School of Computer Science and Statistics

A Decentralised Website Login System using ‘Decentralized Identifiers’

David Kilroy
Supervisor: Dr. Donal O’Mahony

8th August 2021

A Dissertation submitted in partial fulfilment of the requirements for the degree of Master in Computer Science
Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write.

Signed: ___________________________    Date: ___________________________
Abstract

Proof of Identity has always been an important security consideration for organisations and individuals alike. As an alternative to passwords or other forms of authentication, ‘Decentralized Identifiers’ (or DIDs) proposed by the W3C, offer a publicly verifiable and entirely user-controlled digital identity mechanism.

Using the blockchain as a distributed ledger, DIDs can be verified by any interested party while being securely owned by a single controller. Because DIDs are fully controlled by their controller and can be verified to others without revealing any personal information, they are an attractive alternative to traditional, centralised identity management.

As an exploration into a potential extension to the DID specification, I would like to investigate how they can be used to implement an alternative fully decentralised website login mechanism that can incorporated with verifiable credentials to allow for private, ethical and user-centric authentication.
Acknowledgments

I would like to thank my supervisor, Donal O’Mahony, for his continual guidance throughout the research process.

Thank you to my friends for their moral support, solid advice and for making my college experience worthwhile.

Most importantly, I would like to thank my family for sharing a workspace with me throughout this unprecedented year of my life.
# Table of Contents

1 Introduction ........................................... 8

2 State of the Art ........................................ 9
    2.1 Introduction ........................................ 9
    2.2 Identity Management Systems ...................... 9
       2.2.1 Digital Identity ................................. 9
       2.2.2 Trust Mechanisms ............................... 11
    2.3 OAuth ............................................... 14
    2.4 Blockchain ......................................... 15
       2.4.1 Overview ....................................... 15
       2.4.2 Structure ....................................... 15
       2.4.3 Access Permission .............................. 15
    2.5 Decentralized Identifiers (DIDs) ................... 17
       2.5.1 W3C Specification ............................. 17
       2.5.2 Authentication Types ......................... 18
5 Evaluation

5.1 Introduction ........................................ 46
5.2 Project Analysis .................................... 46
5.3 Benefits of Implementation ......................... 47
  5.3.1 Privacy ........................................... 47
  5.3.2 User-controlled Data ............................. 48
5.4 Future Work ....................................... 49
  5.4.1 Usability ......................................... 49
  5.4.2 Interoperability ................................. 49
  5.4.3 Business Incentive to use DID Login .......... 49
5.5 Closing Remarks .................................... 50
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>Certificate Authority</td>
</tr>
<tr>
<td>DID</td>
<td>Decentralized Identifier</td>
</tr>
<tr>
<td>DLT</td>
<td>Distributed Ledger Technology</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>IDM</td>
<td>Identity Management</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>PII</td>
<td>Personally Identifiable Information</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SSO</td>
<td>Single Sign-on</td>
</tr>
<tr>
<td>SSI</td>
<td>Self-sovereign Identity</td>
</tr>
<tr>
<td>URN</td>
<td>Uniform Resource Name</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WIF</td>
<td>Wallet Import Format</td>
</tr>
<tr>
<td>EBSI</td>
<td>European Blockchain Services Infrastructure</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This dissertation aims to investigate the suitability of using ‘Decentralized Identifiers’ (DIDs) to create a fully decentralised website login system. It seeks to improve the current state of internet login protocols by building on top of the blockchain authentication mechanism of DIDs. I will propose a detailed web-based login protocol that will allow users to log into a website using their own DIDs.

My primary aims are to use this technology to increase the privacy and security of logging into a website in order to help minimise the risk of personal data leakage from websites (even if their security has been compromised). I also aim to identity and evaluate the benefits and challenges that arise when implementing a decentralised login system.

There is a lack of comprehensive implementation analyses with this specific use of DIDs and I would like to build upon the core W3C specification to investigate this area. I will explore how a decentralised identity solution compares with existing website login systems in order to find out whether the positives of such an implementation outweigh any negatives that may arise. I will also investigate and compare the blockchains that are are best suited to building the proposed decentralised identity management system.
Chapter 2

State of the Art

2.1 Introduction

This section describes the background research done to investigate the current implementation standards of applications in the area of web-based digital identity systems, especially those that are decentralised. I will go into detail describing foundational information (such as traditional website login systems and blockchain technology) insofar as it is required to understand state-of-the-art decentralised protocols. Any research undertaken and documented in this section was done to inform design decisions described in Chapter 3: Design, or, to investigate the best possible implementation standards as described in Chapter 4: Implementation.

2.2 Identity Management Systems

2.2.1 Digital Identity

Naik and Jenkins [17] identify 3 distinct models of digital identity management on the web; Centralised, Federated and Self-sovereign.
<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy system to implement</td>
<td>Heavily reliant on user memory</td>
</tr>
<tr>
<td>Intuitive / Easy to understand</td>
<td>Poor security when passwords are reused across many services</td>
</tr>
</tbody>
</table>

Table 2.1: Centralised IDM Analysis

Centralised Identity

Centralised IDM can be described as the traditional form of identity proof which was inherited into digital IDM solutions from physical methods of IDM. A standard example of this in the digital domain is username-and-password login. The provider of an internet application will establish a trust relationship with the user based on a shared secret that is used to prove that the user is authentic.

