

**Self-Sovereign Identity using Smart Contracts
on the Ethereum Blockchain**

by

Zachary Diebold

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfilment

of the requirements

for the degree of

Master in Computer Science

University of Dublin, Trinity College

Supervisor: Dr. Donal O'Mahony

May 2017

Declaration of Authorship

I, Zachary Diebold, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Zachary Diebold

May 18, 2017

Self-Sovereign Identity using Smart Contracts on the Ethereum Blockchain

Zachary Diebold, Master in Computer Science
University of Dublin, Trinity College, 2017

Supervisor: Dr. Donal O'Mahony

Centralised identity services that exist today fail to operate transparently and protect the rights of users. Single points of trust present constant operational risks for both companies and individuals. Self-sovereign identity is a solution to address this, which specifies a user-focused approach that gives full control of an identity back to the individual. This paper proposes the blockchain, a secure and decentralised trust-less system, as the platform to achieve this. A proof-of-concept identity system for the *Ethereum blockchain* is designed and developed in this paper. *Smart contracts* are used to facilitate the secure storage and open processing of user data. It also presents a novel approach to the secure recovery of encrypted private data. Emphasis is placed on the implementation security, information privacy and data recovery procedures of the system.

Summary

Digital identity systems that exist today are fragmented between service providers. Users need to duplicate their identity information between services, which reduces overall usability and increases the risk of data compromise.

In addition, logically centralised providers that offer single sign-on methods restrict the end-user from controlling how their data is stored and processed. Trusting centralised entities to manage sensitive information can lead to issues like identity theft, data leaks and privacy breaches.

Self-sovereign identity is a user-focused approach to identity management, that ensures full data control and transparency is retained by the individual. Self-sovereign identity also serves to protect the rights of the users from the ever-increasing control of centralised entities.

Blockchain technology offers a decentralised, transparent and immutable platform on which to safely transmit and track assets like digital currency. The Ethereum blockchain extends this concept by allowing programmable pieces of code known as *smart contracts* to be executed. These smart contracts can manage digital currency, and store and verify arbitrary data.

This paper investigates existing identity solutions, blockchain technology and decentralised systems. It then proposes a solution implemented with Ethereum smart contracts that encompasses the elements of cryptographic security, operational transparency, data autonomy and account recoverability. It is finally evaluated under a set of design objectives and comprehensive security considerations.

Acknowledgements

I'd like to express my sincere gratitude to the following people for their support:

- My supervisor, Dr. Donal O'Mahony, for his unwavering enthusiasm, immense guidance and continuous support throughout my research.
- The kind folks over at IPFS, uPort and the Solidity support channels for their extensive answers and technical advice.
- My fellow classmates in Integrated Computer Science, for their shared advice, dedication and energy during our college career.
- Finally, my loving family, for their constant encouragement, help and guidance throughout all my endeavours in life.

ZACHARY DIEBOLD

University of Dublin, Trinity College

May 2017

Contents

Declaration of Authorship	i
Abstract	ii
Summary	iii
Acknowledgements	iv
List of Figures	x
List of Tables	xi
List of Acronyms	xii
1 Introduction	1
1.1 Motivation	1
1.2 Research Objectives	2
1.3 Use Cases	3
1.4 Technical Approach	4
2 Digital Identity	6
2.1 Introduction	6

2.1.1	Definitions	6
2.1.2	Know Your Customer	8
2.1.3	Self-Sovereign Identity	9
2.2	Identity Management Systems	10
2.2.1	Overview	10
2.2.2	OpenID	11
2.2.3	SAML	11
2.2.4	OAuth	12
2.2.5	Discussion	13
2.3	Public Key Infrastructure	13
2.3.1	History	13
2.3.2	Certificates	14
2.3.3	Certificate Authorities	14
2.3.4	Attribute Authorities	15
2.3.5	Signatures	15
2.3.6	Web of Trust	15
2.3.7	Discussion	16
3	Blockchain	17
3.1	Introduction	17
3.1.1	History	17
3.1.2	Digital Wallets	17
3.1.3	Transactions	18
3.1.4	Distributed Consensus	19
3.2	Ethereum	20

3.2.1	Smart Contracts	20
3.2.2	Ethereum Virtual Machine	21
3.2.3	Discussion	22
3.3	Decentralised Storage	22
3.3.1	IPFS	23
3.3.2	Swarm	23
3.3.3	Discussion	24
3.4	Blockchain Identity Systems	24
3.4.1	Blockstack	24
3.4.2	Estonian e-Residency	25
3.4.3	Evernym	25
3.4.4	ShoCard	26
3.4.5	uPort	27
3.4.6	Discussion	28
4	Solution Design	29
4.1	System Overview	29
4.1.1	Smart Contracts	30
4.1.2	Identity Creation	31
4.1.3	User Attributes	32
4.1.4	Attribute Signing	33
4.1.5	Attribute Disclosure	33
4.1.6	Identity Recovery	35
4.1.7	Key Revocation	36

5	Implementation	37
5.1	Components	37
5.1.1	Web3 Framework	37
5.1.2	Transaction Signing	38
5.1.3	TestRPC	38
5.1.4	IPFS	38
5.2	Interface	39
5.2.1	Main Screen	39
5.2.2	Data Transfer	40
5.3	Additional Features	42
5.3.1	Deterministic Wallet Seed	42
5.3.2	User Attribute Privacy	42
6	Security Evaluation	44
6.1	Attack Considerations	44
6.1.1	Disclosure Replay Attack	44
6.1.2	Man-in-the-Middle Attack	45
6.1.3	Multi-User Compromise	45
6.1.4	Sybil Attack	46
7	Conclusion	48
7.1	Research Objectives	48
7.2	Future Work	50
7.2.1	Metropolis Release	50
	Bibliography	52

Appendices	59
Appendix A Contract Source Code	60
A.1 Identity Contract	60
A.2 Recovery Contract	61
A.3 Project Repository	62

List of Figures

3.1	How the Bitcoin blockchain propagates transactions	18
4.1	Identity Creation: Key generation and contract deploy steps.	32
4.2	Attribute Disclosure: Steps to disclose user attributes to a third party.	34
4.3	Identity Recovery: Steps to recover an identity using a network of trusted peers.	35
5.1	Home Screen: Initial view of a user's identity on the platform.	40
5.2	Signature Request: An example using a QR code to request a signature.	41

List of Tables

4.1	Identity contract storage variables	30
4.2	Recovery contract storage variables	31
5.1	Message types and values for passing data between users.	41

List of Acronyms

PKI Public Key Infrastructure

GDPR General Data Protection Regulation

KYC Know Your Customer

AML Anti-Money Laundering

IPFS InterPlanetary File System

OP OpenID Provider

RP Relying Party

SAML Security Assertion Markup Language

SSO Single Sign-On

MSD Mean Shortest Distance

CA Certificate Authority

UUID Universally Unique Identifier

JSON JavaScript Object Notation

OCSP Online Certificate Status Protocol

DLT Distributed Ledger Technology

MRZ Machine Readable Zone

DHT Distributed Hash Table

IPNS InterPlanetary Naming System

RPC Remote Procedure Call

BIP Bitcoin Improvement Proposal

PII Personally Identifiable Information

Chapter 1

Introduction

This section introduces the motivation for the dissertation work. The background to the field is explored, as well as the desired research objectives and possible challenges. Finally, the technical approach to implementing the planned work is outlined.

1.1 Motivation

The concept of identity is fundamental to human activities. The identity of an individual, their attributes, and representation of *the self* is understood worldwide regardless of culture. Psychological and social well-being is also said to be dependent on maintaining a stable personal identity [1].

Modern society binds the notion of individual identity with a legal identity, such as social security numbers, passports and driving licences. This binding leads to situations where one can lose their identity if the state rescinds it. It is especially apparent in the case of refugee crises, where migrants have difficulty living in countries in which they are not registered.

Digital identity represents entities such as people and organisations in an online

context. In its current state digital identity is siloed between providers, necessitating the creation of many identities across the internet.

Due to this fragmentation, users eventually link multiple identities to their legal identity over time, which can pose risks of identity theft, privacy breaches and data leaks [2].

As the digital world becomes more prevalent in everyday life and the quantity of internet-connected devices increases, the risks of trusting disparate centralised services become very high. As a result of this, an opportunity arises to redefine modern concepts of digital identity in a decentralised context.

1.2 Research Objectives

Self-sovereign identity is the notion that an individual should have complete authority over their digital identity and its data. It removes control from centralised entities and pushes it out to the edges of the network, the users themselves.

