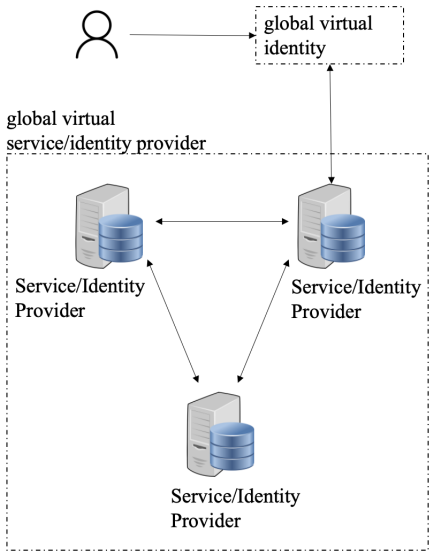


Figure 3.8: Centralized identity model.

entity purpose, DIDs can either be public or private [55]. Moreover, they can be used for establishing communication channels. The DID syntax has been defined by the W3C [56]. 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 string, generally a hash, that represents the DID. An example of a DID is:

did:btcr:xyv2-xzpq-qrst-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. DID documents 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 (Figure 3.10 [56]). Every DID has an accompanying DDO. The DDO can be stored on the blockchain itself or stored off-chain and mapped within the DID Resolver [55].

- **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]. Another

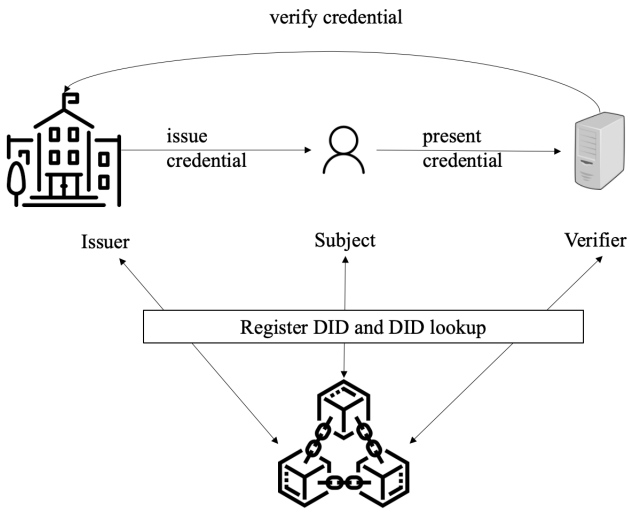


Figure 3.9: Decentralized identity model.

```
{
  "@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 3.10: Example DID Document.

advantage is that through blockchain's transactions and their hashes, 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. One ledger that is made just for creating and managing a decentralized identity network is Hyperledger Indy [57].

- **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 decentralized 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 [33] and [58] have used the Blockchain and Smart Contracts to implement a DID resolver.

- **Claims**: a claim is a statement about an individual or organization that can be proven or disproven. Claims are a key component of verifiable credentials and verifiable presentations 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.

- **Verifiable Credential (VC)**: A verifiable credential is a digital document that contains a set of claims about an individual or organization that can be independently verified by a third party. Verifiable credentials are designed to be self-sovereign, meaning 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 (e.g., degrees, licenses), and other relevant details (Figure 3.11 [56]). Verifiable credentials 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, VP's are not stored on public database or ledger, but rather on the user storage device and managed by a software known as a digital wallet [59].

- **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 verifiable presentations are derived from one or more credential. Another difference is that in order to prevent a replay attack [56], a VP contain both the cryptographic signature from the issuer who issued the VC and additional signature from subject who generated the VP. 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.

EXAMPLE 3: A simple verifiable claim


```
{
  "@context": "https://w3id.org/security/v1",
  "id": "http://example.gov/credentials/3732",
  "type": ["Credential", "ProofOfAgeCredential"],
  "issuer": "https://dmv.example.gov",
  "issued": "2010-01-01",
  "claim": {
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "ageOver": 21
  },
  "revocation": {
    "id": "http://example.gov/revocations/738",
    "type": "SimpleRevocationList2017"
  },
  "signature": {
    "type": "LinkedDataSignature2015",
    "created": "2016-06-18T21:19:10Z",
    "creator": "https://example.com/jdoe/keys/1",
    "domain": "json-ld.org",
    "nonce": "598c63d6",
    "signatureValue": "BavEll0/I1zpYw8XNi1bgVg/sCneO4Jugez8RwDg/+
    MCRVpjOboDoe4SxxKjkCOvKiCHGDvc4krqi6Z1n0UfqzxGfmatCuFibcC1wps
    PRdW+gGsutPTLzvueMWmFhwYmfIFpbBu95t501+rSLHIEuujM/+PXr9Cky6Ed
    +W3JT24="
  }
}
```


Figure 3.11: Example 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.

3.5.3 Current State of Decentralized Identity

Decentralized identity is a rapidly evolving field that is gaining increasing attention from both academia and industry. It can be implemented with or without a blockchain as long as it supports the necessary features and capabilities. The following are list networks and platforms being developed and used today: uPort[1] , Sovrin[2], IDchainZ[3], EveryID[4], SelfKey[5] and Civi[6]. The authors in [10] provide an evaluation of each of the mentioned platforms along with the challenges and limitations.

[1]https://www.uport.me/
[2]https://sovrin.org/
[3]https://www.chainzy.com/products/idchainz/
[4]https://www.everyid.com/
[5]https://selfkey.org/
[6]https://www.civic.com/

3.6 InterPlanetary File System

The InterPlanetary File System (IPFS) is a distributed file system that aims to make the internet more resilient and scalable [60]. 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 (Figure 3.12). The resulting CID is used to locate and retrieve the content. An example of a CID is QmV8RgHXhv7n68EoyYG5N8rGZn3vZ5z5LcN5Z8uAVhX9y7. Users can then use the CID to retrieve the content from the IPFS network.

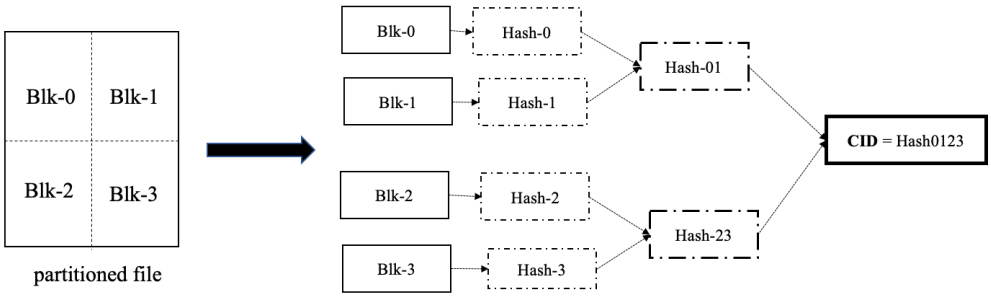


Figure 3.12: Creating a IPFS Content Identifier (CID).

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.

Chapter 4

Proposed Scheme

The purpose of this chapter is to provide a comprehensive introduction to DeA^{2}uth. To achieve this goal, we will offer a detailed explanation of the scheme's underlying concepts and architecture. DeA^{2}uth's concept consists of several critical components, including those responsible for data management, logic, and storage. In addition, we will delve into DeA^{2}uth's functions and algorithms that govern its operation.

4.1 Concepts

DeA^{2}uth is comprised of three main components: the wallet, smart contracts, and IPFS, which are unified by the blockchain (Figure 4.1). The wallet's main responsibilities are identity and blockchain address management. All participants must have a wallet to hold an identity and interact with smart contracts. The smart contracts contain DeA^{2}uth's business logic. There are two smart contracts, one to manage the creation and lookup of DIDs and the other to send messages and data. The last component is a data storage layer. Messages and data are not stored on the on the blockchain to minimize the memory growth rate. For DeA^{2}uth we use IPFS to store large pieces of data. Because IPFS is a distributed P2P network, all participants have access to store and retrieve data form it. The rest of this section describes each component in detail.

4.1.1 Wallet

The wallet is the control center for each participant in the network. It allows users to manage their identity and interact with the blockchain, including the smart contracts that run in the blockchain. Wallets are generally used as desktop, mobile or web application. Additionally, they have different

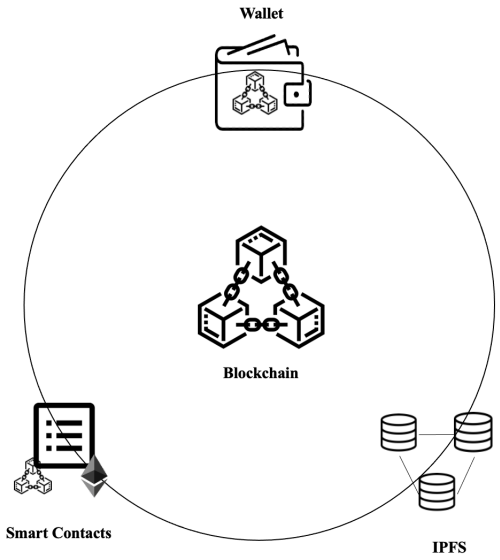


Figure 4.1: DeA^{2}uth components.

features depending on what type of system they are used in. For example, wallets in a decentralized identity system manage DIDs, while wallets in blockchain system manage blockchain addresses and submit transactions to the blockchain. Since DeA^{2}uth is a decentralized identity and blockchain application, our wallet requires features to handle both responsibilities. DeA^{2}uth wallet features include:

- Generating and storing public/private key pairs
- Storing addresses, DIDs and VCs
- Create, lookup and revoke DIDs
- Verify VCs and VPs
- Create blockchain address
- Communicate with the blockchain
- Interact with smart contracts

Key Management

DeA^{2}uth wallet is a deterministic wallet that organizes keys using Hierarchical Deterministic (HD) key generation [61]. In HD key generation, private and public keys are generated from a same

secret or seed. Moreover, the same pair of keys are generated given the seed has not changed. Non-deterministic wallets on the other hand generate each key from a different, randomly generated, seed, thus the same keys cannot be regenerated. Virtually and infinite number of keys can be generated. In order for our wallet to be compatible with multiple blockchains we follow wallet implementation which have been standardized such as BIP32 [62] and BIP44 [63].

A deterministic wallet is more suited for suited for DeA^{2}uth and other blockchain application than non-deterministic wallets for a few reasons. If a user deletes their wallet and the keys along with them, their seed is enough to recover the same keys. Keys are well organized into branches like files and folders on hard drive (Figure 4.2 [64]). Lastly, deterministic wallets can create public keys without the need for private, thus can be used on an insecure server.

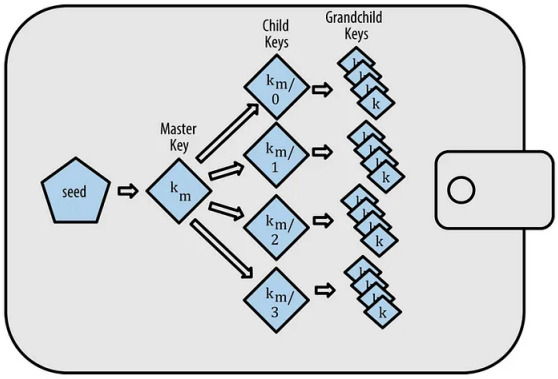


Figure 4.2: Hierarchical Deterministic (HD) key generation.

Address and Identity Management

User DIDs are derived from public keys and are stored in the digital wallet. Since DeA^{2}uth's wallet is managing both blockchain address and DIDs, which both are derived from public keys, we must have a way to separate the two. To solve this, we modified the BIP44 standard to accommodate keys that are used for creating DIDs. BIP44 is a technical specification for HD wallets. It defines a standard way of organizing the hierarchy of keys in an HD wallet, which 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 included when generating a

public/private key pair. An example path is m/44/60'/0/0/0. 44 represents the BIP44 standard and 60 is the constant value for the Ethereum Blockchain. More coin types can be found in [63].

In DeA^{2}uth we modified the levels to accommodate managing DIDs. First, we modify account such that even values represent accounts used for blockchain address and odd values represent accounts used for DIDs. Furthermore, we modify the change level. For a blockchain address, 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). 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 DeA^{2}uth, 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 in order to establish private communications between wallets.

Some example paths for blockchain address and DIDs are:

- 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

Agents

Agents are used in decentralized identity systems to facilitate interactions between users and the DeA^{2}uth network. They can be thought of as intermediaries that help users to manage their identity information and use it to prove their identity to other parties. Agents are built into the DeA^{2}uth's wallet software and can be accessed through APIs.

Services that agents perform are:

- Send and receive messages
- Encrypt and decrypt messages and files
- Authenticate and verify DIDs, VCs and VPs
- Manage wallet information

Services are made public and part of the DDO. An example of a service in a DDO shown in Figure 4.3.

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

Figure 4.3: Example service in a DDO.

4.1.2 Smart Contracts

DeA^{2}uth utilizes two smart contracts, the Identity contract and Authorization contract. Recall smart contracts enable interaction with the blockchain and allow users to change the state through sending transactions (Section 3.4.7). The Identity contract responsibilities are to create and lookup DIDs. While the Authorization contract is used to submit authorization messages that control private data sharing permissions.

There is only one Identity contract deployed; however, there can be many Authorization deployed. In DeA^{2}uth, the data managers (DMs) each deploy their own data contract. The address of the smart contract is included as a service in the DDO, so other user's wallet knows where to send transactions to (Figure 4.4).

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

Figure 4.4: Example smart contract in a DDO.

4.1.3 Data Storage and IPFS

DeA^{2}uth uses IPFS for storing authorization messages and files. Large amounts of data is expensive to store on the blockchain thus we use IPFS an off-chain storage layer. Only the DIDs and CIDs are stored on the blockchain.

4.1.4 Authorization Messages

Authorization messages are used to control data sharing permission in a secure and transparent manner. Additionally, the messages enable a comprehensive record of all the permissions requested and granted for data sharing between users. Messages are saved in IPFS and their CID submitted to the blockchain via the Authorization Contract. To uphold the privacy of users, the authorization messages are encrypted with a user's public key derived from a private DID before saved in IPFS.

Message Types

The message types describe what type of authorization message is being sent. There are five message types; however, can be extended to include more types if needed.

1. **RequestData**: request data from another user.
2. **AskPermission**: ask permission to transfer data that is not owned by them.
3. **AllowPermission**: owner allows the transfer of a user's private data.
4. **RejectPermission**: owner denies the transfer of a user's private data.
5. **SendData**: signify a data transfer and contains the location of the data being transferred.

Message Schema

The message schema is divided into two categories: non-data (Figure 4.5) and data (Figure 4.6). We use JavaScript Object Notation (JSON) [65] to define the schema. All except the SendData message type uses the non-data schema.

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

Figure 4.5: Non-data message schema.

The data schema uses the same property fields as the non-data schema, and also include fields for the CID of the encrypted data being shared, and the CID of the symmetric encryption key.