Federated Identity

Federated IDM manages to solve some credential management issues inherent in centralised IDM. It introduces a third party to vouch for the identity of the end user. This allows the webserver to access specific information about the end user from the identity provider, when the end user successfully logs into their 3rd-party account. Federated IDM can allow for easier password management on the side of the end user (by reducing the number of passwords they use to log into serveral different websites). It is also the basis for creating networks with SSO functionality. A typical example of this type of IDM is when a user signs into a website (www.examplewebsite.com) using the login details of their Facebook or Google account. In this case, Facebook or Google are the trusted third party that verifies the identity of the user to allow them to sign in to www.examplewebsite.com.
### Table 2.2: Federated IDM Analysis

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users can avoid the registration step for many websites</td>
<td>Identity provider becomes a single point of failure.</td>
</tr>
<tr>
<td>Reduces overall number of passwords a user has to remember</td>
<td>Requires a high level of trust in the 3rd-party</td>
</tr>
</tbody>
</table>

**Self-sovereign Identity**

Self-sovereign IDM is the latest development of digital identity management and is the method that I will primarily focus on throughout this dissertation. Self-sovereign IDM improves upon both centralised and federated forms by removing the need for cumbersome password management while also not relying on trust in a centralised third party for identity verification. Self-sovereign identities can be cryptography verified by some decentralised mechanism (most commonly by writing to a blockchain or other DLT) to allow website verifiers to trust a user’s identity claim. SSI is an interesting paradigm in digital IDM as it allows much more privacy control than was possible in previous forms of identity verification. SSI-based ‘Decentralized Identifiers’ (DIDs) allow for zero-knowledge proof of identity whereby users can prove control of DIDs without revealing any PII and without requiring permission from a centralised 3rd-party entity. The DID can be used as a verifiable identifier of any entity seeking to log into the website (whether that is a person, device, organisation etc.)

#### 2.2.2 Trust Mechanisms

In any digital identity management system, the protocol used to verify users is only as useful as the trust mechanism behind it. This trust mechanism will vary entirely depending on the nature of the application and the actors involved. In an identity management system, ensuring a trust relationship between parties and the protocols being used is an extremely important factor which always needs to be considered, no matter the form of IDM
In a Centralised website login system, the trust relationships required to authenticate users are very simple. The login mechanism requires that the end user (or client) trusts the website to authenticate them once they provide their registration details. The client may also like to make sure the server is definitely who they claim to be which is generally achieved by sending messages over HTTPS with X509 certificates. Beyond this, no other pre-existing trust is required to authenticate users. Website servers do not need to trust that a user’s login details are correct as they can verify this firsthand with the user data that they possess (usually a username and password).

Figure 2.1: Trust relationships in Centralised IDM

The trust mechanism of Federated login systems becomes more complex as a third party is introduced into every login request. In addition to client-server trust, servers must be willing to trust that the external identity-verifying party will provide them with accurate client data. As well as this, before the client can initiate a login request with a 3rd-party account, the 3rd-party verifier and server must trust one another as legitimate. Trust between the server and 3rd-party is usually facilitated by the 3rd-party’s reputation. For this reason, smaller websites will tend to trust accounts from well-known sites like Google, Facebook, Github etc. Trust of the server by the 3rd-party is often established by requesting their own API token which allows the 3rd-party to authenticate requests made by the server and to keep track of their previous interactions.

If the login system uses OAuth bearer tokens, the token will allow website servers to obtain data about the client once these trust relationships exist.
The trust relationships required for self-sovereign identity management systems is quite similar to Federated IDM with one major exception. Instead of trusting a well-known 3rd-party, website servers must trust the cryptographic verification mechanism of a specific blockchain. Due to identifier control being shifted to users, servers should be able to trust that the client is authentic. [6]
2.3 OAuth

OAuth is an token-based authorisation framework [8] that enables 3rd-parties to provide client information to authenticated website server that request it. It is a widely used protocol that enables the trust mechanism of Federated IDM as mentioned previously. In an internet context, this allows websites to request an authorisation code from a trusted authority that vouches for the identity of the user who would like to login to that website. Once authorised, the website server will be presented with a bearer token that grants them access to specific information that can be to identify the end user (In the case of Google sign-in, this is: user ID, name, profile picture URL, email address [10]).

In the case of HTTP protocols, OAuth sends the bearer token to the requesting website in the "Authorization Request Header Field" using the bearer authentication scheme. [12] To make this process secure the RFC clarifies that this must be done with HTTP over TLS (i.e. HTTPS) to make sure tokens are not sent between the servers and clients in unencrypted plaintext.

As detailed in [4], Dodanduwa and Kaluthanthri highlight that a pre-existing trust relationship is required between the resource owner (i.e. the website’s end user with a 3rd-party account) and client (i.e. the website itself) prior to relying on OAuth as a authorisation mechanism. Establishing a trust relationship outside of protocols themselves is an important factor in many digital identity verification systems, not just OAuth. Any mechanism for verifying the identity of an entity is only as useful as verifiers are willing to trust the parties involved.
2.4 Blockchain

2.4.1 Overview

Interest in blockchain technology has skyrocketed in the last decade due to the increased popularity and profitability of cryptocurrencies such as Bitcoin and Etherium. Blockchain technology is the basis for decentralised authentication for these cryptocurrencies and there is ongoing research into how blockchains could be used to verify identity without the need for centralised involvement.

2.4.2 Structure

Blockchain is so named as it consists of a series of records called blocks which build upon data in preceding ones. Each block contains a cryptographic hash of the previous blocks in the chain, as well other relevant transaction data specific to the blockchain in question. Critically, transactions recorded on the blockchain are immutable. They cannot be modified retroactively because each block that is written to the chain is created using a one-way cryptographic hash function (for example, Bitcoin uses SHA-256 as its proof-of-work algorithm). This chain forms the basis of a decentralised ledger where the entire history of all blocks are recorded and can be viewed by anyone with a copy of the chain. Blockchains are considered to be decentralised, peer-to-peer networks where new transactions are checked according to a set of agreed rules and propagated throughout peer nodes to verify them as legitimate. Proof-of-work (PoW) and proof-of-stake (PoS) are two common blockchain consensus rules. [7]

2.4.3 Access Permission

Blockchains can be set up with different permission access depending on their use case. Read and write access to a blockchain tends to be regulated in one of three types: public-permissionless, private or hybrid.
**Public-permissionless** blockchains are ones that any individual may access. They lack any additional permission requirements to read or write to the blockchain as opposed to Permissioned blockchains. Public-permissionless blockchains are truly decentralised as they are not managed by a central organisation. As a result, most cryptocurrency blockchains are considered to be public-permissionless blockchains (Bitcoin, Etherium etc.)

**Private** blockchains are created for private use within a specific organisation or group or organisations. Private blockchains tend to restrict the decentralised nature of blockchains (by adding central management controls) in order to reduce the scope in which they are used. Hyperledger Fabric and Multichain are examples of private blockchains aimed at providing enterprise-level DLT solutions. Private blockchains are also permissioned by definition.