The core mission of blockchain technology is to provide a decentralised trust network. This can be mapped nicely onto the problems with centralised system trust in the identity landscape.

Thus, this dissertation proposes a system using the Ethereum blockchain, a decentralised application platform, to create a self-sovereign identity solution. Such a solution, in pursuit of this goal, should conform to the following requirements:

1. **Data Security:** Identities and their corresponding attributes should be stored in a secure manner, to prevent data compromise. Access and modification of user data should be restricted to the sole owner of the data.

2. **Interface Usability:** To interface with the system, it should not require prior knowledge of the underlying blockchain technology, key management procedures or encryption protocols. Public Key Infrastructure (PKI) is an example of a system that has not reached widespread consumer adoption, partly due to poor system usability [3].
3. **Identity Verifiability:** Individuals should be uniquely identifiable in the system, and have the ability to prove ownership of their information. External entities should be able to attest to the attributes of an individual.
4. **Account Recovery:** The system should allow for identity recovery in the case of device loss or data compromise. Revoked identities and attributes should also be discoverable by third parties.
5. **Self-Sovereignty:** The storage of data should not depend solely on a centralised entity, and as such the identity should be under full control of the user.

1.3 Use Cases

Such a self-sovereign system provides full autonomy over a user's personal identity, reputation and online presence. Example use cases for a decentralised self-sovereign identity platform are given below for context.

- Users can maintain a single identity for connecting to online services, such as social networks or enterprise systems. The system provides authentication of the user to the online service to prove ownership of their identity. This allows controlled single sign-on without the need for transferring passwords or secret information, as only the proof is transmitted.

- Governments can attest to the citizenship status or legal attributes of citizens, to then verify them in later interactions. Self-sovereignty and portability of the data allows it to be used internationally across borders.
- Financial institutions can establish an improved Know Your Customer (KYC) process, to verify the legal identities of customers using their services. Liability is reduced by not storing raw customer data, while still complying with Anti-Money Laundering (AML) regulation.

1.4 Technical Approach

The Ethereum blockchain, while operating in a similar manner to Bitcoin by facilitating currency transfers, can also store programs known as *smart contracts*. These programs are compiled to machine code and immutably deployed to the blockchain, and are verified and appended to the shared public ledger of transactions. The application state stored in these smart contracts is transparent and verifiable, and the execution of contract code is supported by the decentralised network.

Users in this system generate public and private keys corresponding to Ethereum wallets, and subsequently deploy Ethereum contracts containing their unique identifier. The generation and deployment of the smart contracts is done in the client, using a JavaScript framework for interfacing with Ethereum known as *Web3.js*.

Identity attributes are be stored on the decentralised storage platform known as InterPlanetary File System (IPFS), which allows for the data to be stored in a redundant and immutable manner.

As the core of the platform is the Ethereum smart contract, the client-side interface is seen as complementary and not central to the operation of the system. All the

security and access-control features of the system are contained solely in the immutable contract code.

Chapter 2

Digital Identity

2.1 Introduction

2.1.1 Definitions

Identity is a set of characteristics and attributes that describe an entity, that can then be used to uniquely identify it. Digital identity is the representation of an entity in a specific environment or context. It consists of unique identifiers, descriptive attributes and data claims related to the system for which it is issued.

To understand the variety of identity platforms that have existed over the years, some core concepts about identity should first be strictly defined.

Identifiers

An *identifier* is a piece of information used to distinguish a distinct person or entity in the context of a system. As noted by security researcher Steve Riley [4], this is the user's answer to the question "*Who are you?*". An entity can possess multiple identifiers associated with different functions. An example of an identifier for a person,

is their name, social security number, email address or credit card number.

Authentication

Authentication of an entity is performed by proving an association between a user and their identity. Again, this answers the question "*Ok, how can you prove it?*". The system will present a challenge that the user must answer with secret or private information known as the *authenticator*. Authenticators can include passwords, PINs and private keys. In the case of passwords, the system compares against secret information it already holds. For private keys, this is different, as it only needs to verify a proof that you know the secret value and does not require its transmission.

Multi-factor authentication is commonly used to enhance system security and has been noted by recent research [5] to consist of the following three methods:

1. **Something you know:** This is the most common form of an authenticator, like a password.
2. **Something you have:** This is a physical object, like a smart card or hardware key.
3. **Something you are:** This is an intrinsic attribute of the entity, such as a fingerprint or biometric value.

Authorisation

Authorisation is done to request permission to perform an action based on a particular identifier or attribute. This can be seen as the user asking the question "*What can I do?*". A system performs authorisation of the user against a database or rule set before approving their requested actions. The authenticated identity of the user is stored by

the system, once authorised, to enable future access to resources. This often consists of issuing a time-bound token or ticket to the user.

Attributes

Attributes are pieces of data related to an entity that helps describe it. The values of which can be persistent or temporary. An example of the attributes of a person are date of birth, gender, employer and physical address.

Attribute Attestations

Attestations or *claims* are verifications that establish the mapping from a given user attribute to their identity. In many identity systems, identity authentication precedes the attribute verification process, in order to confirm that the correct identity is being presented. An example of an attribute attestation is a bank attesting to the credit score of a customer, or the police attesting to the name and date of birth of a citizen.

2.1.2 Know Your Customer

Know Your Customer (KYC) is the process by which a business verifies the identities of its customers. This is often linked closely with Anti-Money Laundering (AML) state regulation and is enforced with the view to reducing criminal activity and fraud. Accurate identification of customers is required by law and mistakes are met with hefty penalties [6, 7].

Therefore, businesses in the financial sector have a strong economic incentive to ensure that stakeholders are accurately identified in their systems. This incentive speeds up the pace of innovation in this sector, in search of improved procedures and

enhanced technology.

2.1.3 Self-Sovereign Identity

Self-sovereign identity is a user-focused approach to identity, favouring the rights of the end-user. It proposes that individuals should have full control and autonomy over their identity and its data. These are succinctly defined by the cryptographer Christopher Allen [8] with his ten pillars of self-sovereign identity:

1. **Existence:** Users must have an independent presence in the system, and it must be an extension of their existing personal identity.
2. **Control:** Users are the ultimate authority over their identity, how it is used and how data is disclosed.
3. **Access:** Users must have easy access to their own data, without it being hidden or kept from them.
4. **Transparency:** Systems that manage the data, and the algorithms that analyse it, must be transparent, open-source and public.
5. **Persistence:** Identities must have the ability to be long-lived or permanent, at the discretion of the user.
6. **Portability:** Identities cannot be held by a single entity, trusted or not, and must be transportable between services.
7. **Interoperability:** Identities must have the ability to operate across systems, companies and borders.

8. **Consent:** Sharing of user data must be done with explicit consent and knowledge of the user.
9. **Minimalisation:** Data that is disclosed or shared should be minimised where possible. This is helped by the introduction of selective disclosure and zero-knowledge techniques.
10. **Protection:** The rights of the user must be central to the system, and be independent, decentralised and censorship-resistant.

These pillars may seem to be strict, but all ten are required if a truly self-sovereign system is to be produced. The General Data Protection Regulation (GDPR) in Europe [9] that will take effect in 2018 will legislate for some of these principles and puts the importance of this research into context.

Other research organisations like ID2020 [10] are already attempting to tackle these issues on a global identity scale, and are doing so by developing new technology, lobbying governments, and advocating for the principles outlined above.

2.2 Identity Management Systems

2.2.1 Overview

Identity management systems were introduced to address some of the risks outlined with siloed providers in Section 1.1. They attempt to solve the fragmented multi-account problem by centralising the user's identity data and standardising its usage. Some recent approaches are summarised below.

2.2.2 OpenID

OpenID is a protocol that was developed in 2005 by Brad Fitzpatrick, the creator of the popular blogging platform LiveJournal [11]. It allows users to create an account with an identity provider that supports the standard, known as the OpenID Provider (OP). The user can then use this account to authenticate with a third party service or Relying Party (RP), without managing multiple usernames and passwords.

The user must be sure to select a site that is reputable and long-lived, to ensure that the identity safely persists over time. OpenID removes the requirement for remembering passwords across many sites, but still leaves the users trusting their OpenID identity provider with important data. Accounts could be modified or deleted at any time at the discretion of the identity provider, so it does not provide much sovereignty to the user.

This inherent centralisation, coupled with the fact that users are forced to rely on an abstract identity system, eventually caused OpenID to lose prominence on the web [12].

2.2.3 SAML

Security Assertion Markup Language (SAML) [13] is an XML-based open standard for web browser Single Sign-On (SSO) using secure tokens. It is commonly used in enterprise contexts for authentication between applications and a company Active Directory. Identity providers pass identity information to service providers through digitally signed XML documents.