```
{
    to: the receiver's public DID
    from: the sender's public DID
    type: the message type
    encryptedData: CID of the encrypted data
    encryptionKey: CID of the symmetric encryption key
                   which has been encrypted with a private key
}
```

Figure 4.6: Data message schema.

4.1.5 Data and Encryption

Data is transferred only when AllowPermission is submitted by the owner. The data can be anything from files, images, mp3s or videos. It transferred to the IPFS network when ready to be shared. Prior to that, it is assumed to be stored on a privately owned database. Before data is saved to IPFS; however, it must be encrypted to ensure the the owner's privacy. Anything on IPFS is public and all participants in the network can retrieve the file. Data is encrypted with AES using a randomly generated symmetric key, k. This key is sometimes referred to as a session key [66].

After the data is encrypted, key k itself is encrypted and saved to IPFS in order for the requester to use k to decrypt the requested data. Wallet agents encrypts k with a public key extracted from a requester's DDO before it is saved to IPFS.

4.2 Architecture

DeA^{2}uth's architecture is divided into two schemes: authentication (Figure 4.7) and authorization (Figure 4.8). Authentication follows generational sequence between a Subject S, Issuer I and Verifier V.

1. S requests a VC from I
2. I issues a VC to S
3. V requests a VP from S
4. S presents VP to V
5. V verifies VP with I

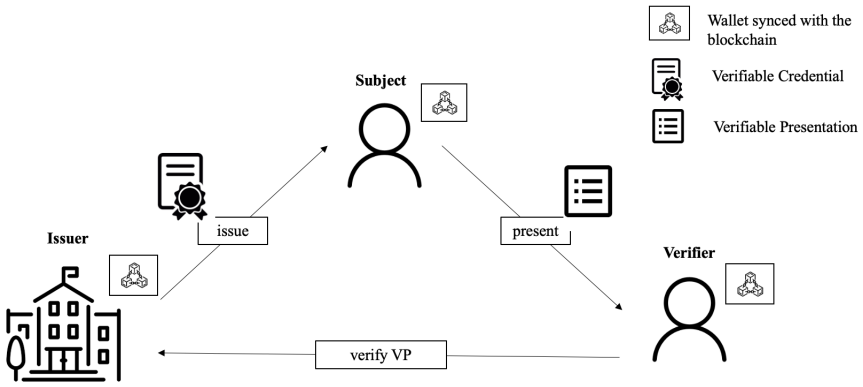


Figure 4.7: DeA^{2}uth Authentication Scheme.

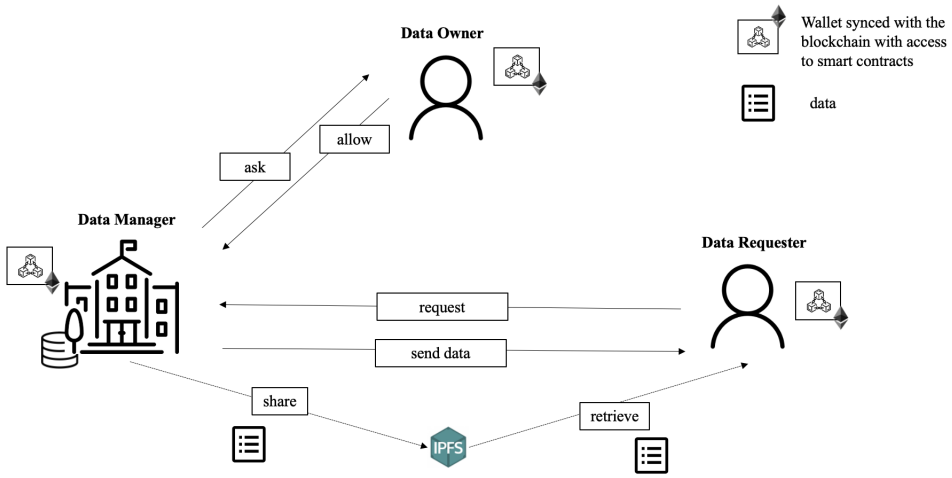


Figure 4.8: DeA^{2}uth Authorization Scheme.

The authorization scheme in Figure 4.8 illustrates how private data is transferred between users in a decentralized system. Transfer between data occurs after authentication has been performed. In the authorization scheme, there are three types of participants the Data Manager (DM), Data Owner (DO) and Data Requester (DR). The DM is an entity that manages and stores data containing PII about an DO. DMs cannot share this data without a DO's permission. The DR is the participant that is requesting data about a DO from a DM. The authorization sequence is listed below:

1. DR submits a *RequestData* authorization message via the authorization smart contract.
2. DM submits a *AskPermission* message.
3. DO submits a *AllowPermission* message.

4. DM encrypts and shares the data on IPFS.

5. DM submits a *SendData* message.

6. DR retrieves the data from IPFS and decrypts it.

4.2.1 Functions

In this section we define all major functions used in the DeA^{2}uth scheme and are categorized into 6 function categories: (1) general, (2) wallet, (3) smart contract, (4) identity, (5) authorization message and (6) data.

(1) General Functions:

- $sk, pk \leftarrow$ **genKeyPair**$(seed, path)$ - generates a private key (sk) and public key (pk) pair.
- $k \leftarrow$ **genSymmetricKey**$(seed)$ - generates a symmetric key for data encryption.
- $CID_m \leftarrow$ **lookupTID(TID)** - looks up a blockchain transaction ID (TID) on the blockchain and returns and embedded content Identifier (CID) associated with a message (m).

(2) Wallet Functions:

- **createPublicConn**$(pubDID, pubDID)$ - creates a public connection between two wallets using public DIDs ($pubDID$).
- **createPrivateConn**$(privDID, privDID)$ - creates a private connection between two wallets using private DIDs ($privDID$).
- $true|false \leftarrow$ **authenticateDID**$(pubDID|privDID)$ - authenticates a DID. Returns true if it the DID owner verifies the signature; false otherwise.
- $true|false \leftarrow$ **authenticateVP**(VP) - authenticates a VP. Return true if it the VP owner verifies the signature; false otherwise.
- **sendVPRequest**$(pubDID)$ - sends a VP request to a Subject.
- **send**$(pubDID, VP|privDID|TID)$ - sends a VP, privDID, or TID to another wallet using their public DID.

(3) Smart Contract Functions:

- $pubDID \leftarrow$ **publishDID**(DID, CID_{DDO}) - publishes a DID to the blockchain making it public. The DID is mapped to a DID Document (DDO) CID.
- $DID_{DDO} \leftarrow$ **lookupDID**$(pubDID)$ - returns a CID_{DDO} given a DID what has already been published.
- **submitMessage**$(t, CID_{m^{CT}})$ - submits an encrypted authorization message (m) to the blockchain. The function requires the message type (t) and the CID of the message $CID_{m^{CT}}$, which is encrypted before storing on IPFS.

(4) Identity Functions:

- $DID \leftarrow$ **createDID**(sk, pk) - creates a DID using a public key (pk). The DID is signed by a private or secret key (sk).
- $DDO \leftarrow$ **createDDO**(DID) - creates a DDO from a DID.
- $signature \leftarrow$ **extractProof**$(DDO|VP)$ - extracts the signature from a DDO or a VP and returns the signature. The signature is labeled as "proof" in the DDO.
- $true|false \leftarrow$ **verifyProof**$(signature, pk)$ - verifies a signature was digitally signed by a pk and return true if so; otherwise, returns false otherwise.
- $pk \leftarrow$ **extractPublicKey**(DDO) - extracts a pk from a DDO.

(5) Authorization Message Functions:

- $m_t \leftarrow$ **createMessage**$(t, pubDID_{sender}, pubDID_{receiver})$ - creates a non-data message given the type t, the sender's pubDID and the receiver's pubDID.
- $m_{SendReport} \leftarrow$ **createDataMessage**$('SendData', pubDID_{sender}, pubDID_{receiver}, CID_{f^{CT}}, CID_{k^{CT}})$ - creates a data message.
- $CID_m \leftarrow$ **saveMessageToIPFS**(m) - saves m to IPFS and returns the CID of m.
- $m \leftarrow$ **retrieveMessageFromIPFS**(CID_m) - retrieves a m from IPFS given the CID.
- $m^{CT} \leftarrow$ **encryptMessage**(pk, m) - encrypts m with pk.
- $m \leftarrow$ **deceryptMessage**(sk, m) - decrypts m with sk

(6) Data Functions:

- $f^{CT} \leftarrow$ **encryptFile**(k, f) - encrypts file f with symmetric key k.
- $f \leftarrow$ **decryptFile**(k, f^{CT}) - decrypts file f with symmetric key k.

4.2.2 Algorithms

In this section, we describe the algorithms used in Authentication and Authorization schemes. They combine multiple functions defined in the previous section.

- **Algorithm 1** creates DIDs and associated DDOs for all participants in the network and publishes them to the blockchain to make the public. The DDOs are stored on IPFS.
- **Algorithm 2** authenticates a DID by verifying the signature in the DDO is valid and signed by the proper sk.
- **Algorithm 3** exchanges private DIDs between two participants with DIDs: DID_i and DID_j
- **Algorithm 4** I sends a VC to S using a private connection.
- **Algorithm 5** S creates a VP and is validated by a V.
- **Algorithm 6** DR requests data about DO. The data is managed by DM.
- **Algorithm 7** after receiving a data request message, DM requires DO's permission.
- **Algorithm 8** DO allows permission for DM to transfer data to DR.
- **Algorithm 9** DM receives permission from DO to share with DR. DM encrypts and saves O's private data to IPFS.
- **Algorithm 10** DR downloads encrypted private data f^CT and encrypted symmetric key k^CT, decrypts k^CT then decrypts f^CT to obtain data f.

Algorithm 1 Create Public DIDs

1: **for** $n \in N$; where $i = S, I, V$ **do**
2: $sk_i, pk_i \leftarrow$ **genKeyPair**$(seed, path)$
3: $DID_i, DDO_i \leftarrow$ **createDID**(pk)
4: $CID_{DDO_i} \leftarrow$ **saveToIPFS**(DDO_i)
5: **publishDID**(DID_i, CID_{DDO_i})

Algorithm 2 Authenticate DID

$DDO \leftarrow \textbf{lookup}(DID)$
$DID, DDO \leftarrow \textbf{createDID}(pk)$
$signature_{DDO} \leftarrow \textbf{extractProof}(DDO)$
$true \mid false \leftarrow \textbf{verifyProof}(sk, signature_{DDO})$

Algorithm 3 Exchange Private DID

$sk, pk \leftarrow \textbf{genKeyPair}(seed, path)$
$DID, DDO \leftarrow \textbf{genDID}(pk)$
$pk \leftarrow \textbf{extractPublicKey}(DDO)$
$enc(DDO) \leftarrow \textbf{encrypt}(DDO)$
$\textbf{exchange}(DID_i, DID_j)$
$DDO \leftarrow \textbf{decrypt}(enc(DDO))$

Algorithm 4 Transfer Verifiable Credentials

$\textbf{createPrivateConnection}(DID_{priv,subject}, DID_{priv,issuer})$
$\textbf{sender.sendVerifiableCredential}(VC_{subject})$

Algorithm 5 Verify Verifiable Presentation

$VP \leftarrow \textbf{createVerifiablePresentation}(VC)$
$signature \leftarrow \textbf{extractProof}(VP)$
$true \mid false \leftarrow \textbf{verifyCredential}(signature, sk)$

Algorithm 6 Request Data

$m_{RequestData} \leftarrow \textbf{createMessage}('RequestData', pubDID_{DR}, pubDID_S)$
$pk_{pubDID_{DM}} \leftarrow \textbf{extractPublicKey}(pubDID_{DM})$
$m^{CT}_{RequestData} \leftarrow \textbf{encryptMessage}(pk_{privDID_{DM}}, m_{RequestData})$
$CID_{m^{CT}_{RequestData}} \leftarrow \textbf{saveToIPFS}(m^{CT}_{RequestData})$
$TID_{RequestData} \leftarrow \textbf{submitMessage}('RequestData', CID_{m^{CT}_{RequestData}})$

Algorithm 7 Ask Permission

$m^{CT}_{RequestData} \leftarrow \textbf{lookUpTID}(TID_{RequestData})$
$m^{CT}_{RequestData} \leftarrow \textbf{retrieveMessageFromIPFS}(CID_{m^{CT}_{RequestData}})$
$m_{RequestData} \leftarrow \textbf{decryptMessage}(sk_{pubDID_{DM}}, m^{CT}_{RequestData})$
$pubDID_s, pubDID_{DR} \leftarrow \textbf{extractDID}(m_{RequestData})$
$m_{AskPermission} \leftarrow \textbf{createMessage}('AskPermission', pubDID_{DR}, pubDID_S)$
$m^{CT}_{AskPermission} \leftarrow \textbf{encryptMessage}(pk_{privDID_S}, m_{AskPermission})$
$CID_{m^{CT}_{AskPermission}} \leftarrow \textbf{saveToIPFS}(m^{CT}_{AskPermission})$
$TID_{AskPermission} \leftarrow \textbf{submitMessage}('AskPermission', CID_{m^{CT}_{AskPermission}})$

Algorithm 8 Allow Permission

$m^{CT}_{AskPermission} \leftarrow$ **lookUpTID**$(TID_{AskPermission})$
$m^{CT}_{AskPermission} \leftarrow$ **retrieveMessageFromIPFS**$(CID_{m^{CT}_{AskPermission}})$
$m_{AskPermission} \leftarrow$ **decryptMessage**$(sk_{privDID_S}, m^{CT}_{AskPermission})$
$pubDID_s, pubDID_{DR} \leftarrow$ **extractDID**$(m_{AskPermission})$
$m_{AllowPermission} \leftarrow$ **createMessage**$('AllowPermission', pubDID_S, pubDID_{DR})$
$m^{CT}_{AllowPermission} \leftarrow$ **encryptMessage**$(pk_{privDID_{DM}}, m_{AllowPermission})$
$CID_{m^{CT}_{AllowPermission}} \leftarrow$ **saveToIPFS**$(m^{CT}_{AllowPermission})$
$TID_{AllowPermission} \leftarrow$ **submitMessage**$('AllowPermission', CID_{m^{CT}_{AllowPermission}})$

Algorithm 9 Send Data

$m^{CT}_{AllowPermission} \leftarrow$ **lookUpTID**$(TID_{AllowPermission})$
$m^{CT}_{AllowPermission} \leftarrow$ **retrieveMessageFromIPFS**$(CID_{m^{CT}_{AllowPermission}})$
$m_{AllowPermission} \leftarrow$ **decryptMessage**$(sk_{privDID_{DM}}, m^{CT}_{AllowPermission})$
$pubDID_s, pubDID_{DR} \leftarrow$ **extractDID**$(m_{AllowPermission})$
$k \leftarrow$ **generateEncryptionKey**$(seed)$
$f^{CT} \leftarrow$ **encryptData**$(seed)$
$f^{CT} \leftarrow$ **encryptKey**$(pk_{privDID_{DR}}, k)$
$CID_{k^{CT}} \leftarrow$ **saveToIPFS**(k^{CT})
$CID_{f^{CT}} \leftarrow$ **saveToIPFS**(f^{CT})
$m_{SendData} \leftarrow$ **createMessage**$('SendData', pubDID_S, pubDID_{DR}, CID_{k^{CT}}, CID_{f^{CT}})$
$m^{CT}_{SendData} \leftarrow$ **encryptMessage**$(pk_{privDID_{DR}}, m_{SendData})$
$CID_{m^{CT}_{SendData}} \leftarrow$ **saveToIPFS**$(m^{CT}_{SendData})$
$TID_{SendData} \leftarrow$ **submitMessage**$('SendData', CID_{m^{CT}_{SendData}})$

Algorithm 10 Retrieve Data

$CID_{m^{CT}_{SendData}} \leftarrow$ **lookUpTID**$(TID_{SendData})$
$m^{CT}_{SendData} \leftarrow$ **retrieveMessageFromIPFS**$(CID_{m^{CT}_{SendData}})$
$m_{SendData} \leftarrow$ **decryptMessage**$(sk_{privDID_{DR}}, m^{CT}_{SendData})$
$CID_{k^{CT}} \leftarrow$ **extractCID**$(m_{SendData})$
$CID_{f^{CT}} \leftarrow$ **extractCID**$(m_{SendData})$
$k^{CT} \leftarrow$ **retrieveMessageFromIPFS**$(CID_{k^{CT}})$
$f^{CT} \leftarrow$ **retrieveMessageFromIPFS**$(CID_{f^{CT}})$
$k \leftarrow$ **decryptKey**$(sk_{privDID_{DR}}, k^{CT})$
$m \leftarrow$ **decryptData**(k, f^{CT})

4.2.3 Sequences

This section list the interaction sequence between users for the authentication and authorization schemes. The section also includes the data sharing sequence that defines how private data is shared once authorization is complete.

Authentication

The authentication sequence occurs between I, S and V:

1. I, S and V initiate wallets.
2. I, S and V create public DIDs.
3. I, S and V exchange private DIDs.
4. I issues VC to S.
5. V sends a VP request S.
6. S creates a VP from a VC.
7. S sends the VP to VC.
8. V verifies the VP with I.

Authorization

The authorization sequence occurs between DR, DM and DO. Prior to this sequence, all parties have already authenticated each other and exchanged private DIDs.

1. DR creates a $m_{RequestData}$ with DM as the receiver.
2. DR encrypts $m_{RequestData}$ with the private DID-public key $pk_{privDID_{DR,DM}}$.
3. DR saves $m^{CT}_{RequestData}$ to IPFS and obtains $CID_{m^{CT}_{RequestData}}$.
4. DR submits $CID_{m^{CT}_{RequestData}}$ to the blockchain and obtains $TID_{RequestData}$.
5. DM uses $TID_{RequestData}$ to lookup $CID_{m_{RequestData}}$.
6. DM uses $CID_{m^{CT}_{RequestData}}$ to retrieve $m^{CT}_{RequestData}$ from IPFS.

7. DM decrypts $m^{CT}_{RequestData}$ with the private DID-private key $sk_{privDID_{DR,DM}}$.
8. DM creates a $m_{AskPermission}$ to DO.
9. DM encrypts $m_{AskPermission}$ with the private DID-public key $pk_{privDID_{DM,DO}}$
10. DM saves $m^{CT}_{AskPermission}$ to IPFS and obtains $CID_{m^{CT}_{AskPermission}}$.
11. DM submits $CID_{m^{CT}_{AskPermission}}$ to the blockchain and obtains $TID_{AskPermission}$.
12. DM and DO create a private connection.
13. DM sends DO the $TID_{AskPermission}$.
14. DO looks up $TID_{AskPermission}$ and retrieves $CID_{m^{CT}_{AskPermission}}$ from the blockchain transaction.
15. DO uses $CID_{m^{CT}_{AskPermission}}$ to retrieve $m^{CT}_{AskPermission}$ from IPFS.
16. DO decrypts $m^{CT}_{AskPermission}$ with the private DID-private key $sk_{privDID_{DR,DM}}$.
17. DO extracts $pubDID_{DM}$ from $m_{AskPermission}$ and authenticates $pubDID_{DM}$.
18. DO creates $m_{AllowPermission}$.
19. DO saves $m_{AllowPermission}$ to IPFS.
20. DO encrypts $m_{AllowPermission}$ with the private DID public key $pk_{privDID_{DM,DO}}$
21. DO saves $m^{CT}_{AllowPermission}$ to IPFS and obtains $CID_{m^{CT}_{AllowPermission}}$.
22. DO submits $CID_{m^{CT}_{AllowPermission}}$ to the blockchain and $TID_{AllowPermission}$ is returned.
23. DO and DM create a private connection.
24. DO sends DM the $TID_{AllowPermission}$.
25. DM looks up $TID_{AllowPermission}$ and retrieves $CID_{m^{CT}_{AllowPermission}}$ from the blockchain transaction.
26. DM uses $CID_{m^{CT}_{AllowPermission}}$ to retrieve $m^{CT}_{AllowPermission}$ from IPFS.
27. DM decrypts $m^{CT}_{AllowPermission}$ with the private DID secret key $sk_{privDID_{DM,DO}}$.
28. DM now the authorization from the DO to share their DO's private data.

Data Sharing

The data transfer sequence begins immediately the authorization sequence between the same participants.

1. DM retrieves requested data f from their private database.
2. DM generates a symmetric key k.
3. DM encrypts f with k.
4. DM encrypts k with $pk_{privDID_{DR}}$.
5. DM saves f^{CT} and k^{CT} to IPFS and obtains $CID_{f^{CT}}$ and $CID_{k^{CT}}$.
6. DM creates $m_{SendData}$.
7. DM encrypts $m_{SendData}$ with the private DID-public key $pk_{privDID_{DR,DM}}$.
8. DM submits $CID_{m^{CT}_{SendData}}$ the the blockchain and obtains $TID_{SendData}$.
9. DM and DR create a private connection.
10. DM sends $TID_{SendData}$ to DR.
11. DR looks up $TID_{SendData}$ and retrieves $CID_{m^{CT}_{SendData}}$ from the blockchain transaction.
12. DR uses $CID_{m^{CT}_{SendData}}$ to retrieve $m^{CT}_{SendData}$ from IPFS.
13. DR decrypts $m^{CT}_{SendData}$ with the private DID-private key $sk_{privDID_{DM,DR}}$.
14. DR uses $CID_{m_{SendData}}$ and retrieves $m_{SendData}$ from IPFS.
15. DR extracts $CID_{f^{CT}}$ and $CID_{k^{CT}}$.
16. DR retrieves f^{CT} and k^{CT} from IPFS.
17. DR decrypts k^{CT} with $sk_{privDID_{DR}}$ to obtain k.
18. DR decrypts f^{CT} with k.

4.3 Use Case Scenarios

In this section we apply DeA^{2}uth to three real-world use cases:

1. Undergraduate Transcript Verification
2. Credit Report Transfer
3. Electronic Healthcare Records Transfer

4.3.1 Use Case 1: Transcript Verification

Alice, would like to to apply to graduate school (Grad). The graduate school requires an official transcript from Alice's undergraduate school (UGrad) (Figure 4.9). The application process accepts transcripts as a $VC_{transcript}$. In this scenario Alice is the Subject Alice, the undergraduate school is the Issuer I and the graduate school is the Verifier V.

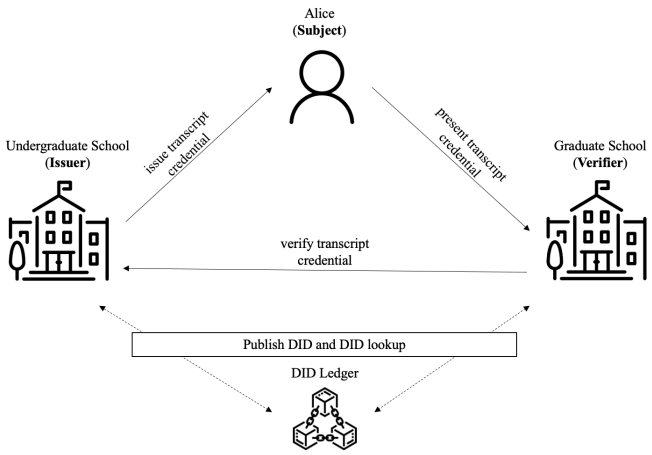


Figure 4.9: Use case 1 - transcript verification activity diagram.

Assumptions:

- All participants have a wallet.
- All participants have published a public DID and DDO.
- Alice has as a pre-registered account with the UGrad to receive a $VC_{transcript}$.

Sequence:

1. Alice signs into her undergraduate school account to request a $VC_{transcript}$ and provides her DID $pubDID_{Alice}$.

2. UGrad looks up $pubDID_{Alice}$ to retrieve $DDO_{DID_{Alice}}$
3. UGrad authenticates $pubDID_{Alice}$ using the public key $pk_{pubDID_{Alice}}$ provided in $DDO_{DID_{Alice}}$.
4. Alice looks up $pubDID_{UGrad}$ to retrieve the $DDO_{DID_{UGrad}}$
5. Alice authenticates $pubDID_{UGrad}$ using the public key provided in $DDO_{DID_{UGrad}}$.
6. Alice and UGrad create new DIDs to be used as a private DIDs $privDID_{S,I}$ and $privDID_{UGrad,Alice}$.
7. Alice encrypt $privDID_{Alice,UGrad}$ with $pk_{pubDID_{UGrad}}$ and sends it to UGrad.
8. UGrad encrypt $privDID_{UGrad,Alice}$ with $pk_{pubDID_{Alice}}$ and sends it to Alice.
9. Alice and UGrad create a private connection using $privDID_{Alice,UGrad}$ and $privDID_{UGrad,Alice}$
10. UGrad generates $VC_{transcript}$ for Alice.
11. UGrad sends $VC_{transcript}$ to Alice.
12. Alice stores $VC_{transcript}$ in her wallet.
13. Alice begins the the application and Grad is requiring $VC_{transcript}$.
14. Alice looks up $pubDID_{Grad}$ to retrieve $DDO_{DID_{Grad}}$.
15. Alice authenticates $pubDID_{Grad}$ using the public key $pk_{pubDID_{Grad}}$ provided in $DDO_{DID_{Grad}}$.