Recently, some blockchains have taken a hybrid approach to access management. Hyperledger Indy is a self-proclaimed ‘public-permissioned' blockchain which means anyone can send transactions across the network however only certain nodes can vote to verify those transactions. [9] This management of the blockchain network by a small number of permissioned nodes allows an organisation to regulate transactions written to the blockchain. Built on an instance of the Hyperledger Indy, a decentralised identity application called Sovrin uses permission controls to ensure that transactions to their blockchain are formatted correctly and that they are used for identity verification purposes only. [24]

<table>
<thead>
<tr>
<th>Private</th>
<th>Public-Permissionless</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperledger Fabric</td>
<td>Bitcoin</td>
<td>Hyperledger Indy</td>
</tr>
<tr>
<td>Corda</td>
<td>Etherium</td>
<td>Alastria</td>
</tr>
<tr>
<td>Multichain</td>
<td>XRP Ledger</td>
<td>EBSI</td>
</tr>
</tbody>
</table>

Table 2.3: Blockchain Permission Types
2.5 Decentralized Identifiers (DIDs)

2.5.1 W3C Specification

Decentralized Identifiers (or DIDs) are a form of decentralised, self-sovereign identifier proposed by the W3C. A DID consists of a string that can be used to identify any type of entity (such as a person, device, organisation etc.). The entity a DID refers to is known as the ‘DID subject’.

Each DID also has a ‘DID Controller’ who has created the DID and decides which entity the DID identifies. The subject of a DID may be the DID Controller themselves. A DID Document is an object containing metadata about its associated DID. It links the DID subject to its controller as well providing information about interacting with the DID subject, such as a URL that may be queried to reference the DID in question. The W3C specification necessitates that DIDs satisfy the following properties:

1. Permanent Identifiers
   They cannot be revoked or reassigned to identify a different entity other that the initial DID subject. This is distinct from identifiers such as DNS addresses where ownership of the address can be revoked and reassigned to different entities.

2. Resolvable
   A DID can be used to lookup metadata about the DID subject. This is enabled by the existence of the DID document.

3. Cryptographically Verifiable
   Control of a DID can be proven using cryptography. This is the primary attribute which allows DIDs be truly decentralised.

4. Decentralised
   No one central authority is required to verify entities identified by DIDs.
DIDs are formatted as URI strings containing their scheme, DID Method and specific identifying string.

![Example DID string format](image)

**Figure 2.4: Example DID string format**

**DID Methods**

To ensure DID implementations across different blockchains are in accordance with the W3C standards, a DID Method specification must be written for each blockchain. The DID Method specifies the CRUD (Create, Read, Update and Deactivate) operations for DIDs registered on that specific blockchain, as well as other relevant human-readable registration instructions.

### 2.5.2 Authentication Types

**Ledger-based**

The most common DID authentication mechanism is to write data to a blockchain (or similar form of distributed ledger) which allows for decentralised, verifiable identification of the DID subject. Ledger-based DID implementations usually fall into two categories:

1. The DID is created using information from a blockchain address. In the case of Bitcoin DIDs, the DID string is derived from a specific Bitcoin transaction hash between two BTC addresses. [30] Once the
transaction has been signed and registered on the blockchain, the
transaction hash can be used to create a DID.

2. The DID is created using an asymmetric key pair and is registered to
the blockchain using the private key.

In the case of the Ontology blockchain, writing to the ledger falls into the
second category. DIDs are digitally signed by a symmetric key pair and
registered on-chain using its public keys and metadata. This allows full
control of the DID to remain with the DID controller as they are the owner
of that DID’s private key.

Layer 2

While >90% of current DID Methods use ledger-based DID authentication
[32], some DID implementations have been developed using other forms
of decentralised authentication.
Layer 2 DID implementations utilise blockchains (or other DLTs) without
registering information onto the chain directly.
The most notable of these is the Microsoft’s ION DID Network. [15] In
early 2019, Microsoft began development of the Layer 2 DID Network
which is looking to leverage the trust mechanism of blockchain ledgers
while avoiding some of the transaction complexity inherent in transferring
cryptocurrencies. ION is described as an overlay built on top of the Bit-
coin blockchain. It also aims to increase scalability of DID registration by
verifying thousands of DIDs using a single Bitcoin transaction.

Peer-to-peer

Another alternative to ledger-based authentication are peer-to-peer (P2P)
DID networks. These networks implement DID authentication between
peers without the use of a distributed ledger. They are well suited to pri-
vately resolvable group or one-to-one DID networks (referred to as N-wise
and Pairwise DID Networks respectively)
As they don’t depend on any form of publicly accessible ledger, they are a
much cheaper to implement and a more scalable type of DID authentica-
tion network, however, they lack the global resolvability that is inherent in
blockchain-based DID networks. As a result, in a P2P DID network, peers
must also develop their own trust-relationship infrastructure to be able to
non-trivially use DIDs.

2.6 Verifiable Credentials (VCs)

2.6.1 Verifiable Claims

The W3C defines a verifiable claim as a cryptographically verifiable ‘... piece of information about an entity’s background’. [29] Similar to DIDs,
they refer to a specific subject and can be publicly verified using cryptogra-
phy by entities looking to verify them. Contrary to DIDs, Verifiable Claims
should be signed by a trusted ‘Claim Issuer’ who verifies the content of the
claim. For example, a university may be a claim issuer for ‘student status’
claims or, alternatively, a government body may be a claim issuer for ‘le-
gal name’ claims. Verifiable Claims should be combined with an external
identifier system (whether that is DIDs or another mechanism) to create a
non-trivial implementation that has a reliable trust anchor. [28]

Structure

Verifiable Claims must contain values under the following attribute head-
ings to be considered valid.[26] As claims are transferable to different iden-
tifiers (that are owned by the same entity), these attributes are separated
into ‘Identity Profile’ data and ‘Entity Credential’ data.