The four main components of the standard are as follows:

1. Assertions - These consist of authentication assertions used to prove the owner-

ship of an identity, and attribute assertions used to generate specific information about the person.

2. Protocols - These define how certain SAML elements like assertions are packaged for transfer and also the processing rules associated with them.
3. Bindings - These define the raw transport methods used to transmit the authentication data. These can include HTTP GET, HTTP POST and SOAP.
4. Profiles - These define in detail how assertions, protocols and bindings combine to form a use case. SAML 2.0 introduces many more profile contexts, including allowing the request to begin from the service provider itself.

The verbosity of the SAML standard and the fact that it can require extra infrastructure to support the transfer protocols [14] has restricted its deployment to legacy enterprise environments.

2.2.4 OAuth

OAuth [15] was introduced to allow a user to grant access to private resources connected to their identity. This is done without sharing private identity details or passwords between services. A long-lasting access token is specified by the protocol, which can be used by entities for continued access to user resources.

An example of this would be a user granting access of their Twitter or Facebook account to an external service, perhaps to allow it to make posts or gather analytics. The authorisation procedure and access token used by this external service would be managed using the OAuth specification.

OAuth is distinct from OpenID, as although it shares the common architecture of redirection for obtaining authorisation, it only manages the access control of resources. OAuth does not have a concept of identity and does not provide identity authentication.

OAuth and its updated standard OAuth 2.0 are both still in active use by many social networks and dependent applications, but it does not provide much value to the end-user in terms of identity sovereignty or control.

2.2.5 Discussion

Identity management systems, while attempting to reduce risks in the field, can still suffer from failure [16, 17]. This is owed to their inherent centralisation of data and resources. Therefore, there exists a gap in the state of the art for a management system not owned and managed by a single entity.

2.3 Public Key Infrastructure

2.3.1 History

PKI is a system for managing public-key encryption and digital certificates. It helps facilitate the secure transfer of information, and it authenticates parties operating over an insecure network. The principles behind this technology known as *asymmetric cryptography* were developed in 1976 by Whitfield Diffie and Martin Hellman [18]. This was also explored in 1978 by another set of mathematicians when the RSA algorithm for asymmetric key generation was published [19].

In asymmetric cryptography, a public and private key are first generated. The relationship between these is such that data encrypted with one can only be decrypted

by its matched pair. The public key can be widely distributed for identification of an entity, while the private key acts as a verifiable authorisation mechanism of that identifier.

2.3.2 Certificates

Critical to the functioning of PKI is the use of digital certificates, which give key pairs meaning and establish the identity of entities within a given context. Certificates contain information about the issuing party, the subject of the certificate, the expiry of the content and a digital signature. This digital signature is added to prove that the issuing party verified and signed the certificate with their private key.

The authenticity and integrity of such certificates and how they are issued must be protected, to maintain a level of trust in the system. Centralised parties that issue certificates can be susceptible to errors or failure, such as DigiNotar [20] and Symantec [21], so revocation procedures in the given system should be robust.

2.3.3 Certificate Authorities

A Certificate Authority (CA) manages the lifecycle of digital certificates and acts as an entity that both the subject of the certificate and the party relying on it trusts. Root authorities are CAs that are inherently trusted by the end user and are pre-installed on devices and browsers. CAs are fundamentally centralised entities that perform individual validation for every entity that wishes to be certified.

Certificates that are presented by a party can be traced back to a root authority to verify authenticity. This chain of trust underpins the PKI, and all transactions and exchanges of information are protected by the certificates it contains.

2.3.4 Attribute Authorities

Attribute Authorities [22] are similar to Certificate Authorities, but they exist when the separation between public keys and attributes is necessary. They can issue attribute certificates, which are used to control access to systems or resources.

This distinction from public key certificates is useful when the lifetime of an attribute certificate is different to that of the parent certificate. It is also sometimes required when the CA cannot authorise a user for accessing a particular resource, so a secondary authority must do so instead. The authorisation information for an entity is thus logically separated from its identity for these reasons.

2.3.5 Signatures

Digital signatures act as a proof for authenticity and integrity of a piece of data. It certifies that digitally signed information was produced by a trusted source and has not changed since the signature. Data is passed through a hashing algorithm which produces a fixed-length output that cannot be easily reverse engineered. This hash is then encrypted with the private key of the entity and sent with the data. The receiver can validate the integrity of the message by passing the data through the same hashing function and matching it with the decrypted signature.

2.3.6 Web of Trust

The Web of Trust [23] concept was introduced with PGP version 2.0, and it is a decentralised model of identity reputation achieved via a network of trusted parties. Users build up a reputation by having their keys verified and signed by their peers. A chain of trust can then be followed back to a locally trusted user when presented with

a new key.

A physical meeting in person is usually done to verify the mapping from an individual to their private key before creating a signature. Key signing parties [24] exist for this purpose, where multiple individuals sign each other's keys after identity verification.

The Mean Shortest Distance (MSD) value is used to calculate how trusted a given key is, by finding the average shortest distance or number of hops from the user's trusted keys.

2.3.7 Discussion

The core concepts of PKI are widely used on the internet to secure transactions and files. Data transfer over the HTTP protocol is encrypted and executable application files are signed using certificates.

PKI is suitable in the context of delivering content from enterprise to end-users, but it does not work well the other way around. Key distribution, management and revocation issues have prevented PKI from being widely deployed as a user-focused identity system.

Chapter 3

Blockchain

3.1 Introduction

3.1.1 History

The technology behind the blockchain was first conceived with the release of Bitcoin in 2008 by Satoshi Nakamoto [25]. The Bitcoin blockchain is a revolutionary idea that allows peer-to-peer transacting of digital currency in a trust-less environment. It uses cryptographic signatures to verify exchanges of currency, and a distributed ledger to maintain the order of transactions.

3.1.2 Digital Wallets

Digital wallets are used in the Bitcoin network to hold digital currency. All parties who hold a wallet have a public and private key pair. This is similar to PKI, which uses key pairs as an identification and authorisation mechanism. The public key is the address used to publicly identify the wallet and receive funds, while the private key is

used to make transactions using the wallet and prove ownership.

In a sense, the public key is an anonymous representation of identity on the network as it is not tied directly to an individual. However, research has shown [26, 27] that a method known as *chain analysis* can trace transactions as they propagate through the blockchain, thus implying that the network is instead pseudonymous.

3.1.3 Transactions

How the Bitcoin Blockchain Works

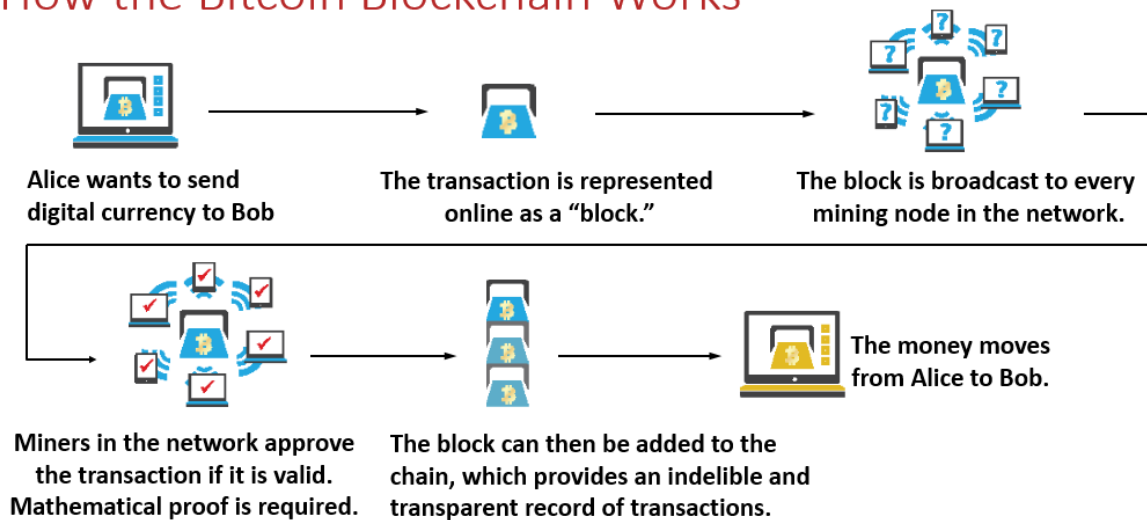


Figure 3.1: How the Bitcoin blockchain propagates transactions ¹

An example transaction is shown in Figure 3.1 above. If a user *Alice* wishes to send another user *Bob* digital currency from her wallet, she needs to first obtain the public address of Bob's wallet to create a new transaction. This transaction contains Bob's address, the payment amount, and the network fee. Alice must then sign this transaction with her private key, which proves to the network that she has cryptographic

¹Image source: D.Sleeter, "The Promise of Blockchain Technology," Jan. 2017. [Online]

ownership of her corresponding wallet address. Alice then broadcasts this transaction on the blockchain, where it is received by other machines known as *miners*.