16. Grad looks up $pubDID_{Alice}$ to retrieve the $DDO_{DID_{Alice}}$.
17. Grad authenticates $pubDID_{Alice}$ using the public key $pk_{pubDID_{Alice}}$ provided in $DDO_{DID_{Alice}}$.
18. Alice and Grad create a new DID and DDO to be used as a private DIDs $privDID_{Alice,Grad}$ and $privDID_{Grad,Alice}$.
19. Alice encrypt $privDID_{Alice,Grad}$ with $pk_{pubDID_{Grad}}$ and sends it to UGrad.
20. Alice encrypt $privDID_{Grad,Alice}$ with $pk_{pubDID_{Alice}}$ and sends it to Alice.
21. Alice and Grad create a private connection using $privDID_{Alice,Grad}$ and $privDID_{Grad,Alice}$
22. Alice create $VP_{transcript}$ from $VC_{transcript}$.
23. Alice sends $VP_{transcript}$ to Grad.
24. Grad verifies the signature in $VP_{transcript}$.

4.3.2 Use Case 2: Credit Report Transfer

Alice is requesting a loan from the Mortgage Company (MC). MC needs to acquire a credit report form the Credit Report Company (CRC), in order to authorize a loan. However, before the CRC can send MC the report, the CRC requires Alice's permission. Once permission is given, the CRC sends the report to the MC. In this scenario, Alice is the Data Owner, the MC is the Data Requester and the CRC is the Data Manager (Figure 4.10). Additionally, for this scenario we skip the authentication sequence for brevity, but still occurs between all participants.

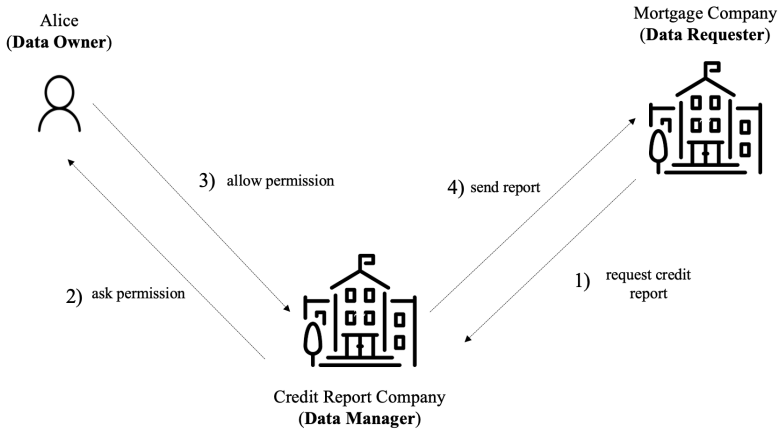


Figure 4.10: Use case 2 - credit report transfer diagram.

Assumptions:

- All participants have a wallet.
- All participants' public DID have been published to the blockchain.
- All participants have exchanged private DID between each other.
- An authorization smart contract has been deployed by CRC.

Sequence:

1. Alice applies for a loan from MC.
2. MC creates a RequestData message, $m_{RequestData}$.
3. MC encrypts $m_{RequestData}$ with the CRC's private DID-public key $pk_{privDID_{CRC,MC}}$.
4. MC saves $m^{CT}_{RequestData}$ to IPFS and obtains $CID_{m^{CT}_{Request}}$.

5. MC submits $CID_{m^{CT}_{RequestData}}$ to the blockchain using the authorization contract and obtains $TID_{ReqestData}$.
6. MC and CRC create a private connection and MC sends $TID_{ReqestData}$ to CRC.
7. CRC looks up $TID_{ReqestData}$ and obtains $CID_{m^{CT}_{RequestData}}$ from the blockchain transaction.
8. CRC uses $CID_{m^{CT}_{RequestData}}$ and retrieves $m^{CT}_{RequestData}$ from IPFS.
9. CRC decrypts $m^{CT}_{RequestData}$ with their private DID-private key $sk_{privDID_{CRC,MC}}$.
10. CRC creates an AskPermission message $m_{AskPermission}$.
11. CRC encrypts $m_{AskPermission}$ with private DID-public key $pk_{privDID_{CRC,S}}$.
12. CRC saves $m_{AskPermission}$ to IPFS and obtains $CID_{m_{AskPermission}}$.
13. CRC submits $CID_{m_{AskPermission}}$ to blockchain and obtains $TID_{AskPermission}$.
14. CRC and Alice create a private connection.
15. CRC sends Alice $TID_{AskPermission}$.
16. Alice looks up $TID_{AskPermission}$ and obtains $CID_{m^{CT}_{AskPermission}}$ from the blockchain transaction.
17. Alice decrypts $m^{CT}_{AskPermission}$ with their private DID-private key $sk_{privDID_{Alice,CRC}}$.
18. Alice accepts and creates $m_{AllowPermission}$.
19. Alice encrypts $m_{AllowPermission}$ with private DID-public key $pk_{privDID_{Alice,CRC}}$.
20. Alice saves $m_{AllowPermission}$ to IPFS and obtains $CID_{m_{AllowPermission}}$.
21. Alice submits $CID_{m_{AllowPermission}}$ to blockchain and obtains $TID_{AllowPermission}$.
22. CRC generates symmetric encryption key k.
23. CRC encrypts the credit report f with k.
24. CRC encrypts the k with $pk_{privDID_{CRC,MC}}$.
25. CRC saves f^{CT} to IPFS and obtains $CID_{f^{CT}}$.

26. CRC saves k^{CT} to IPFS and obtains $CID_{k^{CT}}$.

27. CRC creates a SendData message, $m_{SendData}$

28. CRC saves $m_{SendData}$ to IPFS and obtains $CID_{m_{SendData}}$

29. CRC encrypts $m_{SendData}$ with private DID-public key $pk_{privDID_{MC,CRC}}$.

30. CRC submits $CID_{m^{CT}_{SendData}}$ to blockchain and obtains $TID_{SendData}$.

31. CRC and MC create a private connection.

32. CRC sends MC $TID_{SendData}$.

33. MC looks up $TID_{SendData}$ and obtains $CID_{m^{CT}_{SendData}}$ from the blockchain transaction.

34. MC decrypts $m^{CT}_{SendData}$ with their private DID-private key $sk_{privDID_{MC,CRC}}$.

35. MC extracts $CID_{f^{CT}}$ and $CID_{k^{CT}}$ from $CID_{m_{SendData}}$.

36. MC retrieves f^{CT} and k^{CT} from IPFS.

37. MC decrypts k^{CT} with $sk_{privDID_{MC,CRC}}$.

38. MC decrypts f^{CT} with k and now has Alice's credit report.

4.3.3 Use Case 3: Electronic Health Record Transfer

In this scenario there are two hospitals, Hospital A (H_A), and Hospital B (H_B). Alice registers with Hospital H_A using her a driver's license VC which she obtained from the Department of Motor Vehicles (DMV). After some time, Alice then wants to transfer to H_B. Consequently, H_B requests Alice's patient records from Alpha; however, H_A requires Alice's permission to do so. For this scenario, Alice is the Subject, H_A is the Data Manager and H_B is Data Requester. The EHR request and transfer scenario are represented in Figure 4.11 and Figure 4.12. Like the scenario we skip the details of DID authentication sequence, but it still occurs between all participants.

Assumptions:

- All participants have a wallet.
- All participants have a public DID published to the blockchain.

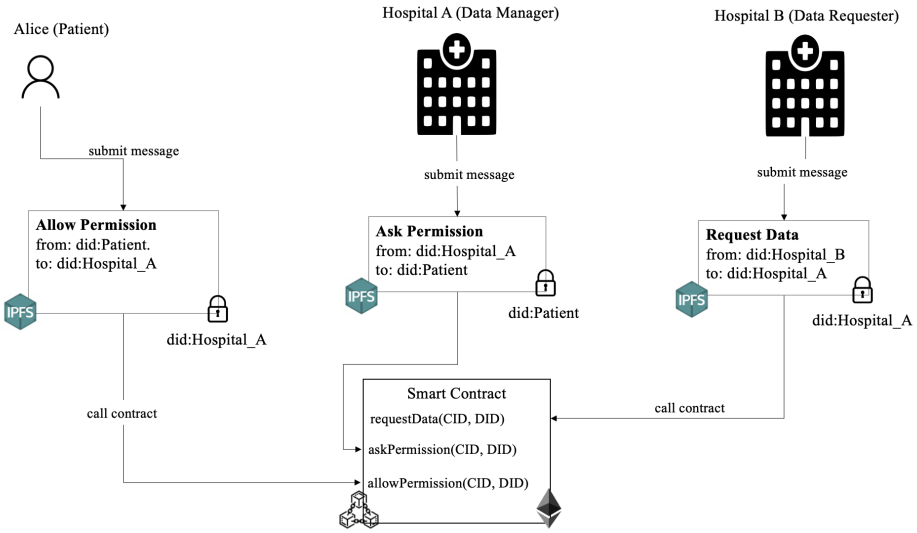


Figure 4.11: Use case 3 - EHR request diagram.

- An authorization smart contract has been deployed by H_A and is the owner of the contract.

Sequence:

1. Alice applies to be a patient at H_A by entering $pubDID_{Alice}$ into the patient registration form.
2. Alice and H_A authenticate their public DID, exchange private DIDs and create a private connection.
3. H_A sends a message requesting Alice to provide a VC.
4. Alice selects her $VC_{license}$ and creates a $VP_{license}$.
5. Alice signs $VP_{license}$ with $sk_{pubDID_{Alice}}$.
6. Alice sends $VP_{license}$ to H_A
7. H_A verifies Alice's and the DMV's signature that are embedded in $VP_{license}$.
8. H_A sends Alice's a new patient credential $VC_{patient}$.
9. H_B creates a RequestData message, $m_{RequestData}$.
10. H_B encrypts $m_{RequestData}$ with the private DID-public key $pk_{privDID_{H_A,H_B}}$.
11. H_B saves $m^{CT}_{RequestData}$ to IPFS and obtains $CID_{m^{CT}_{Request}}$.

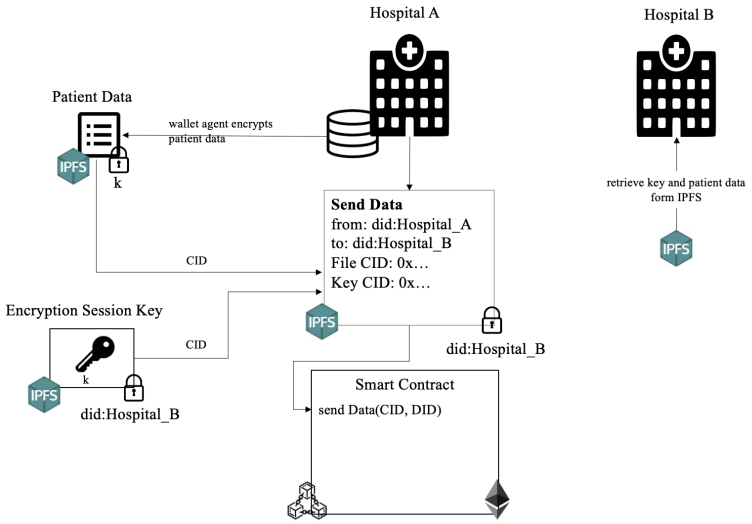


Figure 4.12: Use case 3 - EHR transfer diagram.

12. H_B submits $CID_{m^{CT}_{RequestData}}$ to the blockchain using the authorization contract and obtains $TID_{ReqestData}$.

13. H_B and H_A create a private connection and H_B sends $TID_{ReqestData}$ to H_A.

14. H_A looks up $TID_{ReqestData}$ and obtains $CID_{m^{CT}_{RequestData}}$ from the blockchain transaction.

15. H_A uses $CID_{m^{CT}_{RequestData}}$ and retrieves $m^{CT}_{RequestData}$ from IPFS.

16. H_A decrypts $m^{CT}_{RequestData}$ with their private DID-private key $sk_{privDID_{H_A,H_B}}$.

17. H_A creates an AskPermission message $m_{AskPermission}$.

18. H_A encrypts $m_{AskPermission}$ with Alice's private DID-public key $pk_{privDID_{H_A,Alice}}$.

19. H_A saves $m_{AskPermission}$ to IPFS and obtains $CID_{m_{AskPermission}}$.

20. H_A submits $CID_{m_{AskPermission}}$ to blockchain and obtains $TID_{AskPermission}$.

21. H_A and Alice create a private connection.

22. H_A sends Alice $TID_{AskPermission}$.

23. Alice looks up $TID_{AskPermission}$ and obtains $CID_{m^{CT}_{AskPermission}}$ from the blockchain transaction.

24. Alice decrypts $m^{CT}_{AskPermission}$ with their private DID-private key $sk_{privDID_{Alice,H_A}}$.

25. Alice accepts and creates $m_{AllowPermission}$.

26. Alice encrypts $m_{AllowPermission}$ with $pk_{privDID_{Alice,H_A}}$.

27. Alice saves $m_{AllowPermission}$ to IPFS and obtains $CID_{m_{AllowPermission}}$.

28. Alice submits $CID_{m_{AllowPermission}}$ to blockchain and obtains $TID_{AllowPermission}$.

29. H_A generates symmetric encryption key k.

30. H_A encrypts the credit report f with k.

31. H_A encrypts the k with $pk_{privDID_{H_A,H_B}}$.

32. H_A saves f^{CT} to IPFS and obtains $CID_{f^{CT}}$.

33. H_A saves k^{CT} to IPFS and obtains $CID_{k^{CT}}$.

34. H_A creates a SendData message, $m_{SendData}$

35. H_A encrypts $m_{SendData}$ with private DID-public key $pk_{privDID_{H_B,H_A}}$.

36. H_A saves $m_{SendData}$ to IPFS and obtains $CID_{m_{SendData}}$

37. H_A submits $CID_{m_{SendData}}$ to blockchain and obtains $TID_{SendData}$.

38. H_A and H_B create a private connection.

39. H_A sends H_B $TID_{SendData}$.

40. H_B looks up $TID_{SendData}$ and obtains $CID_{m^{CT}_{SendData}}$ from the blockchain transaction.

41. H_B decrypts $m^{CT}_{SendData}$ with their private DID-private key $sk_{privDID_{H_B,H_A}}$.

42. H_B extracts $CID_{f^{CT}}$ and $CID_{k^{CT}}$ from $CID_{m_{SendData}}$.

43. H_B retrieves f^{CT} and k^{CT} from IPFS.

44. H_B decrypts k^{CT} with $sk_{privDID_{H_B,H_A}}$.

45. H_B decrypts f^{CT} with k and now has *Alices*'s patient records.

Chapter 5

Implementation

The aim of this chapter is to provide an overview of the tools and methods that were utilized in the development of a prototype application. The initial version of the application utilized a centralized service to simulate the functionality of a blockchain as well as data storage. However, in the subsequent iteration, decentralized services were integrated to enable the application to operate in a decentralized manner. In this chapter, we will offer an in-depth discussion of the various tools and methods used during the development process, including the rationale behind their selection and implementation. The following technologies were used to develop the application:

- Next.js[1] - a React framework to create to web applications.
- Firebase[2] – Google's back end as a service and cloud storage service.
- Web3.Storage[3] – a service to store files accessible via IPFS.
- Ganache[4] – a local Ethereum blockchain node.

5.1 Application Development

5.1.1 Wallet

The bulk of the development efforts were dedicated to designing and implementing the wallet interface and its associated functionalities. The user interface (UI) of the wallet was built utilizing

[1]https://nextjs.org/
[2]https://firebase.google.com/
[3]https://web3.storage/
[4]https://trufflesuite.com/ganache/

Next.js and segmented into three distinct display sections: messaging, DIDs/VCs, and a specialized area dedicated to the presentation of specific details such as VC signatures (Figure 5.1).

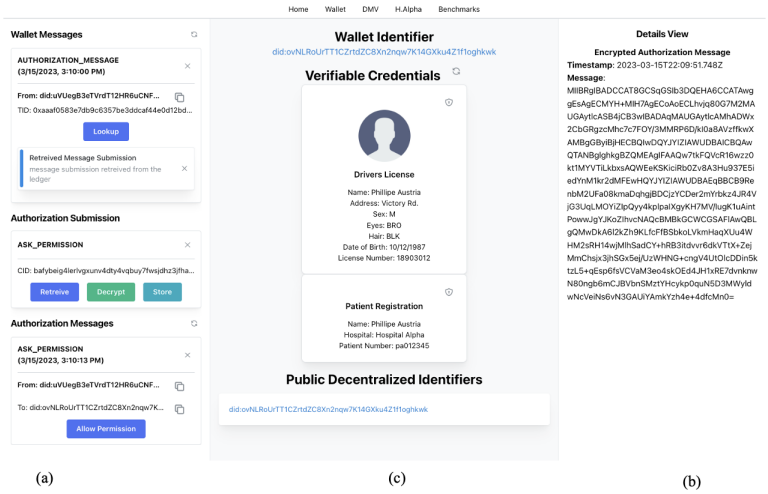


Figure 5.1: Wallet user interface with three sections: messaging (a), DIDs/VCs, and a details view (c) which shows information about the message or DIDs/VCs.

The messaging section of the wallet interface presents P2P wallet messages originating from other wallets, as well as messages regarding authorization message submissions to the blockchain and the authorization n messages themselves. These wallet messages serve to alert the user that another wallet has requested to verify their DID or VC. Furthermore, wallet messages notify the user when an authorization message has been submitted to the blockchain and they are the intended recipient. Each message is associated with specific actions that can be undertaken in response. For instance, as depicted in Figure 5.2, 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 5.1.

Table 5.1: List of wallet messages, 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

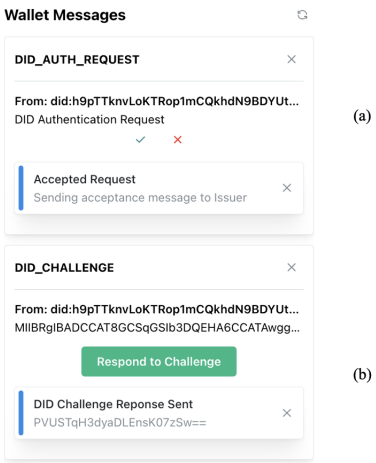


Figure 5.2: Example wallet messages. The wallet has been sent and accepted a DID authentication request (a). In response, the wallet is sent a challenge in which wallet responds to validate the DID (b).

The authorization submissions section of the wallet interface displays the authorization message submissions that have been retrieved from the blockchain. Upon being designated as the receiver of an authorization message, the wallet is notified via a corresponding wallet message. The TID associated with the authorization message is subsequently retrieved from the blockchain and presented within the authorization submission section of the wallet interface, see Figure 5.3(a). The process of authorizing the submission entails three discrete operations: (1) accessing the authorization message from IPFS, (2) decrypting the message with the associated DID private key, and (3) storing the decrypted message within the wallet. The operations are show in Figure 5.3(b).

The authorization message section of the wallet interface presents plain text authorization messages, such as RequestData, AskPermission, AllowPermission, and others, after their retrieval from IPFS and subsequent decryption. Each message type is associated with a particular set of actions that can be taken in response. For example, when a RequestData message is received, a button is displayed enabling the user to seek permission from the DO to share their data. Figure 5.4 comprehensively catalogs the various types of authorization messages, along with their associated actions.

The DID/VC section of the wallet UI presents the DIDs and VCs that are associated with the wallet. Upon the creation of a new wallet, a key pair is generated, with the public key being encoded in base58 to yield the wallet's public DID. Although the prototype did not include the capability to create new public or private DIDs, this feature is anticipated to be incorporated in

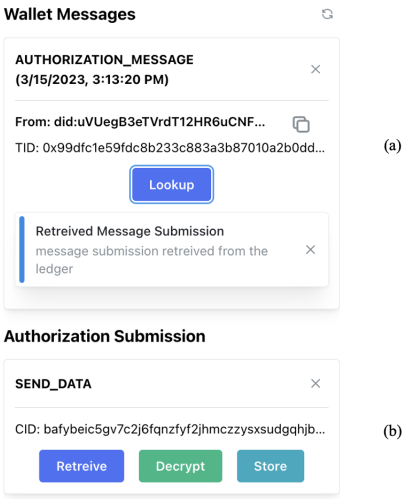


Figure 5.3: Screenshot of receiving a SendData authorization message. Message is first looked up from the blockchain (a). Then the Send Data submissions gives options to download the message from IPFS, decrypt and store (save) it to the wallet.

future iterations of the application and saved for future works.

5.1.2 P2P Wallet Messaging

The current iteration of DeA^{2}uth did not encompass the development of a fully functional P2P wallet messaging system, as this aspect was not the central focus of the project. Nevertheless, we do intend to integrate this feature into future iterations of the application, drawing upon the specifications and protocols outlined in the DIDComm[5] 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, API library was used that enabled the implementation of calls to store and retrieve data from Firebase. In

[5]https://identity.foundation/didcomm-messaging/spec/

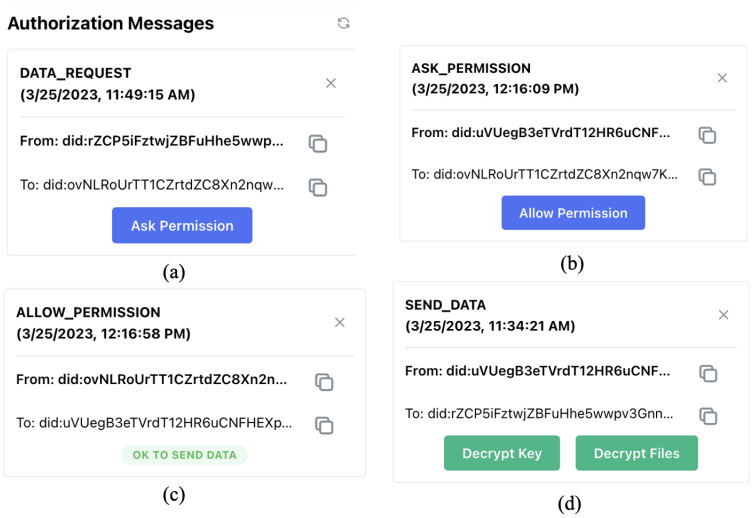


Figure 5.4: Various authorization messages displayed in the Authorization Message section of the. Data_Request (a), Ask_Permission (b), Allow_Permission (c) and Send_Data (d).

essence, these calls simulated smart contract calls to the blockchain.

In the subsequent iteration of our project, 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[6] editor, as detailed in Appendix C, Figure C.1 and Figure C.2. Initially, we conducted testing using Ganache, which is a local Ethereum node. Thereafter, we conducted testing on the 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 subjects to obtain a digital version of their driver's license, which is the VC. Figure 5.5 shows a screen of the prototype where the subject enters their wallet DID to receive VC. After the DID is submitted the DID owner will receive a message to begin the DID authentication process.

Upon obtaining the VC, it is displayed in the central portion of the wallet interface (refer to Figure 5.1). 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. These details include the digital signature that was generated by the DMV's private key, as depicted in Figure 5.6. Upon clicking the lock symbol on the credential,

[6]https://remix.ethereum.org/

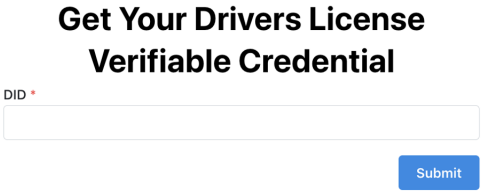
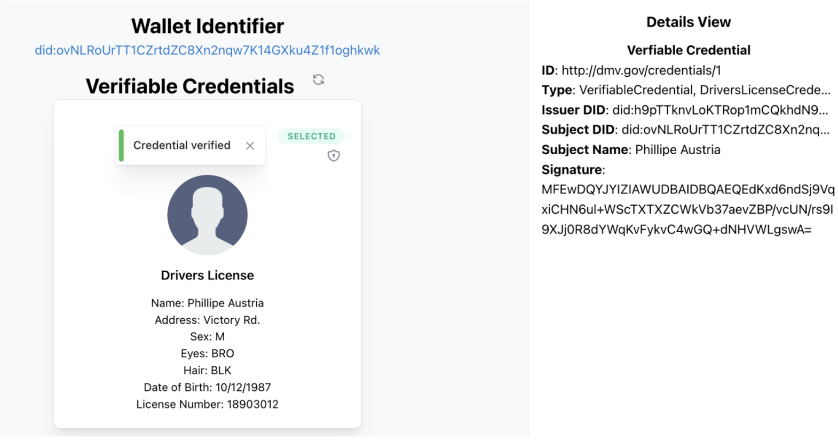


Figure 5.5: Screenshot of DMV page to obtain a driver's license VC.

the VC verification process is initiated, whereby the DMV validates the signature with their private key. No wallet message is generated during this process.



(a) (b)

Figure 5.6: Screenshot of a verified driver's license VC from the DMV (a). The VC details, including the signature, show in the Details View of the wallet (b).

5.1.5 EHR Implementation

In the prototype, we have implemented the use case scenario for transferring EHRs (as discussed in Section 4.3.3). This implementation comprises three primary functionalities, which are illustrated in Figure 5.7: (1) patient registration, (2) patient data request and (3) sending patient data to another hospital.

In this scenario subject registers with H_A. H_B then will request patent records from H_A. The subject first registers as a patient with H_A and obtains a VC as proof of registration, $VC_{patient}$. While actual registration generally requires the patient to provide detailed medical information, in this scenario, 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 $VC_{drivers-license}$,

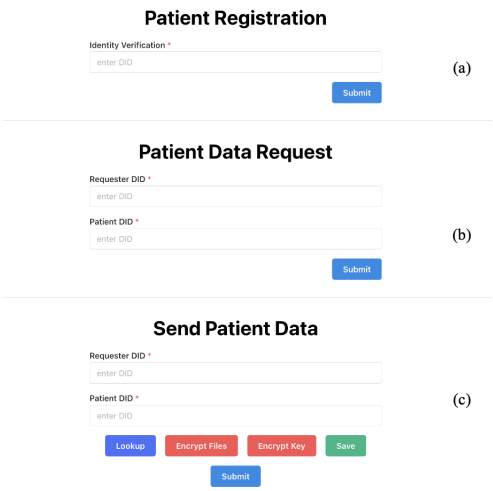


Figure 5.7: Screenshot of the various forms: patient registration form (a). patient data request form (b). sharing patient EHRs form (c)

generates $VP_{drivers-license}$ by signing the $VC_{drivers-license}$ with their private key and sends it to H_A, shown in Figure 5.8(a)(b). H_A validates both patient's signature and the DMV's signature. Once $VP_{drivers-license}$ is validated, H_A sends the patient $VC_{patient}$, shown in Figure 5.8(c).

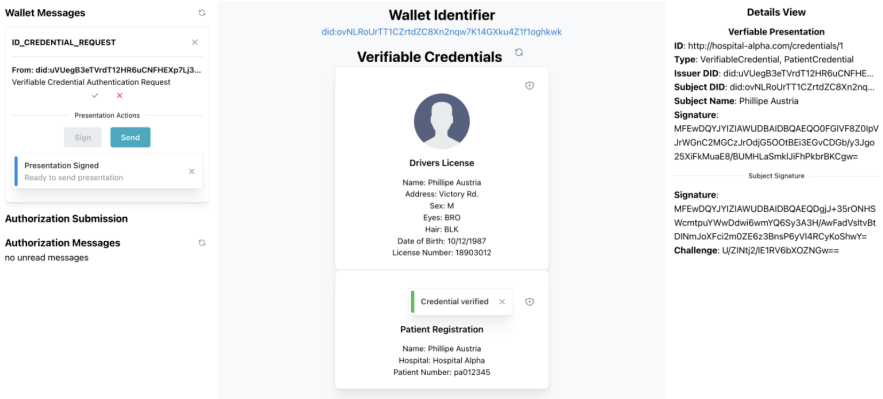


Figure 5.8: Screenshot of receiving a $VC_{patient}$. Patient signs and sends $VP_{drivers-license}$ (a). The patients signature displays in the Details View section (b). After signature validation, H_A sends the $VC_{patient}$.

5.1.6 Important Libraries and Source Code

Table 5.2 provides an overview of the important libraries and their corresponding versions utilized in the development of the DeA^{2}uth 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, although they did not directly contribute to the DeA^{2}uth project. All these libraries can be searched and accessed through npmjs.com[7] .

Table 5.2: Important libraries used for the DeA^{2}uth prototype.

Library Name	Version	Description