• Identity Profile

Contains an identifier URI representing the subject, the type of the
identifier and a signature that verifies the identifier.
• **Entity Credential**

  Consists of an identifier URI representing the claim, the type of the identifier, the actual claim content, an identifier URI representing the issuer, and the date the claim was issued.

  The Entity Credential data also contains a signature that verifies the issuer's validity. A revocation mechanism reference should also be placed in this section of the document.

Fig 2.5 illustrates a sample verifiable claim in JSON format, the example assumes use of DIDs as identifiers.

```json
{
    "@context": "https://w3id.org/security/v1",
    "id": "https://example.uni/credential/1234",
    "type": ["Credential", "StudentClaim"],
    "issuer": "https://tcd.example.uni",
    "issued": "2021-01-01",
    "claim": {
        "id": "did:example:1234",
        "studentStatus": true
    },
    "signature": {
        "type": "RSASignature",
        "created": "2021-01-01",
        "creator": "https://tcd.example.uni/keys/#1",
        "nonce": "000001",
        "signatureValue": "bd932608ccdebe8c0d1ebe1645be24715cd2276f8420ecca2e179e5cca2bc82"
    },
    "revocation": {
        "id": "https://tcd.example.uni/revocations",
        "type": "SimpleRevocationList"
    }
}
```

**Figure 2.5: Verifiable Claim Template**
2.6.2 Claims vs. Credentials

In [27], Verifiable Credentials (or VCs) are defined as sets of claims about the same subject that build upon one another to form a more complete picture of the subject’s identity. VCs are distinct from other forms of official credentials for two reasons. Firstly they are can be verified cryptographically, like all mechanisms defined previously. And secondly, credential holders can selectively reveal aspects of their identity to verifiers depending on the circumstance. This usually means credential holders can reduce the amount of information they are required to share in order to prove claims about themselves. For example to prove ‘student status’ (assuming the person has a valid claim from a trusted university) a verifiable claim can allow an inquirer to verify this using only a signed claim to the subject’s student status. When compared to using a student card for verification, as is done traditionally, this allows individuals to prove asserted claims without also revealing their name, age and profile picture.

In this way VCs can afford more privacy control to individuals that are being authenticated.
2.7 Decentralised Login Systems

2.7.1 DAuth

Development of decentralised authentication systems is not a novel concept. DAuth is a web-based authentication system that is built on the NameID authentication protocol instead of DIDs. [19] Similarly to DID-based implementations, DAuth uses a blockchain (in this case Etherium) to cryptographically verify and authorise user IDs. The paper describes well the process of registering an ID to the blockchain by use of smart contract transactions. As the OpenID implementation of DAuth requires a 3rd-party to verify identity, the proposed architecture uses the Etherium blockchain to fill this role. As a result, the parties involved in the protocol are similar to other 3rd-party/Federated identity authentication mechanisms. These are:

- Website Server (the application which provides website content and allows users the option to login using DAuth)
- Client (the user who provides a form of decentralised identifier to be logged into the website)
- 3rd-party identity verifier (in this case, the Etherium blockchain)

The steps required to create a web-based decentralised authentication protocol are also detailed in this paper. Firstly, an identifier must be created and written to the blockchain to prove uniqueness and allow cryptographic verification. The process of web server authentication an given identifier consists of the following steps:

1. Details of a given identifier are requested from the blockchain by the website server.
2. The website server asks the client to sign a generated message using the private key associated with the given identifier.
3. The client returns the signed message to the website server.
4. The signature provided by the client is compared to the verifying signature on the blockchain. If they match, the user is authenticated and logged into the website.

My proposed design follows a protocol very similar to this with two key differences. Firstly, the identifiers used are W3C DIDs, not NameID/OpenID ones. Secondly and more importantly, the final step in the verification process is reworked to be usable in a real-world application.

There are major issues with using a straight signature comparison to verify that users are who they claim to be. As identifier details are publicly available to anyone who requests it, any party can obtain the signature linked to a particular identifier. This means exact the answer that the website server expects to its challenge mechanism is public knowledge. The login protocol proposed by DAuth can then be easily broken by providing the expected signature back to itself directly from the blockchain. This enables any individual to login to with any identifier whether or not they are the controller of that identifier.

In my proposed design, the website server compares the client’s signature with the *public key* obtained from the blockchain, not the DID document’s publicly available signature. This ensures that only the possessor of a DID’s private key can create valid signatures to verify that DID, while allowing any other party to check these signatures as valid.

### 2.7.2 Other systems

There are also a number of protocols to enable blockchain-based authentication specifically for IoT devices. [33] [16]

Zhaofeng et al. propose a blockchain-based authentication scheme for IoT networks called BlockAuth. [33] The system deploys a local instance of the Hyperledger Fabric blockchain to facilitate peer-to-peer authorisation between devices. The authentication mechanism also relies on password-based certificate generation for each device within the network. Once certificates are written to the local blockchain in the form of a transaction, sig-
natures of these transactions can be requested and verified before communicating from one IoT device to another. In this case, a node in the system is assigned as the CA administrator to handle registration of device certificates to the blockchain. This removes the need for a public well-known CA and public key infrastructure to vouch for devices in the network, which is suitable for IoT network authentication.

The BlockAuth protocol proposes some very interesting concepts, some of which are transferable to a DID-based environment. The signature verification mechanism (which uses ECDSA encryption) is very similar to the one I use in my web application. Signatures are checked as valid when compared to the expected value of decryption using a device’s public key. Other concepts in this system, such as setting up a local blockchain, work very well in an IoT network as they do not require an existing PKI but wouldn’t be suitable to authenticate DID owners in a web application.
Chapter 3

Design

3.1 Overview

The aim of my project is to test whether DIDs can be used as an alternative to federated account sign-in to reduce the negative privacy impact of websites that store large amount of user data. Using DIDs for this purpose also significantly decreases the risk of user identity exposure in the case of a website data breach, when compared to other forms of user authentication. Whether the breach is accidental or a malicious attack, the user can be assured that their identity is not at risk because the DID resolution and authentication mechanisms do not necessitate storage of PII for use in identity verification.

In order to investigate this, the system I propose uses includes a web application to resolve and authenticate a client’s DID as an alternative to traditional login techniques. Once authenticated with their DID, the user can then avail of the benefits that the website provides to logged-in users without increasing their risk of personal data exposure.