3.1.4 Distributed Consensus

Bitcoin replaces a centralised payment ledger with a distributed one, but the problem arises of ensuring these transactions are ordered in a synchronised way. In the transaction between Alice and Bob above, the network needs to ensure that the coins being sent exist in Alice's wallet, and cannot be sent twice. This is referred to as the *double spending* problem [28].

A *proof-of-work* verification system is used to tackle this. It specifies a certain algorithm which takes a non-trivial amount of processing power time to compute but conversely the solution is trivial to validate.

When a transaction is broadcast to the network and received by a node, it is bundled with other unconfirmed transactions into a *transaction block*. This block contains a list of transactions, a link to the previous block, and a random nonce number. This number is incremented until the computed hash of the block including this nonce begins with a specified number of zeros, thus providing a method of proof of work.

Calculating this hash correctly is computationally intensive, and ensures that a certain amount of computing power is required to validate a block. Once a block has been created with the desired hash, as shown in the figure above, it is broadcast to the network, and other nodes can verify its legitimacy by adding it to the end of their chain of blocks. This block is now considered *valid*, and the order of transactions within the chain is thus preserved.

In return for using computing power to verify blocks, the operator of the mining

node receives some freshly generated Bitcoin currency in their wallet, as well as the network fees that users included in their transactions. This block reward began at *50 BTC*, but it has been halved twice since to combat inflation and now stands at *12.5 BTC* per block mined at the time of writing.

A key strength that comes with the ordering of transactions is that each transaction can be traced back to the original block where its source Bitcoin was minted. Before a Bitcoin node can start to mine and verify blocks of transactions, it must first download the entire history of transactions since the blockchain inception and verify each one independently. Only then does it begin to verify new transactions and add them to the blockchain.

3.2 Ethereum

Ethereum was released as an *"alternative protocol for building decentralised applications"* [29]. It builds on the blockchain features described in Bitcoin above, by adding programmability and scalability to the network. Ethereum does not just represent a digital currency, as Bitcoin does, but also allows many other use cases to be stored as application-level code on the blockchain.

3.2.1 Smart Contracts

Ethereum contains smart contracts, which are computer programs compiled and stored in the Ethereum blockchain. These are assigned addresses and can receive currency transactions just like standard wallets, but their functions can also be initiated with special types of function call transactions.

Smart contracts can accept parameters, store state, manipulate internal state as a

result of function calls, and return data in responses. This added functionality is not present in the Bitcoin blockchain, and it comes with distinct advantages. Application logic is performed and displayed transparently on the public blockchain, the internal state is available openly for scrutinising, and the operations are completely autonomous. They provide cryptographically auditable, append-only ledgers for building a new era of decentralised applications.

3.2.2 Ethereum Virtual Machine

Ethereum miner nodes are required to verify all state changes as blocks are propagated through the network. Application level code that is executed within smart contracts is considered a state change that is performed by the network via consensus.

To achieve this consensus through code execution, the *Ethereum Virtual Machine* was developed. This allows mining nodes to run code of arbitrary algorithmic complexity in a deterministic manner, and to reach consensus on the outcome of the computation. Smart contract operation is parallelised across all nodes in the network, which ensures fault tolerance, zero downtime, and permanent irrefutable state changes.

In the same way that Bitcoin miners accept network fees for verifying transactions, Ethereum miners that compute smart contracts receive fees for program execution. This is called the *gas price* of the transaction. The sender must pay for each line of code the program executes, including computation, events and storage. This is to discourage attacks like infinite loops in code from affecting the network.

3.2.3 Discussion

Ethereum provides secure transfers of value, auditable and autonomous program execution, fault-tolerant redundant storage, and an immutable record of information. This makes it a powerful platform on which to tackle global problems in a new decentralised paradigm. It's been considered as the first *Global Computer*, capable of operating completely censorship-free and across international borders.

The Ethereum platform has been chosen for the system design in this paper as it addresses many of the problems with centralised identity management. This is further outlined in the Solution Overview and Implementation details in Chapters 4 and 5.

3.3 Decentralised Storage

After the introduction of the blockchain, decentralised storage solutions have arisen that offer redundant hosting of long-form information and files. They present the following advantages:

- Low-latency data retrieval
- Efficient content caching
- Reliable fault-tolerant storage
- Censorship resistance
- File versioning and archival functionality

Some notable implementations are discussed below.

3.3.1 IPFS

IPFS [30] is a content-addressed distributed file system supported by a peer-to-peer network of machines. It can operate on any transport protocol and uses Distributed Hash Tables (DHTs) for peer identification and routing. It provides cryptographic-hash content addressing, file integrity, and filesystem encryption and signing.

IPFS also supports a name service known as the InterPlanetary Naming System (IPNS) for the persistent naming of dynamic content. This allows for routing of a static name to the hash of a file and is based on access control concepts from PKI.

The function of IPFS is similar to Bittorrent, whereby nodes serve local copies of content to the network. When files are requested, the local node caches the response and continues to seed the file back to the network. If all nodes serving a file go offline, then the file is no longer available. An economic incentive such as Filecoin [31] has been suggested as an addition to the protocol, which would encourage nodes to continue serving content for financial reward.

3.3.2 Swarm

Swarm [32] is a distributed storage platform and content distribution network similar to IPFS. Both projects provide decentralised storage of content-addressed files split into chunks.

Swarm is distinct in that it runs on the Ethereum peer-to-peer networking layer, and was developed in the context of close integration with the Ethereum blockchain. It also supports incentivised system benefits through smart contracts native to Ethereum via the pool of network peers.

Swarm is slightly behind IPFS in terms of development and global reach, and it is

possible that the two projects could integrate together sometime in the future [33].

3.3.3 Discussion

Decentralised storage is cheaper than using Ethereum contract storage, and it does not bloat the network with large amounts of data. It is therefore seen as valuable to use an external decentralised storage system for the solution proposed in this paper.

3.4 Blockchain Identity Systems

3.4.1 Blockstack

Blockstack [34] is a decentralised identity, discovery and storage platform, built on blockchain technology. It makes use of virtualchains [35] that allow the output of arbitrary state machines to be pinned to underlying blockchain infrastructure.

Blockstack is similar to Ethereum in that it supports decentralised applications, but instead performs its computation off-chain. The underlying blockchain technology is used to authenticate an application before it is run by the user. Applications are not Turing-complete by design, but they can interface with the Turing-complete Ethereum blockchain by use of a virtualchain.

Blockstack supports an identity project known as Onename, which allows users to register an identity on the Blockstack network. This features peer-to-peer identity attestations and verifications. Originally Onename used the Namecoin blockchain as its infrastructure, but changed to Bitcoin instead in response to centralised-mining and spam issues with Namecoin [36].

Although Onename supports a decentralised blockchain-based identity service, it

still relies on off-chain computation using Blockstack with many layers of resolvers and verifiers [37]. The lack of a stable and transparent network supporting the system reduces the usefulness of the project.

3.4.2 Estonian e-Residency

The Republic of Estonia released a state digital identity system known as e-Residency in 2014 [38]. It is a transnational secure identity offered by the government and supported by a physical smart card.

Citizens apply to the government with their legal information, including copies of their fingerprints, before being issued a digital identity. Residents can use the system for company registration, banking, payment services and document signing.

The cards use 2048-bit RSA encryption for document signing and verification. Legal documents can be digitally signed using this technology, with the full support of the Estonian legal system. The system currently does not use blockchain-based infrastructure, but it has partnered with initiatives like Identit.ee [39] and Bitnation [40] to pursue this in the future.

3.4.3 Evernym

Evernym [41] is an identity system built on the permissioned Distributed Ledger Technology (DLT) known as Sovrin, which is dedicated solely to decentralised identity.

The Sovrin network is supported by the Sovrin Foundation, and it consists of interconnected nodes forming a consensus on a shared ledger. Users create self-sovereign identities with personal attributes, and request claims from trusted third parties to build reputation.

Currently, there is no financial incentive to host a Sovrin node, so the majority of the network is research-focused. The network plans to introduce premium claims in the future to provide rewards to nodes that distribute and verify identities [42].