abi-decoder	2.4.0	accept or reject & helper library to decode smart contract call input
base-58	0.0.1	provides helpers 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 determinist key library to generate BIP32 compliant wallet keys
truffle/hdwallet-provider	2.1.8	library used to create a Ethereum provider that autosigns transactions
virgi-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

The source code for the smart contracts and important functions such as validating digital signatures can be found in Appendix C. The entire prototype is hosted on Gitlab and is available for the public to experiment with[8].

5.2 Implementation Analysis

5.2.1 Blockchain

DeA^{2}uth 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 [67, 68]. The 51% attack is one of the most significant security threats to blockchains and is also relevant to DeA^{2}uth [69]. The approach taken to carry out such an attack depends on the consensus mechanism employed by the blockchain.

The DeA^{2}uth 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) [70]. To control 51% of the staked ETH, the attackers would need to possess a con-

[7]https://www.npmjs.com/
[8]https://gitlab.com/paustria/unlv-dissertation

siderable amount of Ether (ETH), which is held by validator nodes. Currently, there are 222,052 validator nodes with a combined 7,105,596 ETH. To conduct a 51% attack, an attacker would require at least 3,552,798 ETH, which is valued at approximately $6,316,874,844 based on today's ETH price of $1,778 per ETH.

Moreover, the POS consensus mechanism provides an opportunity for the community to mount a counterattack against a 51% attack [70]. 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.

Attacks on a blockchain network are inevitable, and the likelihood of an attack increases for public, non-permissioned blockchains such as Ethereum. Therefore, it is crucial to select a blockchain with a healthy number of nodes and to use a well-researched and extensively tested consensus mechanism.

5.2.2 Wallet

In DeA^{2}uth, 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 DeA^{2}uth'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 [56]. 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 [30].

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. Furthermore, recent research has proposed using the Shamir Secret Sharing algorithm to recover forgotten keys [71, 72].

5.2.3 Encryption and Data Reliability

By default, data is not encrypted on IPFS, which was intentional by its creator to prevent application developers from being limited due to a lack of modularity, flexibility, and future-proofing [73]. However, in DeA^{2}uth, 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 [41, 74]. 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 [35]. In DeA^{2}uth, 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.

In the context of DeA^{2}uth, an additional consideration relates to the reliability of the storage network. 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, which are commonly included in Service Level Agreements of storage providers such as Amazon S3, Dropbox, and Google Drive. A study conducted by [75] found that Sia, one of the earliest blockchain-based storage services, exhibited similar redundancy, durability, and availability factors as centralized cloud services. However, it is important to note that these factors were primarily dependent on the storage implementation rather than the blockchain itself.

For our prototype we utilized Web3.Storage, a service that uses both IPFS and Filecoin, a blockchain based storage network [76]. Web3.Storage maintains its own IPFS nodes to ensure availability and redundancy and employs the Filecoin network for durability. However, specific information regarding the number of IPFS nodes they host, and deals made on Filecoin are not disclosed in the Web3.Storage documentation. For instance, to achieve a level of redundancy that is comparable to Google Cloud, they would need to replicate data at least twice.

In practical applications, it is expected that a DR would store the data on their own hard drive or storage network once they receive the files from the DM. Furthermore, while the message submission transactions are permanently recorded on the blockchain, it is advisable for all participants to retain the authorization messages for the auditing purposes and keeping a historical record.

Chapter 6

Performance Evaluation

The purpose of this chapter is to offer a comparative analysis of the primary operations involved in DeA^{2}uth, 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 will present a detailed examination of both the timing and cost results for each prototype. Table 6.1 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 DeA^{2}uth.

Table 6.1: Operations and services evaluated.

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

6.1 Timing

Figure 6.1 shows the average time to save and retrieve authorization messages from storage. Saving to Web3.Storage took approximately 1.4 seconds longer than saving to Firebase. Retrieving messages were comparable between the two.

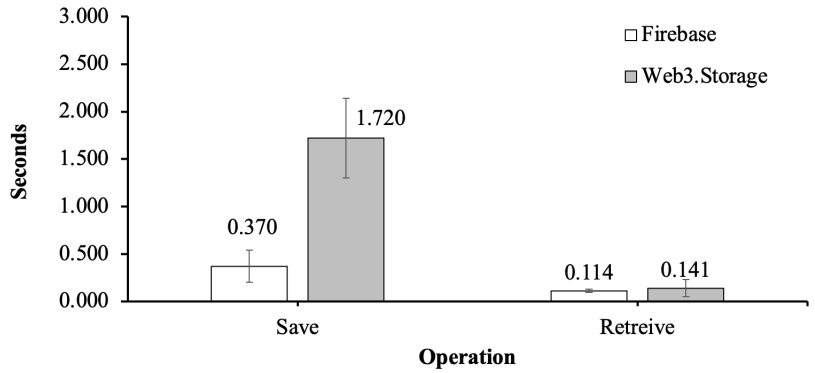


Figure 6.1: Operation time to save and retrieve authorization messages from storage.

The results of comparing the average time for submitting and retrieving an authorization message transaction between the centralized and decentralized versions of DeA^{2}uth are presented in Figure 6.2. It is worth noting that authorization messages are saved to storage and subsequently submitted to the blockchain. The use of Firebase resulted in a significantly faster submission time of 0.370 seconds, whereas Ethereum took 15.101 seconds. On the other hand, retrieving a transaction from Ethereum was faster, taking only 0.283 seconds, but it was still slower than Firebase, which took only 0.114 seconds.

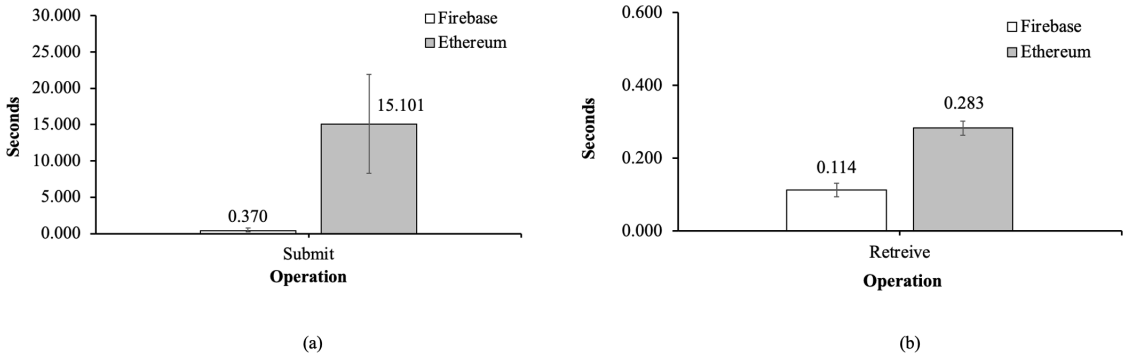


Figure 6.2: Operation time to submit (a) and retrieve (b) authorization message transactions.

Figure 6.3 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 1MB, 10MB, and 1000MB, 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 1000MB file, Firebase was on average approximately 59 seconds faster than Web3.Storage. However, for the 100MB file, Firebase surprisingly took longer to upload than Web3.Storage. Possible reasons for this unexpected

observation will be discussed in the subsequent discussion section.

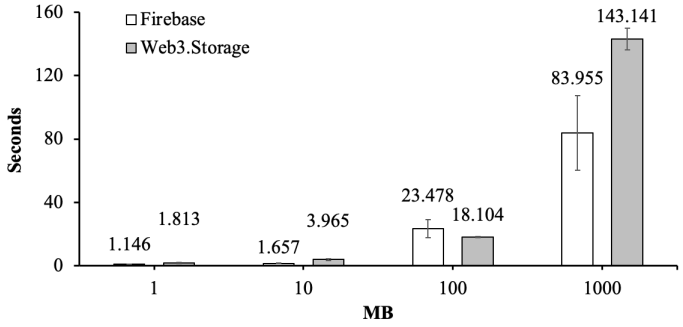


Figure 6.3: Upload times of various file sizes to storage.

Figure 6.4 shows average time to download various sized files to local storage. Additionally, we also setup our own IPFS node and measured the time to download the 100MB and 1000MB file. 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 100MB and 1000MB files.

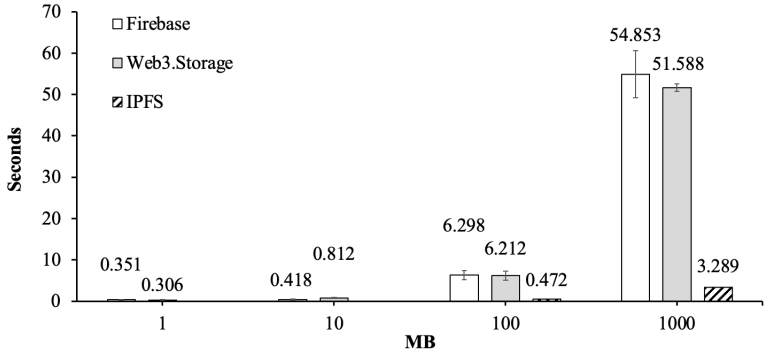


Figure 6.4: Download time of various files sizes from storage.

6.2 Costs

In this section, we report on the cost of DeA^{2}uth's main operation transactions, which fall into two categories: deploying contracts and smart contract function calls. The reported ETH amounts include miner gas, and the exchange rate at the time of testing was \$1719.74/ETH. All costs are reported in USD.

Table 6.2 presents the cost of performing DeA^{2}uth'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 6.2: Operation costs 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 6.3 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. For comparative purposes, we assumed that the Data Requester would want to download the data after receiving it from the Data Manager. 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. In contrast, Web3.Storage charges a higher rate of $0.08/GB*mo for storage but does not charge for download bandwidth.

Table 6.3: Cost of storing data.

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

Chapter 7

Discussions

The objective of this chapter is to conduct an evaluation of the timing and cost results obtained from the tests performed in the previous chapter. By analyzing these metrics, we aim to provide readers with a comprehensive understanding of the trade-offs and implications of our different approaches to implementing DeA^{2}uth.

7.1 Timing Evaluation

The research community is well aware of the performance limitations of the blockchain [77, 78]. 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 seconds and can process 27-30 TPS. The impact of these limitations is reflected in Figure 6.1, where it takes approximately 15 seconds to submit an authorization message transaction on the Ethereum blockchain, whereas it occurs nearly instantaneously using Firebase.

In the context of DeA^{2}uth, optimizing performance is crucial, as users may not be willing to wait for 12-20 seconds 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 implementation of DeA^{2}uth [78].

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, the upload

of a 1GB file was completed 60 seconds faster on Firebase. Web3.Storage, which hosts its IPFS nodes, distributes a copy of the uploaded file to each node. However, the implementation details of the service such as the replication factor, which may have an impact on upload time, were not disclosed. 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 and Infura 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 from a gateway server, as depicted in the image below. To explore the potential benefits of leveraging the IPFS network, we established our own IPFS node and measured the time required to download files. Our results demonstrate that downloading files using our IPFS node was significantly faster than using either Firebase or Web3.Storage. This is because IPFS operates in a P2P fashion and can download pieces of files from multiple nodes simultaneously. In future work, we intend to investigate the performance of other IPFS services, such as Filebase and Infura, and compare their download times to those of Firebase and Web3.Storage.

To prioritize performance in the DeA^{2}uth system, it is recommended that data requesters host their own IPFS node to facilitate the efficient download of large files. However, this recommendation is contingent on the data manager's implementation strategy, specifically regarding the storage of files on multiple nodes during the upload process.

7.2 Cost Evaluation

In DeA^{2}uth, 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 version of DeA^{2}uth, there are no costs associated with deploying contracts or submitting authorization message transactions. Even when using Firebase to simulate the blockchain, saving and reading transactions to its database incurs a negligible cost of approximately \$0.0000024[1] per transaction.

For the decentralized prototype 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, as submitting a message on Ethereum

[1] https://cloud.google.com/firestore/pricing

costs approximately $1 (as shown in Table 6.2). This means that an entire authorization sequence, consisting of RequestData, AskPermission, AllowPermission, and SendData, requires four messages, which amounts to approximately $4. This could be a potential concern for data managers who receive many requests to share files.

One possible solution to address the cost issue is to divide the costs between data requester and data managers, with the former paying for the RequestData and SendData messages, and the latter paying for the AskPermission and AllowPermission messages. However, the cost of submitting messages remains unchanged. Another approach to address this issue is to use a blockchain with lower gas fees. To explore this, we tested deploying the contract and submitting authorization messages on the Polygon blockchain, as shown in Table 7.1, where the cost of deploying a contract and submitting a message is approximately $0.03 and $0.002, respectively. This solution can potentially reduce costs significantly.

Table 7.1: Operation costs 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 7.2 displays the comparable data storage costs between Firebase and Web3.Storage, while Table 7.2 presents alternative hosting services such as Infura, Filebase, and Fleek, which are IPFS-based file hosting services. The prices offered by these hosting services are relatively similar. It is important to note that DeA^{2}uth's main objective is to enable data file sharing among users, rather than providing a data storage solution. Thus, a storage network is only necessary to facilitate the exchange of files between users. It is anticipated that data managers will manage and store files on their own database, and data requesters will download the shared files. As a result, long-term storage is not essential. In an ideal scenario, both data managers and requesters would host their IPFS node to eliminate the cost of using third-party IPFS hosting services and promote a genuine P2P model.

Table 7.2: 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

Chapter 8

Conclusions

In this dissertation, we have presented DeA^{2}uth, a novel decentralized authentication and authorization scheme for secure private data transfer. DeA^{2}uth employs blockchain technology, decentralized identity, and P2P distributed storage to enable users to have more control over their personal identifiable information and provide a platform for sharing private data with proper authorization and permission. Authentication of participants is achieved by verifying DIDs and VCs, while authorization is managed by submitting authorization messages to the blockchain. Only after receiving approval from the owner can a manager share private files with a requester.

To test DeA^{2}uth, we developed a prototype using Google Firebase as a centralized cloud storage service and then migrated to Ethereum and IPFS to make it decentralized. Our tests indicated that Firebase had better overall performance and faster times than Ethereum and IPFS when performing operations such as submitting authorization message transactions and uploading files for sharing. However, decentralization comes at a performance cost, which was an expected tradeoff. On the other hand, when downloading shared files, hosting our own IPFS node showed significant speedup in download times, demonstrating the strengths of using a P2P distributed network.

The cost of implementing DeA^{2}uth largely depends on the blockchain used to deploy the contracts. Using Ethereum to deploy contracts and submit authorization messages incurred much higher costs compared to using a cloud-based storage like Firebase. However, using a less expensive blockchain such as Polygon greatly reduced the cost of deploying contracts and submitting authorization messages.

In summary, our work has shown that DeA^{2}uth is a promising solution for secure private data transfer. Further studies are necessary to address the scalability issues of DeA^{2}uth and to test it with real-world use cases. Additionally, testing other IPFS services such as Filebase and Infura

may provide insights into alternative decentralized storage solutions.

8.1 Future Works

The present work represents an exciting contribution to the field of decentralized technology, particularly with regards to the DeA^{2}uth prototype. Nonetheless, there are several important features that still need to be addressed, particularly regarding the wallet. For instance, a more advanced production wallet would need to incorporate P2P communication, and it should be able to connect with multiple blockchains and smart contracts.

Moreover, the effectiveness of decentralized identity relies heavily on the number of participants in the network, including issuers, verifies, and subjects. Therefore, future testing of DeA^{2}uth should be conducted in existing networks, such as uPort, Soverin and Civic.

One of the most exciting aspects of DeA^{2}uth is its flexibility, particularly about the authorization message schema. Fine-grained access control can be applied by including attributes and roles into the schema. Moreover, future investigations may extend to the exploration of potential applications in diverse fields, including supply chain management, pharmaceuticals, and finance. Lastly, because DeA^{2}uth is built on blockchain and IPFS technologies, the prospects for its integration with other decentralized technologies are boundless.

Appendix A

Notations

A.1 Encryption Notations

Enc:	encryption algorithm
Dec:	decryption algorithm
sk:	secret (or private)
pk:	public key
CT:	cipher text
m:	plain-text message
m^{CT}:	encrypted message
t	message type
f:	plain text file
f^{CT}:	encrypted file
k:	symmetric encryption key
k^{CT}:	encrypted encryption k

A.2 Decentralized Identity Notations

S	subject
I	issuer
V	verifier
VC	verifiable credential
VP	verifiable presentation
$pubDID$	public decentralized identifier
$privDID_{i,j}$	private decentralized identifier between peers i and j
DR:	data requester
DM:	data manager
DO:	data owner
DID:	decentralized identifier
DDO:	decentralized identifier document object
TID:	blockchain transaction ID
CID:	IPFS Content Identifier

Appendix B

Glossary

Private Key Infrastructure	PKI
Distributed Ledger Technology	DLT
Peer-to-Peer	P2P
Blockchain-Based Access Control	BBAC
Role Based Access Control	RBAC
Attributed Based Access Control	ABAC
Private Key Generator	PKG
ID Based Encryption	IBE
Electronic Health Record	EHR
Patient Health Record	PHR
Self Sovereign Identity	SSI
Zero Knowledge Proof	ZKP
National Institute of Technology Standard	NIST
Advance Encryption Standard	AES
Rivest-Shamir-Adleman	RSA
Proof of Work	POW
Ethereum Virtual Machine	EVM
Application Programming Interface	API
Single Sign-On	SSO
JSON Web Token	JWT
Decentralized Identifier	DID
DID Document	DDO

Verifiable Credential	VC
Verifiable Presentation	VP
InterPlanetary File System	IPFS
Content Identifier	CID
Hierarchical Deterministic	HD
Data Manager	DM
JavaScript Object Notation	JSON
Data Owner	DO
Data Request	DR
Mortgage Company	MC
Credit Report Company	CRC
Hospital A	H_A
Hospital B	H_B
Proof of Stake	POS
Ether	ETH
Transactions per second	TPS

Appendix C

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 C.1: Identity.sol smart contract written in Solidity.