In the following section I will describe the motivation, design decisions and implementation of my DID login application.
3.2 Choosing the Blockchain

In order to create a testable, verifiable DID, it needs to be written to a blockchain. The main two criteria for deciding which blockchain to write to were:

1. Transparency of access permission
2. Cost per transaction

Due to the nature of my proposed application, I required the DID to be publicly verifiable by any interested web service provider. This limited the set of usable options to public-permissionless blockchains.

From this set of DLTs, I define two distinct sub-categories of blockchains based on their intended use case: cryptocurrency blockchains and SSI blockchains.

The issue of using cryptocurrency blockchains (such as Bitcoin or Etherium) occurs when using them to verify large numbers of DIDs. When attesting DIDs using a blockchain, the monetary cost of writing transactions to the chain are of no interest. For example, according to the Bitcoin DID Method, block transactions are required to create and verify identifiers. These transactions will become increasing expensive if the value of the coin increases over time. This issue reduces the scalability of maintaining a large number of DIDs on the most popular blockchains.

For this very reason, I will discuss some blockchains that have been developed to be used specifically for verifying decentralised identities.
3.2.1 SSI Blockchains

Of the blockchains with defined DID Methods, there are a number aimed specifically at providing a framework for SSI verification.

Even despite this, these blockchains that are aimed at SSI verification still have a glaring issue. It is possible for SSI blockchain tokens to increase in price (due to increased popularity) to the level that it is not sustainable to use to write new DIDs to the chain. Some blockchains aimed at providing SSI have measures to reduce the effect of this, such as Ontology’s dual token architecture. Note that any DLT with a valid DID Method can be used to attest DID validity. The following blockchain technologies are simply marketed as having their primary use be self-sovereign identity management.

For the purposes of testing my prototype, I considered only the ones with comprehensive documentation. This left the following blockchains to choose from: IoTeX, Metadium, OmniOne, Ontology, Veres One and Vivvo.

IoTeX

As the name might suggest, the IoTeX blockchain is primarily aimed at storing decentralised IoT data. This is an interesting use case for DIDs as most existing DID Methods are aimed at identifying individuals. This blockchain seeks to use DIDs to identify specific objects or computers instead of people. Due to the fact that my project would be aimed at authenticating users and because of a lack of documentation compared to other candidates, I chose not to use this blockchain.

Metadium

Metadium’s blockchain is primarily used internally by the company’s own identity management mobile application which offers a digital wallet to manage DIDs. (The identity app is comparable to Sovrin or uPort) When considering Metadium as an option, I found that is had a separate
TestNet blockchain to allow developers to test decentralised apps (DApps) and smart contract scripts. After seeing this feature, is was something I looked out for in subsequent blockchains I investigated.

**OmniOne**

The OmniOne blockchain is unique in that it combines DLT with biometric FIDO standards to create DIDs verified by the user's biometric data. FIDO protocols allow for secure storage of biometric-linked public and private keys on mobile devices. [1] With the FIDO2 protocol, this is done by locally storing encrypted private key info on dedicated hardware located on the user's mobile device while uploading their public key to an online service. While this is an interesting area with potential for further research, OmniOne’s focus on biometric authentication means it is not ideal for testing my project implementation which would use some form of in-browser web-based user authentication.

**Ontology**

Ontology provides a set of blockchains and DLTs for the primary purposes of identity management. Similar to Metadium, Ontology has a MainNet and TestNet to allow for developers to test smart contract code before writing to the MainNet. Ontology has a lot of comprehensive documentation as well as SDKs to interact with their blockchains in several programming languages. Ontology, in particular, attempts to limit huge token value increase by splitting its tokens between ONT (acting as a staking crypto token) and ONG (acting as a "gas price" token for verifying on-chain transactions).

**Veres One**

Veres One is a blockchain that uses HTTP-based endpoints to allow users to register and verify DIDs. On initial inspection it seemed this might integrate well with my RESTful application, however there were major issues
that would prevent this. As it stands, their DID creation documentation is minimal with command-line operations being the only method of writing a DID to the blockchain. While it seemed to be a promising blockchain candidate, the lack of proper documentation made this unsuitable for use in creating the implementation I was planning.

**Vivvo**

While Vivvo is a distributed ledger specifically for self-sovereign identity, it is permissioned and intended to be used privately within an organisation. As a result, this would not be suitable for my implementation as it requires a DID mechanism that is publicly verifiable to allow website owners to authenticate clients.

**Conclusion**

From the above blockchains, I chose to use the Ontology blockchain to implement my project. It contained the most complete documentation about blockchain specifics out of the blockchains that weren’t aimed at niche use-cases. As well as this, it seemed the presence of a dedicated TestNet would be beneficial in testing my decentralised login application.
3.3 Writing to the Blockchain

3.3.1 Ontology SDK

Introduction

The Ontology DID protocol (also known as ONT ID) allows for DID documents to be linked to with a public key and stored on Ontology blockchains as per the defined DID Method. [5] The framework allows for a fully self-sovereign and privacy-preserving implementation of DIDs to be developed and used in applications that require a decentralised identity management system. The following section describes interactions with the Ontology blockchain that are required to create, access and modify Ontology DIDs.

Blockchain Transactions

Interacting with the Ontology blockchain from a local machine involve creating ONT ID Description Objects (DDO) and writing them to the blockchain in the form of transactions. This is usually done by the creation of a smart contract.

DID Creation

According to ONT ID documentation[18], Ontology DID URI strings are must adhere to the following format to be registered to the Ontology blockchain.

- A 20-byte random number is concatenated with a 1-byte version tag (s.t. data = VER || rand[20])
- 4-bytes of parity bits are generated by SHA256 hashing the above data twice over and taking the first 4 bytes.
• The results are Base58 encoded and concatenated to the end of "did:ont" to create a DID string in the following format: ONT DID = "did:ont" || Base58(data || parity)

Once created, DIDs must be registered to the blockchain before they can be resolved. They are registered with a set of asymmetric keys that the user has control of. To do this, a JSON document that contains data describing the DID, known as a DID document, must be created. At a minimum, the DID document contains a public key owned by the user as well as some metadata associated with the key and ONT ID string. ‘@context’, ‘id’, ‘publicKey’, ‘authentication’ and ‘created’ are the required tags needed for a valid Ontology DID document however other properties such as ‘controller’ and ‘recovery’ DIDs and Keys can be specified in the document to allow for key recovery.