3.4.4 ShoCard

ShoCard [43] is a digital identity application focusing on user identification in the travel sector. It offers a mobile application for storing identities, while also pinning hashed and signed identity data to the Bitcoin blockchain.

Users scan their document with the application, which reads each Machine Readable Zone (MRZ) and stores an encrypted version on the device. Each field is then one-way hashed, signed with the user's private key, and published to the blockchain.

Disclosure of user data is done by encrypting the local copy of information with the receiver's public key and transferring it via a QR code. The receiver can then validate the information against the signed version published on the blockchain.

ShoCard specifies that the receiving party, such as an airline gate agent, checks the digital copy against the physical passport before proceeding. The airline can then create certification records confirming this physical and digital link, and hand them back to the user. This is referred to as a *travel token*. The user can then present this token at subsequent passenger checks to streamline verification. However, each checkpoint is still required to compare the token against the blockchain. This is done to check for continued validity and possible revocation of the token.

ShoCard presents a useful data disclosure process, ensuring transferred data is checked against the blockchain during each transaction. It fails to support any key or identity recovery protocols, however, and is inherently tied to identity supported by

physical documents only.

3.4.5 uPort

uPort is a self-sovereign identity platform built on the Ethereum blockchain. At its core, it utilises a network of smart contracts for each user and offers a mobile application and accompanying developer libraries.

The uPort mobile application generates a public and private key for a user and deploys smart contracts to represent their identity. It deploys what is known as the *Proxy Contract* to represent the user's unique identifier, the *Controller Contract* to provide identity access control logic, and the *Recovery Quorum Contract* to facilitate the recovery of the user's identity. It also stores pointers to these contracts in a centralised *Registry Contract*.

uPort presents a novel recovery concept for digital identity, where a quorum of the user's friends can collaborate to restore access to an identity. Friends can vote to replace the ownership key of an identity with a newly-generated one, and the identity contract logic performs this request when a majority consensus is reached.

uPort retains some centralised elements, as it uses a centralised messaging server to transfer attribute information from a user's device, a push notification system and application manager. It also requires the use of a central registry to log the mapping of user public keys to unique identifiers for convenience. These elements are deemed necessary by the project for initial user onboarding, but have the potential to be removed in the future [44].

In addition, uPort does not support the recovery of private data, as this is stored off-chain on the user's device. Sensitive data cannot be stored in public form on the

blockchain without first being encrypted. This research identifies a possible approach to allow this data to be recovered.

3.4.6 Discussion

There are many approaches to providing self-sovereign in the digital space, with promising blockchain implementations. The primary areas of interest that remain unsolved are completely decentralised systems that support the recovery of private identity data.

The focus of this research is to propose a solution that builds upon the advancements of existing work in the field, and to use stable and transparent blockchain architecture with complete privacy and adequate recoverability.

Chapter 4

Solution Design

The proposed solution builds on some of the advancements in the field of self-sovereign digital identity as shown in Section 3.4. It focuses on a decentralised model of data storage and communication, with privacy-preserving features and identity recovery.

4.1 System Overview

The system is comprised of three primary parts:

1. **Smart Contracts:** These are immutable programs stored on the Ethereum blockchain. Their function is to store the unique user identifier, a pointer to the user's data and the logic for modifying this data.
2. **Data Storage:** The user data is stored in JavaScript Object Notation (JSON) format on the decentralised storage platform IPFS, with a reference to this data given to the smart contract.
3. **Device Key Pair:** This is a public and private key pair stored on the end user's device, used for authenticating the user and allowing them to access and update

their identity.

4.1.1 Smart Contracts

The smart contracts are the core of the user's identity. There are two smart contracts that the user deploys onto the blockchain, that represent their identity.

The source code for these contracts is necessarily simple, to facilitate robust code execution and transparent auditing. This can be viewed in detail in Appendices A.1 and A.2.

Identity Contract

This contract contains the authoritative version of the user's identity, as well as the access control logic for attribute modification. Once this contract is deployed to the blockchain, the reference address that is returned is the user's Universally Unique Identifier (UUID). This address never changes, and it ensures the user maintains a persistent identifier even if their personal access keys are recovered or updated. The data stored in the contract is shown in Table 4.1 below.

Contract Data	Purpose
<code>owner_key</code>	Public key of the identity owner
<code>recovery_contract</code>	Address of the associated recovery contract
<code>ipfs_hash</code>	Hash pointing to the user data in IPFS

Table 4.1: Identity contract storage variables

The `owner_key` value represents the public part of the user's key pair stored on their device. The `recovery_contract` value is the address of the recovery contract described below. Requests to change user data are only accepted if they come from the listed public key or recovery contract.

Recovery Contract

This contract contains the logic to restore access to a user's identity. It stores a list of the user's friends that have been selected to facilitate recovery.

Contract Data	Purpose
uuid	Address of the user's identity contract
contacts_list	List of recovery contact addresses
recoveries	List of recovery requests submitted by contacts

Table 4.2: Recovery contract storage variables

The *uuid* is a pointer to the user's identity contract. The *contacts_list* stores UUIDs of accounts selected by the user. Requests to change the contacts list can only be done via the listed identity contract address. The *recoveries* value contains pending recovery requests for the identity, and it is cleared when a pending recovery is approved by a majority of peers.

4.1.2 Identity Creation

The steps for identity creation are shown in Figure 4.1. To create an identity on the platform, the user first generates a public and private key pair on their device. The public key acts as the local identifier for the user and is also an Ethereum address that can hold currency. The private key is used for signing transactions from this address, and to prove ownership of the public key.

The user then compiles a copy of the *Identity Contract* source code and publishes it to the blockchain via their Ethereum address. This transaction returns the Ethereum address to which the contract was deployed, and is used as the unique identifier or UUID for the user.

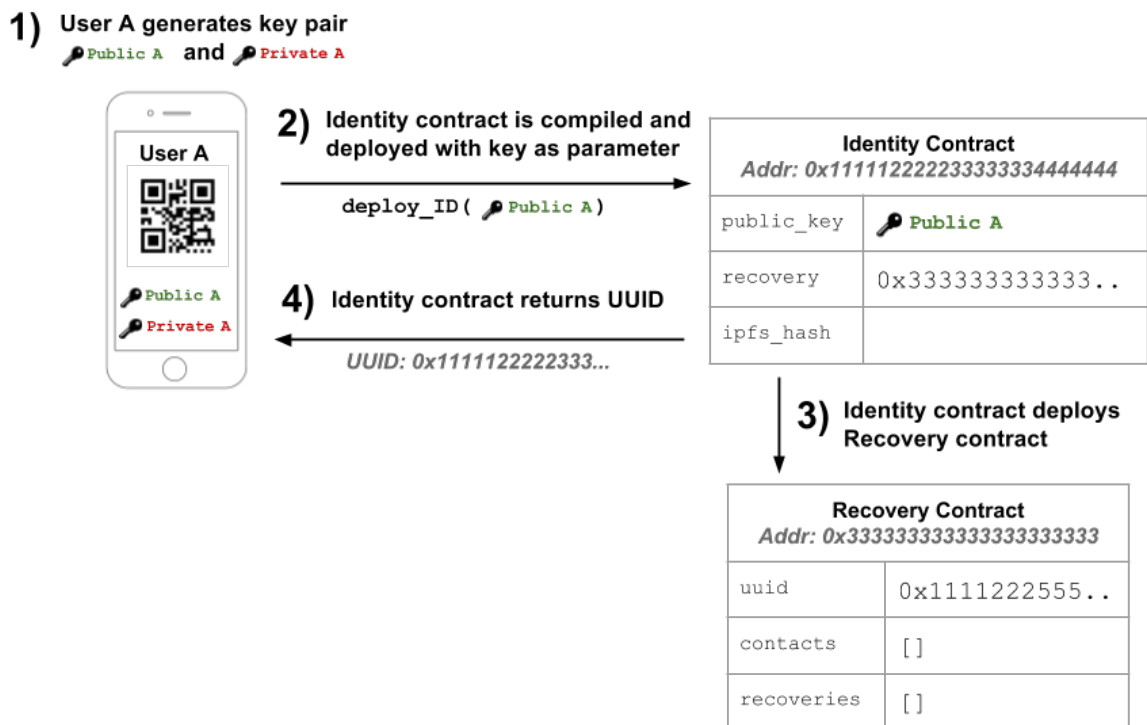


Figure 4.1: Identity Creation: Key generation and contract deploy steps.

This form of identifier is seen in the case of uPort in Section 3.4.5, and it is useful as it functions as both a unique address and a pointer to the contract data on the blockchain.