```solidity
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 C.2: Authorization.sol smart contract written in Solidity.

```
  ...

  /* save to Ledger */
  let txReceipt = ''
  if (process.env.SCHEME === 'DECENTRALIZED') {
    try {
      /* ganache provider */
      const web3 = new Web3(process.env.GANACHE_PROVIDER_URL)

      const contract = new web3.eth.Contract(
        AuthorizationABI,
        process.env.AUTHORIZATION_CONTRACT_ADDRESS
      )

      /* ganache account */
      txReceipt = await contract.methods
        .submit(type, cid)
        .send({ from: process.env.ETH_WALLET_ACCOUNT, gas: MAX_GAS })

      return res.status(200).json({
        data: txReceipt.transactionHash,
        message: 'submitted authorization message to Ledger'
      })
    } catch (error) {
      return res.status(500).json({
        data: null,
        error: error.message
      })
    }
    /* save to Firebase */
  } else {
    /*
      save encrypted message to Firebase
      NOTE: should not save DID in actual ledger since it could be private
   */
    try {
      const messageRef = await addDoc(collection(db, 'messages-ledger'), {
        did,
        cid,
        timestamp: new Date().toISOString(),
        type
      })

      res.status(200).json({
        data: messageRef.id,
        message: 'submitted authorization message to Firebae'
      })
    } catch (error) {
      res.status(500).json({
        data: null,
        error: error.message
      })
    }
  }
}
```