‘@context’

This property must contain a reference to the W3C DID specification webpage as well as a second reference to the Ontology ONT ID specification webpage. This property is the informs the reader that the entity referenced in the document is a W3C DID and furthermore that is written to the Ontology blockchain.

‘id’

The id property specifies the ONT ID string that the document relates to. This string is compared with the query string when performing the DID Resolve operation.

‘publicKey’

The publicKey property is an array of all keys linked to the DID. Each key in the array contains id, type, controller and publicKeyBaseHex elements. The
ID uniquely identifies the key within the array. The ‘type’ element specifies the key generation algorithm used to create the public key. The ‘controller’ element gives the ONT ID string of owner of the public key. This may or may not be DID subject of the document in question. In the case that the ‘controller’ tag doesn’t match the DID of the current document, the DID Subject may be an entity other than the DID Controller. The ‘publicKey-BaseHex’ element contains the hex-encoded publicKey data to be used for DID authentication.

**DID Update**

As per DID Method requirements, DID documents can be updated after they have been created. Documents can only be updated by the DID Controller and any other entities who are listed as "recovery persons" in the DID document. Entities with private keys of these listed DIDs can modify the DID document to add and remove public keys from the authentication list. In this way, recovery entities can help the DID controller and other recovery persons to recover DID ownership in case of key loss. The recovery person must provide valid digital signatures, proving control of their own DID, before updating private key settings on a document.

As specified in the Ontology blockchain DID Method, public authentication keys can be added or removed by users who control keys already contained within the DID document.

**DID Deactivate**

As Ontology-based DID authentication involves proving control of a public key, it is sufficient to delete all public keys from the DID document in order to deactivate it. The DID document of a DID containing no public keys has no way of providing proof of identity in addition to a lack of a mechanism to add any new public key/DIDs to the authentication list. This renders the DID as inactive with no way to reactivate it which fulfils the W3C specification for DID deactivation.
Chapter 4

Implementation

4.1 Introduction

The following section describes the application I have developed to show how DIDs can be used as the basis for a decentralised website login mechanism. Within this application, I define a specific DID-based challenge/response protocol using the Ontology blockchain. To demonstrate the flow of this protocol, the project consists primarily of a website that allows users to login to it using a DID that they can prove ownership of. As well as this, the project includes a command line tool to facilitate digital signing of a message using a DID’s private key and another tool which extends the login functionality to validate ‘Verifiable Claims’.

4.2 Architecture

The application consists of the following components: a Node.js web server, Ontology TypeScript SDK, Ontology Testnet blockchain, digital signing tool (See Fig 4.1)
The **web server** provides RESTful routing to serve and render specific web-pages to the client. It is also responsible for working out the logic for all the DID Method operations that are required to provide user authorisation (namely ‘DID Read’). The web server uses the Ontology SDK to directly interact with the Ontology blockchain through the native smart contract.

The **digital signing tool** is a Powershell command line tool that is isolated from the web application. It allows the user to create digital signatures using a WIF-formatted private key. Once the user inputs the message to be signed, the tool will create a digital signature using the Node.js core crypto library.
The SHA-256 algorithm is used to create an initial message digest using Node's core pseudo-random number generator. This message is then signed with the private key to create a final signature which is sent as output to the terminal.

### 4.2.1 DID Login

Authenticating users by the DID Login protocol is a two-stage process that makes sure DIDs belong to the client. Firstly, DIDs must resolve to a DID document which verifies that the DID exists and the blockchain attests its validity. Secondly, the client must prove ownership of the private key associated with the DID to verify that the DID refers to them and not some other entity.

**DID Resolution**

In order to check that a DID is registered to the blockchain (i.e. check that it is a valid DID), the back-end SDK queries the ONT ID Contract API to attempt to resolve a DID document from the given DID URI. As specified in the Ontology DID Method [5], this is done using:

```go
func GetDDO(ontId string)
```

The function queries the blockchain to fetch the DID document for the given DID. If no document is returned, the DID has not been registered and is not a valid identifier. If a document with valid metadata (i.e. at least a public key, controller and subject) is returned, the DID’s existence is attested by the blockchain. This stage works as validation due to the uniqueness property that is required of all DIDs.
DID Authentication

After the DID is resolved and proven to be written on the corresponding blockchain, it must be verified that the DID belongs to the user. Authoritative protocols for verifying DID ownership are currently not well defined and differing blockchain providers allow users to authenticate DIDs through differing mechanisms. The message sending protocol I have defined for DID authentication in this project is an adapted version of an early proposal called ‘DID Auth’.

‘DID Auth’ is an informally-defined set of protocols that aim to tackle the issue of proving DID control. In a 2018 draft paper co-written by Markus Sabadello (a co-founder of the W3C DID Working Group), many potential methods of verifying DID ownership are put forward. [23]

The paper outlines how a challenge/response protocol could be used to verify DIDs that is similar to other forms of 3rd-party federated authentication. As it is not yet a standard, there are different implementation ideas for the exact data transfer protocols when verifying DID ownership. [22]

I have adapted one of the proposed solutions to define a message-sending authorisation protocol that works specifically for a web application using HTTPS. (See Fig [4.2]) This is done by issuing pseudorandom challenge hashes which must be signed by the private key related to the DID in question.

HTTPS

In order to encrypt all client-server communication across the endpoints of my application, I set up the server to listen for client connections using HTTPS. To do this, the web application includes a self-signed certificate. Use of a self-signed certificate in a live production environment would significantly reduce the security of the website, so it solely exists as a proof of concept to show that DID resolution can be achieved more securely using HTTPS.
Figure 4.2: DID Login Protocol Flow
Front-End

To create a simple front end for my application, I used Pug as the view template engine to generate working HTML and JS. Views were then styled primarily using the Bootstrap.js framework. The login protocol flow is mirrored by the front-end to demonstrate how the mechanism would work in a real-world website.