The identity contract itself also publishes a second contract, known as the *Recovery contract*. This contains the storage and logic for identity recovery using a consensus of the user's selected friends.

4.1.3 User Attributes

These are descriptive attributes relating to the user, for example, their name, address or date of birth. These are stored as JSON objects on the decentralised storage platform

IPFS. To store a user's attributes on their identity, they upload it to IPFS and receive a resultant hash pointing to the content on the IPFS network. The user can then send a signed transaction using their device public key to their identity contract which updates the IPFS hash.

4.1.4 Attribute Signing

Information about a user is not useful unless it is verified by a trusted third party. A user can request that a third party signs their attributes, and can then save the resultant signature with the rest of their identity details. Signatures contain the requested attribute, the UUID of the user, and a signature expiry time.

Smart contracts currently cannot perform signing functions on arbitrary pieces of text, and therefore the signatures must be done using the private key on the third party's device. Signatures performed using device keys are accompanied by the associated contract address for subsequent verification.

There is, therefore, a link between the user's device keys and the UUID value stored on the blockchain that must be consistently verified. All signed data must be checked against the blockchain to ensure that the public key used to sign is still matched with the correct identity.

4.1.5 Attribute Disclosure

To disclose a user's attributes to another party, the third party service creates a disclosure request with the required attributes. This is shown in the first step of Figure 4.2. When the user receives this request on the device, they can confirm or deny the disclosure of the requested data.

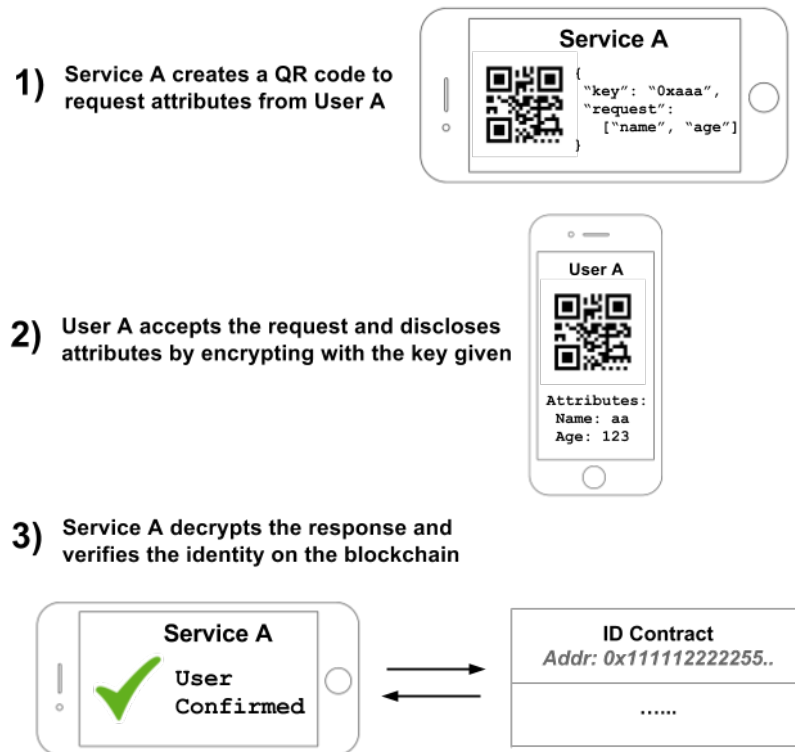


Figure 4.2: Attribute Disclosure: Steps to disclose user attributes to a third party.


The user then encrypts the attributes and any associated signatures with the third party's public key. The user also signs the request with the private key on their device and sends over the result.

The receiving party then decrypts this message and verifies that the public key used to create the signature is linked to the correct identity on the blockchain. The validity of any attribute signatures is also verified by querying the blockchain and checking that the signing parties are trusted.

Attributes of a user can only be considered valid if they are accompanied by attestations from third parties that are trusted by the receiver. An extension to this

solution would allow a chain of trust to be created that links entities to trusted roots. Listing the public keys of trusted parties on an online service like the MIT PGP Public Key Server [45] could facilitate this approach.

4.1.6 Identity Recovery

- 1) User A sends a request to friends to recover their UUID with  New Key 1
- 2) Friends access the identity contract to get the recovery contract address
- 3) They send a request to the recovery contract to update the public key

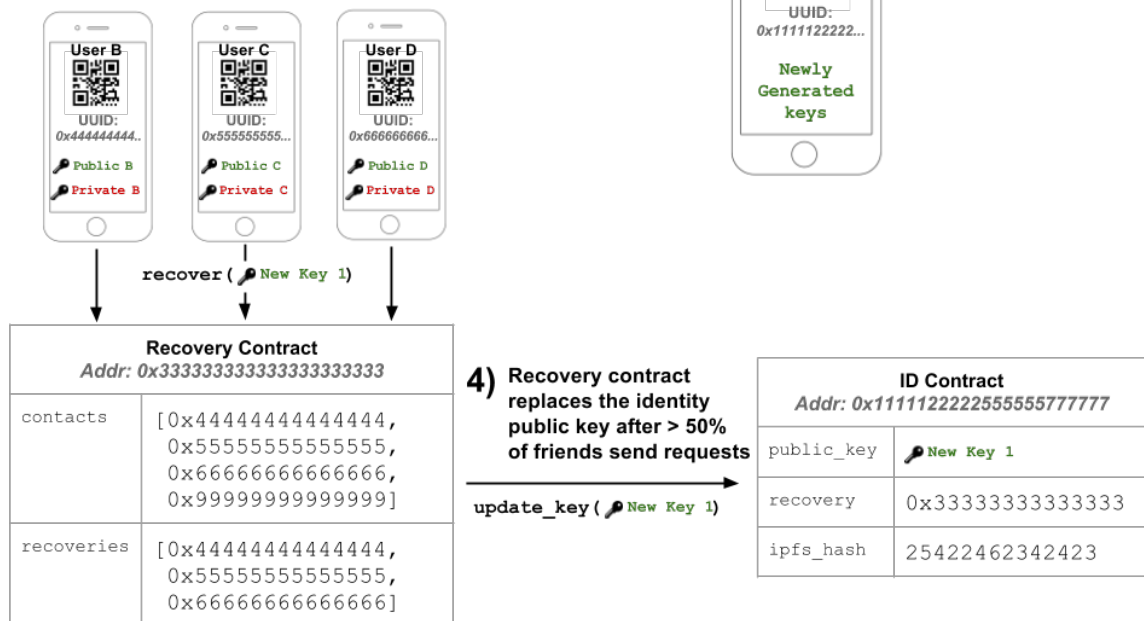


Figure 4.3: Identity Recovery: Steps to recover an identity using a network of trusted peers.

Recovery is a vital part of enabling public key cryptography for general use [46]. In this approach, users preselect a group of recovery contacts to facilitate reinstatement of the user's identity in the event of key loss or compromise. The list of contacts are

stored in the recovery contract connected to the user's identity.

If a user needs to recover their identity, they first generate a new set of public and private keys. This can be seen in Step 1 of Figure 4.3 above. The user then meets with their contacts and asks them to send recovery requests containing their new public key.

The contacts then access the user's associated identity contract to get the address of the recovery contact before sending the request. The recovery contract stores pending requests, and has the power to make the key change once a majority decision is reached.

4.1.7 Key Revocation

Key revocation is used to permanently retire signing and encryption keys from usage [47]. In the context of key loss or compromise, this is an essential function to prevent malicious actors from abusing the system.

As the user's identity contract and UUID is inherently tied directly to their device public key, this is the only valid mapping between these two values. This mapping is verified upon each signature and disclosure request, so separate key revocation does not need to be built into the system.

Key rotation within a user's identity should be logged for archival purposes, so that expired signatures can be verified as being valid at some point in the past. This can be achieved with Ethereum Event Logs [48], which act as a logging tool for smart contracts that is cheaper than internal storage. Parties can subscribe to the outputs of these events and be notified when they are announced.

The invalidation of old signatures when an identity has its keys rotated is necessary to maintain the integrity of the identity network. New valid signatures should be distributed to appropriate parties when a relevant key change occurs.

Chapter 5

Implementation

5.1 Components

5.1.1 Web3 Framework

Web3.js [49] is a JavaScript interface for Ethereum which conforms to the Generic JSON Remote Procedure Call (RPC) Specification [50] used by other Ethereum clients. It can send transactions, call functions in smart contracts and compile and deploy Solidity smart contract code.