Figure C.3: Code snippet for submitting an authorization message to Ethereum or Firebase.

```
  ...

  /* lookup authorization message TID from blockchain  */
  let tx = ''
  if (process.env.SCHEME === 'DECENTRALIZED') {
    try {
      const web3 = new Web3(process.env.GANACHE_PROVIDER_URL)

      tx = await web3.eth.getTransaction(tid)

      abiDecoder.addABI(AuthorizationABI)
      const decodedData = abiDecoder.decodeMethod(tx.input)

      const type = decodedData.params[0] // messageType
      const cid = decodedData.params[1] // cid

      return res.status(200).json({
        data: { type: type.value, cid: cid.value },
        message: 'retreived authorization message from the Ledger'
      })
    } catch (error) {
      return res.status(500).json({
        data: null,
        error: error.message
      })
    }

    /* fetch DDO  from Firebase */
  } else {
    const docRef = doc(db, 'messages-ledger', tid)
    const docSnap = await getDoc(docRef)

    if (docSnap.exists()) {
      res.status(200).json({
        data: docSnap.data(),
        message: 'retreived authorization message submission from Firebase'
      })
    } else {
      res.status(500).json({
        data: null,
        error: 'failed to retreive message submission from Firebase'
      })
    }
  }
}
```

Figure C.4: Code snippet to lookup an authorization message submission from Ethereum or Firebase.

```
  ...

  /* encrypt message */
  let encMessage = ''
  try {
    await initCrypto()
    const vc = new VirgilCrypto()
    const encPubKey = vc.importPublicKey(rawPubKey)
    encMessage = vc.encrypt(JSON.stringify(message), encPubKey)
  } catch (error) {
    return res.status(500).json({
      data: null,
      message: error.message
    })
  }

  const authMsg = {
    data: encMessage.toString('base64'),
    timestamp: new Date().toISOString()
  }

  /* Save encrypted message to IPFS */
  if (process.env.SCHEME === 'DECENTRALIZED') {
    const buffer = Buffer.from(JSON.stringify(authMsg))
    const fileName = nanoid() // arbitrary file name
    const files = [new File([buffer], fileName)]
    const client = makeStorageClient()

    try {
      const cid = await client.put(files, { name: fileName })
      res.status(200).json({
        data: cid,
        message: 'encrypted message with DID public key and saved to IPFS'
      })
    } catch (error) {
      res.status(500).json({
        data: null,
        message: error.message
      })
    }

    /* save encrypted message to Firebase */
  } else {
    try {
      const messageRef = await addDoc(
        collection(db, 'authorization-messages'),
        authMsg
      )

      res.status(200).json({
        data: messageRef.id,
        message: 'encrypted message with DID public key and saved to Firebase'
      })
    } catch (error) {
      res.status(500).json({
        data: null,
        message: error.message
      })
    }
  }
}
```