Each view is a distinct HTML page that is mapped to a corresponding back-end Express.js route that is responsible for rendering the correct elements on screen and handling page navigation logic.

4.2.2 Verifiable Claims

The Verifiable Claims mechanism of my application demonstrates how DID-based login can be expanded upon to provide additional value to users. Specifically, claims are used to allows users to prove that they are students. In the case of a e-commerce application, this would allow store owners to provide student discounts to verified users (i.e. who have a cryptographically-signed claim from a well-known university).

Create Claim

The create_claim.js tool instantiates a Claim object and runs the code required to allow the blockchain attest it. Once written, the claim is written publicly to the claimant's DID document to allow other entities to verify it. To expand upon the functionality of this application, a private user-controlled claim presentation method could be developed to ensure that information associated with claims is only viewable by trusted verifiers. One such method is proposed in [13]. In this case, the claim will be encrypted and the explicit permission of the subject is required to decrypt it. Secondary parties can present claims to verifiers upon request, however they need to be decrypted by the subject before they can be deemed valid.
Once completed the `create_claim.js` tool will return either a SUCCESS response or a relevant error.

**Verify Claim**

Similarly to verifying DID control, Verifiable Claims made about an entity must be approved as cryptographically valid before they can be trusted. Proving validity of a claim consists of 3 distinct steps:

- Verifying that the claim has been attested to a blockchain.
- Verifying that the claim has been signed by a trusted entity. (For a student claim, this would be a well-known university.)
- Checking that the signature of the claim is in date and hasn't been revoked.

In the case of the Ontology DID Method, claims can be written to a user’s DID document for ease of access. This process of storing claims directly on a DID document has privacy issues as personal information that is stored is publicly accessible. Ideally, the process of verifying a claim would involve a verifiable presentation whereby the claim subject can share only the parts of the credential that are necessary to prove an aspect of their identity.

As suggested in the Verifiable Credentials Specification [27], a claim presentation mechanism could involve wrapping a claim in a signed ‘Verifiable Presentation’, privately sharing this with a verifying party, and using the blockchain DID document only to verify the signature of the presentation. This would remove the need for personal information to be stored on a public DID document, while still allowing verifiers to cryptographically verify claims.
Revoke Claim

According to the W3C spec for VCs, issuers must be able to revoke claims they write. When a claim is being issued to an entity, a revocation mechanism must be specified. This may be a simple list of claims and their status as to whether they are still valid or not, or another mechanism entirely. As long as the process allows for any claim issuer to revoke a claim they have made, it is valid to include it as a revocation mechanism.

In the proposed project implementation, revocation is done by checking the 'AttestContract' status of a claim. If a claim is still attested by the blockchain it is considered valid, the assumption being that if an issuer would like to revoke a claim they can send a transaction that revokes the blockchain attestation. In this case, data about revoked claims will not be obtainable from the blockchain and it may be considered either revoked or non-existent by a verifier.
4.3 Security Considerations

In this section I will put forward the security and privacy risks that I have identified with my system’s design as well as the steps taken to mitigate their likelihood and severity. Within my application, the main attack surfaces that I have identified are grouped in the following areas:

- Attacks against Key/Signature Mechanism
- Attacks against bad HTTPS Header use

4.3.1 Key/Signature Mechanism

In the proposed implementation, users are authenticated using a signature-based challenge response mechanism. To verify the assertion that the given DID is controlled by the client, the server issues them a challenge.

The challenge issued is a pseudo-randomly generated 32-byte SHA-256 hash that is uniquely created for each authentication request sent to the backend server. The server expects to receive a signature that has been created with the private key associated with that DID. As the private key is something only the DID controller knows, this signature proves the client’s ownership of the DID and the server can authenticate the user as legitimate.

As the motivation for using DIDs is to enable self-sovereign identity verification, the responsibility for data security is shifted from the server’s password database to the user. This comes with its own challenges however as it’s very possible that regular users need to have some knowledge of data security to store their private keys properly.

Some of the main security concerns for this mechanism are discussed below:
Signature Forgery

The cryptographic strength of a signature is as strong as the specific algorithm used to create the keys used for signing. In this case, the keys used for signing are the ones used to register a DID on the Ontology blockchain. According to the Ontology DID Method, the suggested cryptographic algorithm for DIDs is ECDSA. The current default curve to achieve this is NIST P-256 (a.k.a secp256r1 or prime256v1 by SECG and ANSI, respectively). In the case that another party is able to guess the signature or private key related to any given DID, then that DID is cryptographically insecure and another encryption algorithm should be used instead.

Because of this, it is in the interest of blockchain developers to support EC crypto verification on a well known curve that is considered to be secure. As NIST P-256 is commonly used in EC cryptography, there seems to be few reasons to doubt its security (as per current computing standards at the time of writing this dissertation, it is very possible this might change in the future).\[2\]

Man-in-the-middle Attack

In order to minimise the risk of a Man-in-the-middle attack, the backend server sends and receives messages using HTTPS. Once the TLS connection has taken place, communications between client and server are fully encrypted. For the sake of demonstration, the project implementation contains a self-signed certificate and runs on localhost. In a real world implementation, a website using the DID login mechanism would require a SSL certificate from a well-trusted CA to reduce the risk of a MiTM attack.

In Nodejs versions $>12.x$, use of TLSv1.3 is enabled by default when running a HTTPS server, meaning the frontend client can specify TLSv1.3 ciphersuites to be used in the handshake. Node’s TLS library is built on using OpenSSL commands to generate initially shared keys.
Replay Attack

As the challenge that is issued to the client is a once-used nonce value, the risk of replay attacks is significantly lowered. Once a user is verified by signing the challenge given to them, the same challenge is not used in any future transactions. In order to further improve this, a time constraint could be introduced to the challenge to reduce the time for a bad actor to input a response into the system.

Reflection Attack

As the protocol authenticates the client and server using different methods and unique responses are sent for every authentication request, responses cannot be reflected back to the server to authenticate the user.