Web3 was chosen as the interface for interacting with the Ethereum smart contracts, as it is a platform-independent framework that operates client-side and within the browser. Web3 can use a local Ethereum node for testing purposes, or it can be pointed at a remote node connected to the Ethereum *Test Net* or *Main Net*. Infura [51] is a service that offers public Ethereum nodes that serves blockchain RPC requests for applications.

5.1.2 Transaction Signing

Transaction signing is not offered by Web3, but it can be offloaded to the connected Ethereum node if the keys of the requested account are stored on the node. To enable local transaction signing in the client, packages like `ethereumjs-tx` [52] and `ethereumjs-util` [53] can be added to the project. These packages use private keys stored locally to sign transaction requests, and they subsequently send raw transactions to the Ethereum node.

Metamask [54] is another project for Ethereum account management that runs as a browser extension. It stores the public and private keys for Ethereum wallets in the browser local storage and supports client-side transaction signing.

5.1.3 TestRPC

TestRPC [55] can be used for rapid testing of Ethereum smart contracts and applications. It simulates a full Ethereum node and local blockchain network. It can generate a number of addresses with initial balances and store their keys on the node. It also mines blocks of transactions instantly to facilitate faster development.

The implementation of this project uses TestRPC, but it could be easily changed to point to an Ethereum node connected to the live blockchain. TestRPC was convenient as it gave generated wallets initial account balances, which removed the need for mining Ether to fund contract deployments and transactions.

5.1.4 IPFS

A local IPFS node was set up to store user identity data during development. This could also be easily pointed to a public IPFS node such as one hosted by the organisation

themselves.

Data on IPFS follows a standard format for storing attributes and signatures. An example is shown below.

```
{
  "uuid": "0xa55be2ad88201c3d206dde240d5dc4d356b627dc",
  "attributes": {
    "name": "Alice Alison",
    "birth-date": "1990-12-25",
  },
  "signatures": {
    "name": [
      {
        "signer": "0x2dd6cb3ed8290de01163750378e9b1bcb4df6eb5",
        "signature": "0x9aaf88db5cfaefa20c48a46b958b03667de79397b62
                    506263a39bbd8cf1d52c748791df2a3b4d7a5830c0db5
                    842bfed5f93b2cd9525cd0abbbe8c4b5cd80718e01"
      }
    ],
    "birth-date": [],
  }
}
```

Example snippet hash: *QmUHxAMb53UeAY6srn9xEBUthcBVT5sVrRRJrbwwkqxy3Z*.

5.2 Interface

5.2.1 Main Screen

The primary interface of the implementation is shown below in Figure 5.1. It displays the user's UUID, their name attribute, Ethereum wallet balance, device public key, recovery contract address and IPFS content hash.

The UUID, which is also the address of the user's identity contract, is stored in the local storage of the browser. This is then used to fetch the rest of the user's data.

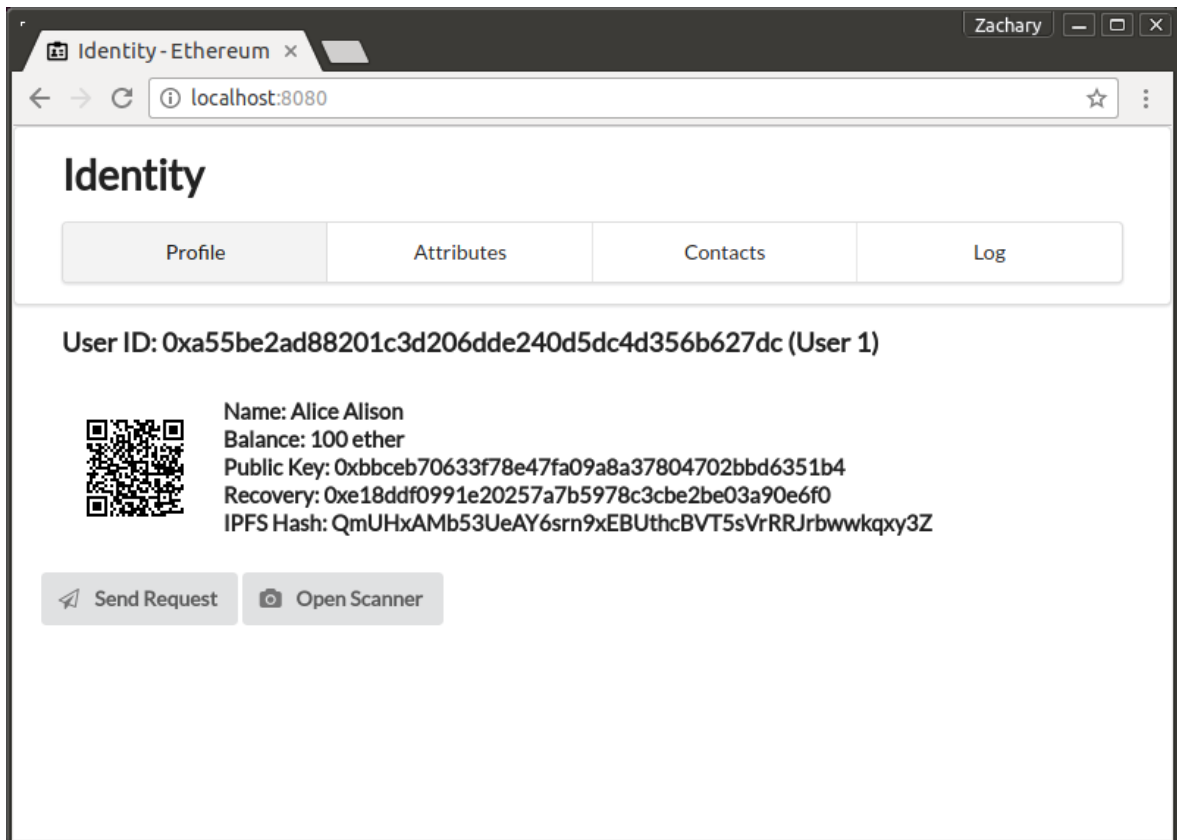


Figure 5.1: Home Screen: Initial view of a user's identity on the platform.

5.2.2 Data Transfer

QR codes were chosen as the transport protocol to transfer information between parties in the system. This can be replaced with any other peer-to-peer messaging protocol like Bluetooth or Wi-Fi Direct. Ethereum has plans to release a dedicated protocol known as Whisper [56] which could also be used in the future.

An example of the QR code used to transmit a signature request is shown in Figure 5.2 below. This shows the accompanying JSON value that contains the user's UUID, the name of the attribute to be signed, and the value. This can be scanned by a third party, approved, and then the attributes signed and returned.

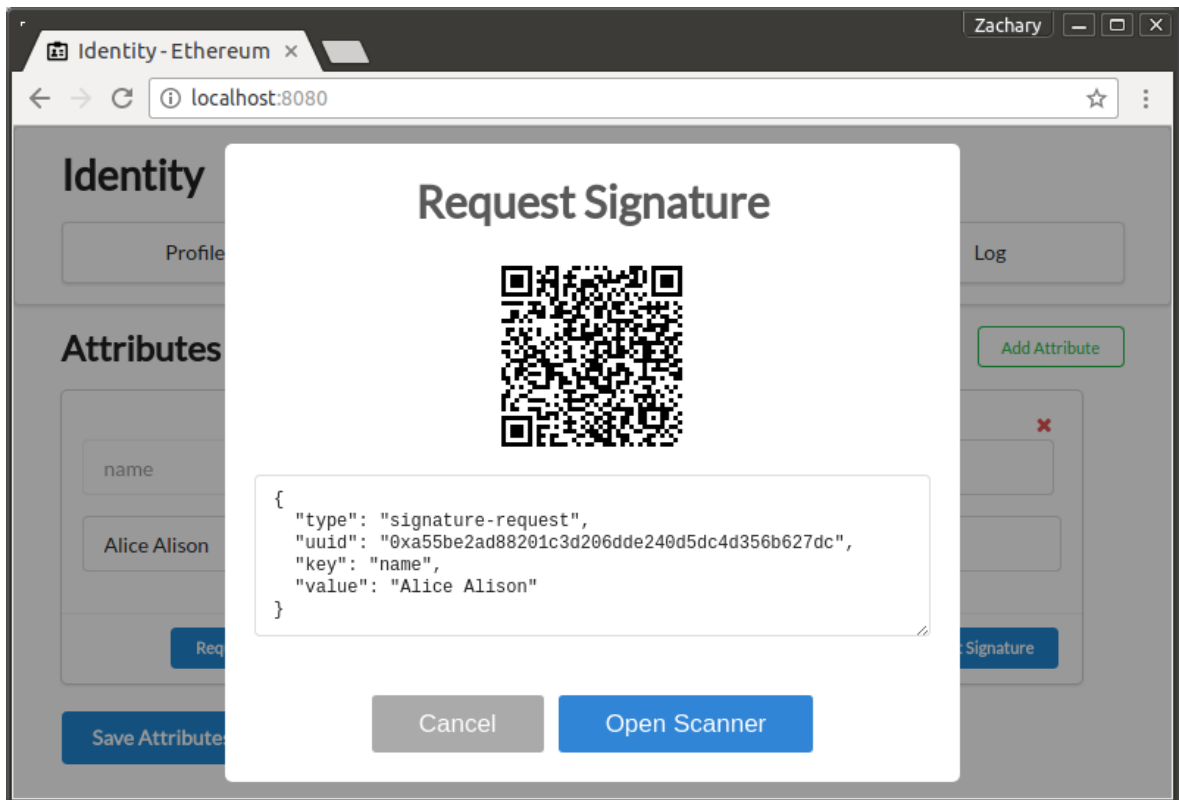


Figure 5.2: Signature Request: An example using a QR code to request a signature.