Figure C.5: Code snippet to save an authorization message to IPFS or Firebase.

```
  ...

  /* retrieve authorization message from IPFS */
  if (process.env.SCHEME === 'DECENTRALIZED') {
    const client = makeStorageClient()
    const ipfsRes = await client.get(cid)

    if (!ipfsRes.ok) {
      return res.status(500).json({ data: null, error: ipfsRes.statusText })
    }

    const files = await ipfsRes.files()
    let text = ''
    for (const file of files) {
      text = await file.text()
    }

    return res.status(200).json({
      data: JSON.parse(text),
      message: 'retreived encrypted authorization message from IPFS'
    })
    /* retrieve message from Firebase */
  } else {
    const docRef = doc(db, 'authorization-messages', cid)
    const docSnap = await getDoc(docRef)

    if (docSnap.exists()) {
      res.status(200).json({
        data: docSnap.data(),
        message: 'retreived encrypted authorization message from Firebase'
      })
    } else {
      res.status(500).json({
        data: null,
        message: 'failed to retreive message from Firebase'
      })
    }
  }
}
```

Figure C.6: Code snippet to retrieve an authorization message from IPFS or Firebase.

Bibliography

[1] M. DeLisi, "Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach Nearly $600 Billion in 2023," 2022.

[2] A. Vigderman and G. Turner, "The Data Big Tech Companies Have On You," 2022.

[3] J. Isaak and M. J. Hanna, "User Data Privacy: Facebook, Cambridge Analytica, and Privacy Protection," *Computer*, vol. 51, no. 8, pp. 56–59, 2018.

[4] K. O'Flaherty, "Google+ Security Bug – What Happened, Who Was Impacted And How To Delete Your Account," 2018.

[5] Y. Zou, A. H. Mhaidli, A. Mccall, F. Schaub, 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 This paper is included in the Proceedings of the " I ' ve Got Nothing to Lose ": Consumers ' Risk Perceptions and Protective Actions after th," *Symposium on Usable Privacy and Security*, pp. 197–216, 2018.

[6] J. Rasalam and R. J. Elson, "Cybersecurity and Management's Ethical Responsibilities: the Case of Equifax and Uber," *Global Journal of Business Pedagogy*, vol. 3, no. 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 *International Conference on Passwords*, pp. 89–101, Springer, 2014.

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

[9] B. Rajendran, "Evolution of PKI Ecosystem," in *International Conference on Public Key Infrastructure and its Applications (PKIA)*, pp. 9–10, IEEE, 2017.

[10] "Decentralized Identity Systems: Architecture, Challenges, Solutions and Future Directions," *Annals of Emerging Technologies in Computing*, vol. 4, no. 5, pp. 19–40, 2020.

[11] K. C. Toth and A. Anderson-Priddy, "Self-Sovereign Digital Identity: A Paradigm Shift for Identity," *IEEE Security and Privacy*, vol. 17, no. 3, pp. 17–27, 2019.