4.3.2 HTTPS Headers

In an effort to reduce the amount of potentially vulnerable headers needlessly accessible to bad actors, the following header options are set for all HTTPS requests/responses across the project:

- **HSTS** is set to force all communications with the server to use HTTPS, reducing the chance of sending messages over unencrypted HTTP.

- The **X-Powered-By** header is removed so names of libraries used to create the website are not broadcasted along with server responses.

- The **X-Download-Options** header is set to ‘noopen’ to block potentially unsafe downloads when using internet explorer.

- The **X-Content-Type-Options** header is set to ‘noSniff’ to prevent guessing of MIME media types for any multimedia displayed on the website. This attack is called MIME sniffing, hence the option name.
• The **X-XSS-Protection** header is set to ‘0’ following this not working correctly on some browsers and causing a security vulnerability issue when applying XSS filters.
Chapter 5

Evaluation

5.1 Introduction

In this section I will discuss the results of the work done in this dissertation, both in terms of physical outcomes of the application I developed and the learnings discovered throughout the entire research process.

From the research and development undertaken to investigate whether a decentralised website login system could be created using DIDs, I have demonstrated the following findings:

5.2 Project Analysis

In order to develop a fully functioning login system, it was necessary to create a protocol that specifies how messages are sent to authenticate DIDs. This is achieved with the proposed DID Login protocol which adapts ideas from DAuth [19] and DID Auth [23] to the web-based context of my application.
The DID Login protocol is similar to other forms of 3rd-party federated login mechanisms as it doesn’t require websites to directly store information related to it’s users for the sake of authentication (i.e. usually a username/password combination). The key difference between it and standard federated login systems, however, is that the verifying party is a decentralised blockchain as opposed to a central authority. As an alternative to token-based authorisation protocols (such as OAuth), the DID Login protocol issues a session token to any entity who has securely proven private key control of the DID being used to log in.

5.3 Benefits of Implementation

5.3.1 Privacy

By using the defined DID Login protocol, user privacy is greatly improved when compared to other centralised identity verification mechanisms. In order to authenticate users, no personal information is required to be stored or processed, unlike centralised and federated IDM protocols. In my application, Users are uniquely identified by their DIDs and authenticated by proving control of the associated private key. Because of this, the risk of user’s private information being exposed is drastically reduced for websites using the DID Login protocol. In the case where there is a security breach (either by an internal data leak or external malicious attack), the only information that will be exposed is a list of DID identifiers and their associated interactions with the website (such as a product purchase history, in the case of an e-commerce website).

The main risk to privacy that may still occur with my implementation would be a linkage attack. In the case that users reuse a DID as an identifier for several websites, it is possible that different pieces of data could be collated to reveal their identity. In an effort to reduce the likelihood of this occurring, users should use a different DID for each website they use. To improve privacy further, a user can selectively delete DIDs for websites that possess more data about them that they would like.
Privacy preservation is one of the main benefits of using a decentralised login mechanism rather than a centralised or federated one (Both of which have the potential to expose information such as usernames, email addresses, hashed passwords and phone numbers). According to Rana et. al. [20], when these specific pieces of information are leaked, they are more likely to allow exposure of other high-risk PII nodes when compared to national identity card leaks. This means that even if a user has shared a specific aspect of their identity (such as student or elderly status) using the verifiable claims mechanism of DID Login, the risk of further identity exposure is still less than a traditional, centralised login system.

5.3.2 User-controlled Data

As my application uses DIDs as a login mechanism, ownership of identifiers remains with the user rather than relying on the website itself or another 3rd-party to safely and ethically process user data.

It is always possible for users to update, delete or create entirely new DIDs. It is this self-sovereignty that allows users to increase their privacy control by deleting DIDs as mentioned above. This can be done using the 'Deactivate' method as specified in the DID Method of the specific blockchain in use. For DIDs on the Ontology blockchain, this can be done by removing all public keys from the DID document.
5.4 Future Work

5.4.1 Usability

Although privacy-preserving and secure, the DID Login application requires some basic knowledge of DIDs and blockchain wallets to use properly. Along with the benefits of shifting control of identity data to users, it also shifts the responsibility of data security to users. As well as this, the user has to be able to run the command line signature-creating script in order to be authenticated by the web server. Some form of direct integration with a user’s DID wallet to allow for signature creation would help to improve the usability of the application.

5.4.2 Interoperability

In the proposed implementation, only DIDs that are attested by the Ontology blockchain can be used to authenticate users. In future work, it would be beneficial to include DID authentication across many different DID methods to allow other DIDs to be used as a login identifier. This could be done by integration with the Universal DID Resolver [3] which is a project aimed at collating all ‘DID Read’ operations into a single resolver. In this way, users would have the choice to use identifiers that are attested by blockchains other than Ontology.

5.4.3 Business Incentive to use DID Login

In order to implement the DID Login mechanism, website owners must give their users the option to login using a ‘Decentralized Identifier’. Firstly, website owners must trust that the cryptographic operations do in fact authenticate users and secondly, they must be willing to include it in their own website. Aside from ethical incentives to improve user privacy and give control of identifiers to users, there seem to be few business incentives to implement the DID login mechanism. Particularly for larger social media
sites that may profit off of user data in some way, the reasons to adopt this login mechanism seem lacking.

Due to the nature of the application I have developed, it is entirely up to website admins whether they adopt this technology or not. Future work into investigating how the mechanism could be made more appealing to website owners would be beneficial.

5.5 Closing Remarks

While there are aspects of the project that could be improved to increase the usability of my application, I think there would be huge benefit in implementing a DID-based web login mechanism similar to the one proposed in this dissertation. As large technology companies continue to monopolise control of user data, it is more important than ever to provide users with more choice in the internet systems and protocols they use especially when these protocols give users more control over their personal information.

From the research done in this dissertation, I am convinced that a decentralised login system that utilises the functionality of DIDs is not only possible but is preferable to traditional forms of website login. The DID Login protocol that I have implemented achieves a complete login mechanism where no user data is required to be stored in order to authenticate them. Users are also afforded total control of their identifiers. These aspects show how blockchain technology can be used to facilitate a truly self-sovereign identity management infrastructure.
Bibliography