[12] Y. Wang, Y. Ma, K. Xiang, Z. Liu, and M. Li, "A role-based access control system using attribute-based encryption," in *2018 International Conference on Big Data and Artificial Intelligence (BDAI)*, pp. 128–133, 2018.

[13] J. Adler-Milstein, A. Holmgren, P. Kralovec, and C. Worzala, "Electronic health record adoption in US hospitals: The emergence of a digital "advanced use" divide," *Journal of the American Medical Informatics Association*, vol. 24, pp. 1142–1148, 2017.

[14] S. Angraal, H. M. Krumholz, and W. L. Schulz, "Blockchain technology: Applications in health care," *Circulation: Cardiovascular Quality and Outcomes*, vol. 10, no. 9, pp. 1–3, 2017.

[15] I. Yaqoob, K. Salah, R. Jayaraman, and Y. Al-Hammadi, "Blockchain for healthcare data management: opportunities, challenges, and future recommendations," *Neural Computing and Applications*, vol. 34, no. 14, pp. 11475–11490, 2022.

[16] "Rethinking Distributed Ledger Technology," *Computer*, vol. 52, no. 2, pp. 68–72, 2019.

[17] Surjandy, Meyliana, H. L. H. Spits Warnars, and E. Abdurachman, "Blockchain Technology Open Problems and Impact to Supply Chain Management in Automotive Component Industry," *6th International Conference on Computing, Engineering, and Design, ICCED 2020*, pp. 12–15, 2020.

[18] J. Kang, "Convergence Analysis of BIM & Blockchain Technology in Construction Industry Informatization," *Proceedings - 4th International Conference on Smart Systems and Inventive Technology, ICSSIT 2022*, pp. 256–259, 2022.

[19] "The Business Process of Good Manufacturing Practice Based on Blockchain Technology in the Pharmaceutical Industry," *Proceedings - CAMP 2021: 2021 5th International Conference on Information Retrieval and Knowledge Management: Digital Technology for IR 4.0 and Beyond*, pp. 91–95, 2021.

[20] A. Shamir, "Identity-Based Cryptosystems and Signature Schemes," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 196 LNCS, pp. 47–53, Springer, 1985.

[21] A. Sahai and B. Waters, "Fuzzy Identity-Based Encryption," in *Advances in Cryptology – EUROCRYPT 2005* (R. Cramer, ed.), (Berlin, Heidelberg), pp. 457–473, Springer Berlin Heidelberg, 2005.

[22] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 89–98, 2006.

[23] 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 Communications*, vol. 19, no. 6, pp. 22–34, 2022.

[24] 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.

[25] 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.

[26] 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.

[27] 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 Transactions on Engineering Management*, pp. 1–16, 2022.

[28] A. Hasselgren, K. Kralevska, D. Gligoroski, S. A. Pedersen, and A. Faxvaag, "Blockchain in healthcare and health sciences—A scoping review," *International Journal of Medical Informatics*, vol. 134, no. May 2019, p. 104040, 2020.

[29] 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.

[30] R. Soltani, U. Nguyen, and A. An, "A New Approach to Client Onboarding using Self-Sovereign Identity and Distributed Ledger," in *International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, (Halifax, NS, Canada), pp. 1129–1136, IEEE, 2018.

[31] O. Avellaneda, A. Bachmann, A. Barbir, J. Brenan, P. Dingle, K. H. Duffy, E. Maler, D. Reed, and M. Sporny, "Decentralized Identity: Where Did It Come from and Where Is It Going?," *IEEE Communications Standards Magazine*, vol. 3, no. 4, pp. 10–13, 2019.

[32] M. Korir, S. Parkin, and P. Dunphy, "An Empirical Study of a Decentralized IdentityWallet: Usability, Security, and Perspectives on User Control," *Proceedings of the 18th Symposium on Usable Privacy and Security, SOUPS 2022*, no. Soups, pp. 195–211, 2022.

[33] "SSIBAC: Self-sovereign identity based access control," *Proceedings - 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2020*, pp. 1935–1943, 2020.

[34] N. V. Kulabukhova, "ZERO-KNOWLEDGE PROOF IN SELF-SOVEREIGN IDENTITY," pp. 381–385, 2019.

[35] "Comparative Analysis of Cryptographic Algorithms," *International Journal of DIGITAL TECHNOLOGY & ECONOMY*, vol. 1, no. 2, pp. 127–134, 2016.

[36] M. Ebrahim, S. Khan, and U. B. Khalid, "Symmetric Algorithm Survey: A Comparative Analysis," vol. 61, no. 20, pp. 12–19, 2014.

[37] "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[38] J. Per, *Integer Factorization*. PhD thesis, University of Copenhagen, 2005.

[39] J. Camenisch and V. Shoup, "Practical verifiable encryption and decryption of discrete logarithms," in *Lecture Notes in Computer Science*, vol. 2729, (Berlin, Heidelberg), pp. 126–144, Springer, 2003.

[40] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 121, pp. 120–126, 1978.

[41] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini, and Y. Khamayseh, "Comprehensive study of symmetric key and asymmetric key encryption algorithms," *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, vol. 2018-Janua, pp. 1–7, 2018.

[42] S. M. Matyas, "Digital Signatures-An Overview," vol. 3, pp. 87–94, 1979.

[43] A. Roy and S. Karforma, "A survey on digital signatures and its applications," *J. of Comp. and I.T*, vol. 3, no. 2, pp. 45–69, 2012.

[44] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." [online document], Oct. 2008. https://bitcoin.org/bitcoin.pdf.

[45] H. Liu, X. Luo, H. Liu, and X. Xia, "Merkle Tree: A Fundamental Component of Blockchains," *2021 International Conference on Electronic Information Engineering and Computer Science, EIECS 2021*, pp. 556–561, 2021.

[46] R. Beer and T. Sharma, "A quick look at Cryptocurrency Mining: Proof of Work," *Proceedings of 2nd International Conference on Innovative Practices in Technology and Management, ICIPTM 2022*, pp. 651–656, 2022.

[47] B. K. Mohanta, S. S. Panda, and D. Jena, "An Overview of Smart Contract and Use Cases in Blockchain Technology," *2018 9th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2018*, pp. 15–18, 2018.

[48] V. Buterin, "A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM," tech. rep., 2015.

[49] C. S. Ke and Y. R. Chen, "Instruction Verification of Ethereum Virtual Machine by Formal Method," *Indo - Taiwan 2nd International Conference on Computing, Analytics and Networks, Indo-Taiwan ICAN 2020 - Proceedings*, pp. 69–74, 2020.

[50] Y. Cao and L. Yang, "A survey of Identity Management technology," *Proceedings 2010 IEEE International Conference on Information Theory and Information Security, ICITIS 2010*, pp. 287–293, 2010.

[51] "The venn of identity: Options and issues in federated identity management," *IEEE Security and Privacy*, vol. 6, no. 2, pp. 16–23, 2008.

[52] T. Huang and F. M. Guo, "Research on Single Sign-on Technology for Educational Administration Information Service Platform," *2021 3rd International Conference on Computer Communication and the Internet, ICCCI 2021*, pp. 69–72, 2021.

[53] J. Jensen, "Federated identity management challenges," *Proceedings - 2012 7th International Conference on Availability, Reliability and Security, ARES 2012*, pp. 230–235, 2012.

[54] B. Alzahrani, "An Information-Centric Networking Based Registry for Decentralized Identifiers and Verifiable Credentials," *IEEE Access*, vol. 8, pp. 137198–137208, 2020.

[55] A. Tobin, "Sovrin: What Goes on the Ledger? Revision History Minor changes Replace identity "owner" with identity "holder" Executive Summary," Tech. Rep. September, 2017.

[56] W3C, "Verifiable Credentials Data Model," 2022.

[57] M. P. Bhattacharya, P. Zavarsky, and S. Butakov, "Enhancing the security and privacy of self-sovereign identities on hyperledger indy blockchain," *2020 International Symposium on Networks, Computers and Communications, ISNCC 2020*, 2020.

[58] 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, no. September, pp. 101011–101028, 2022.

[59] Y. Jing, J. Li, Y. Wang, and H. Li, "The Introduction of Digital Identity Evolution and the Industry of Decentralized Identity," *2021 3rd International Academic Exchange Conference on Science and Technology Innovation, IAECST 2021*, pp. 504–508, 2021.

[60] J. Benet, "IPFS-Content Addressed, Versioned, P2P File System (DRAFT 3)," tech. rep., jul 2014.

[61] D. Khovratovich and J. Law, "BIP32-Ed25519: Hierarchical deterministic keys over a non-linear keyspace," *Proceedings - 2nd IEEE European Symposium on Security and Privacy Workshops, EuroS and PW 2017*, pp. 27–31, 2017.

[62] P. Wuille, "Hierarchical Deterministic Wallets," 2012.

[63] M. Palatinus and P. Rusnak, "Multi-Account Hierarchy for Deterministic Wallets," 2014.

[64] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*. O'Reilly Media, Inc., 1st ed., 2014.

[65] A. A. Frozza, R. Dos Santos Mello, and F. De Souza da Costa, "An approach for schema extraction of json and extended JSON document collections," *Proceedings - 2018 IEEE 19th International Conference on Information Reuse and Integration for Data Science, IRI 2018*, pp. 356–363, 2018.

[66] M. Abdalla and D. Pointcheval, "Simple Password-Based Encrypted Key Exchange Protocols," in *Lecture Notes in Computer Science*, pp. 191–208, Berlin,: Springer, Berlin, Heidelberg, 2005.

[67] S. Sharma and K. Shah, "Exploring Security Threats on Blockchain Technology along with possible Remedies," *2022 IEEE 7th International conference for Convergence in Technology, I2CT 2022*, pp. 3–6, 2022.

[68] A. AlFaw, W. Elmedany, and M. S. Sharif, "Blockchain Vulnerabilities and Recent Security Challenges: A Review Paper," pp. 780–786, 2023.

[69] X. Yang, Y. Chen, and X. Chen, "Effective scheme against 51% attack on proof-of-work blockchain with history weighted information," *Proceedings - 2019 2nd IEEE International Conference on Blockchain, Blockchain 2019*, pp. 261–265, 2019.

[70] L. Pennella, "Proof-of-stake (PoS)," 2023.

[71] M. Aydar, S. C. Cetin, S. Ayvaz, and B. Aygun, "Private key encryption and recovery in blockchain," pp. 1–24, 2019.

[72] H. P. Singh, K. Stefanidis, and F. Kirstein, "A Private Key Recovery Scheme Using Partial Knowledge," *2021 11th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2021*, 2021.

[73] Protocol Labs, "Privacy and encryption."

[74] S. S. Ghosh, H. Parmar, P. Shah, and K. Samdani, "A Comprehensive Analysis between Popular Symmetric Encryption Algorithms," *1st International Conference on Data Science and Analytics, PuneCon 2018 - Proceedings*, 2018.

[75] P. S. Austria, *Analysis of Blockchain-Based Storage Systems.* Masters thesis, University of Nevada, Las Vegas, 2020.

[76] J. Benet and N. Greco, "Filecoin: A Decentralized Storage Network," *Protocol Labs*, pp. 1–36, 2018.

[77] 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.

[78] E. F. Coutinho, "Performance Evaluation of Data Transactions in Blockchain," *IEEE Latin America Transactions*, vol. 20, no. 3, pp. 409–416, 2022.

Curriculum Vitae

Graduate College
University of Nevada, Las Vegas

Phillipe S. Austria

phillipe.s.austria@gmail.com

Professional Experience:

- R&D Assistant Research Scientist, Oak Ridge National Laboratory, 2021-present
- Research Assistant, UNLV Computer Science Department, 2020-2023
- Research Intern, Oak Ridge National Laboratory, 2020-2021
- Graduate Assistant, Office of Research and Sponsored Projects, 2019-2020
- Photolithography Process Engineer, WaferTech L.L.C. (TSMC), 2011-2016

Publications:

- P. Austria, C. H. Park, J.-Y. Jo, Y. Kim, R. Sundaresan and K. Pham, "BBR Congestion Control Analysis with Multipath TCP (MPTCP) and Asymmetrical Latency Subflow," 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2022, pp. 1065-1069, doi: 10.1109/CCWC54503.2022.9720867.
- C. H. Park, P. Austria, Y. Kim and J. -Y. Jo, "MPTCP Performance Simulation in Multiple LEO Satellite Environment," 2022 IEEE 12th Annual Computing and Communica-

tion Workshop and Conference (CCWC), Las Vegas, NV, USA, 2022, pp. 0895-0899, doi: 10.1109/CCWC54503.2022.9720772.

- P. Austria, C. H. Park, A. Hoffman and Y. Kim, "Performance and Cost Analysis of Sia, a Blockchain-Based Storage Platform," 2021 IEEE/ACIS 6th International Conference on Big Data, Cloud Computing, and Data Science (BCD), Zhuhai, China, 2021, pp. 98-103, doi: 10.1109/BCD51206.2021.9581866.

- A. Hoffman, P. Austria, C. H. Park, and Y. Kim, "Bountychain: Toward Decentralizing a Bug Bounty Program with Blockchain and IPFS," International Journal of Networked and Distributed Computing, vol. 9, no. 2–3, pp. 86–93, 2021, doi: 10.2991/ijndc.k.210527.001.

Degrees:

Masters of Science in Computer Science, 2020
University of Nevada, Las Vegas

Bachelor of Science in Chemical Engineering, 2010
University of Washington