A table of the possible message types is also shown in Table 5.1 below. These are read by the JavaScript implementation and present different dialog boxes to the user.

Message Type	Attributes
contact-card	uuid
signature-request	uuid, key, value
attribute-request	attributes, uuid
recovery-request	uuid, new-key
signature-result	key, value, signer, signature
disclosure-result	uuid, attributes, signature

Table 5.1: Message types and values for passing data between users.

5.3 Additional Features

5.3.1 Deterministic Wallet Seed

Another recovery mechanism that could be added to the system is a recovery mnemonic, a human readable seed that is used to generate the public and private key pair.

A Bitcoin Improvement Proposal (BIP) known as BIP39 [57] was released in 2013 to help users make a backup of their digital currency wallets. It builds upon the advancements of deterministic wallet generation from a given seed brought about by BIP32 [58].

A twelve or sixteen-word seed is generated from a list of 2048 English words. This mnemonic is then hashed and converted into a seed value for the key generation function. *Hierarchical Deterministic* wallets allow an infinite amount of addresses to be generated from a given seed, using an index value.

Human readable seeds can thus be given to the user to write down on paper, and they allow multi-currency addresses to be recovered. This should act as the primary backup feature for the user's identity, as it is simpler than coordinating with multiple backup contacts.

5.3.2 User Attribute Privacy

User attributes are stored unencrypted on IPFS and linked directly to the user's UUID. This means that whenever the UUID is shared, the receiving party can view all the public attributes and signatures relating to that user. Other existing implementations such as uPort as seen in Section 3.4.5 allow the use of privately stored attributes that are not on the blockchain, but recovery of these is not possible in cases of device

compromise.

A proposed solution is to have a second "encryption key" in addition to the device signing key owned by the user. This encryption key would be used to encrypt all user data before publishing it to the public blockchain. Disclosure of attributes is done by downloading this encrypted data, and then decrypting it before encrypting with the receivers public key for transport.

As private data cannot be stored in Ethereum smart contracts, the encryption key is not recoverable using the blockchain. The function of storing encrypted data in smart contracts that they can act on is known as secure multi-party computation [59] and remains largely unsolved for blockchain technology. Some facility is therefore required to enable recovery contacts to restore a user's encryption key, to avoid them losing their stored private data.

Threshold cryptography [60] is proposed to solve this, which allows a piece of data to be split such that M of N pieces need to be combined to restore it. It allows secure sharing of a secret value such that no less than the specified number of peers can come together to reveal the data.

The private encryption key of the user could, therefore, be split using the threshold cryptography algorithm, and encrypted with the public keys of the user's recovery contacts. In the event of key loss or compromise, the user can request the decrypted values from M of N contacts and then run the data back through the algorithm to restore their private key.

Chapter 6

Security Evaluation

6.1 Attack Considerations

The security aspects of a system are especially relevant in the identity space, as Personally Identifiable Information (PII) that is hacked or leaked can have a detrimental effect on both users and companies [61].

This section documents some possible attack vectors for the implementation in this paper, and the solutions proposed to counteract them. These are considered *active attacks*, where the attacker is attempting to alter the live operation of the system. They are distinct from *passive attacks* where the attacker is attempting to merely obtain data for later use.

6.1.1 Disclosure Replay Attack

When user attributes are disclosed to a third party, they could be susceptible to a *message replay* or *playback attack* if the response is logged and replayed at a later time. This could lead to identity impersonation as the signed response can be verified

as valid [62].

A solution to this is for the requesting party to present a random string known as a *challenge* to the user. The user must sign this challenge with their private key. Upon receipt, this signature can be verified against the challenge string that was sent. These challenge strings are transient and regenerated for each request.

6.1.2 Man-in-the-Middle Attack

A *man-in-the-middle attack* occurs when an attacker intercepts and alters messages between two parties who are directly communicating with each other. The likelihood of this attack is related to the communication channel chosen by the parties.

The identity system in this paper can only attempt to mitigate this by ensuring that data is encrypted with the receiving party's public key for data privacy, and signed with the sending party's private key for message integrity. The key exchange step of the communication process is assumed to be safe, as the system cannot protect this.

The initial key exchange between parties should be done in person, with visual confirmation that keys transferred are correct. If this information remains unaltered, then future communication through other channels can be considered safe.

6.1.3 Multi-User Compromise

Identity recovery is supported by giving authority to a group of the user's trusted friends. One downside to this approach is that the list of connected friends is stored on the public blockchain, so they could be susceptible to compromise as a group. An attacker only needs to take over 51% of the user's contacts, which could be a small number if they do not possess very many.

Time-delayed actions are proposed to solve this issue, notably by uPort, with appropriate notification and logging. The Ethereum protocol can output *Event Logs* [48], to which other parties can subscribe to receive updates. When the recovery contract receives requests from a user's recovery contacts, it can delay any account changes by a number of days and notify accordingly.

This can be done by binding actions to the timestamp announced within the block, and ensuring they are not activated unless a specified delay has passed. Incoming transactions to the contract check for pending actions and perform the functions if the correct time has elapsed. This is referred to as lazy execution [63], and is required as there is currently no way for a contract to activate itself after a specific amount of time.

6.1.4 Sybil Attack

A *Sybil attack* [64] occurs where a reputation system is undermined by the creation of many forged identities. This can affect the credibility of the system and is exploited by attackers to gain a disproportionate amount of power in the network.

A proposed solution [65] subverts Sybil attacks by adding certain costs to parts of the system:

- Entry Cost - Barrier cost for creating an identity
- Existence Cost - Continuous fee for having an active identity
- Exit Penalty - Penalty fee or dismissal for acting dishonestly

The identity system proposed in this paper does not attempt to inherently prevent Sybil attacks, but is architected in the hope that attestations by reputable authorities

can bootstrap trust into the network.

The addition of *reputation scores* [66] given to identities is also a promising solution, which can be used to derive trust from a network of attestations. This is similar to the Web of Trust concept shown in Section 2.3.6.

Chapter 7

Conclusion

Identity in the digital sphere has been shown in this paper to be important both personally to users, and economically to businesses. The risks and negative effects that centralised identity systems possess imply that they are not a perfect solution for the identity domain.

Decentralised identity systems that take a more user-centric approach pose less risk to the service provider and provide a greater benefit to the end user. This aspect was focused on in the context of blockchain technology, to try and explore a viable and realistic solution to apply to the field.

7.1 Research Objectives

A number of research objectives were outlined at the start of this paper, with the view of evaluating their feasibility and relevance in the proposed solution. They are assessed under the headings below.

Data Security

The aim of this requirement is to ensure that user information kept secure and private. The approach proposed in this paper ensures data security in transit using encryption in Section 5.2.2. It also presents a novel approach to ensuring data at rest is kept private, using threshold cryptography with an encryption key as detailed in Section 5.3.2.

Interface Usability

The primary focus of the interface design was to abstract away the fine details of public and private key creation, management and usage from the user. This was achieved in part by ensuring that key generation is done automatically with identity creation, and key storage was managed by the client application. Key recovery is also facilitated by the recovery quorum process detailed in Section 4.1.6, which removes the requirement of careful key management by the user.

Identity Verifiability

The attribute signing and signature verification processes detailed in Section 4.1.5 guarantee that identities can be securely validated. A careful note can be made about the validity of user attributes, as these are only as trusted as the entities that provide the associated attestations. Even though attestations might be cryptographically valid, the network cannot verify *why* the claim has been made. Lists of trusted authorities should be cautiously monitored to maintain a high level of trust in the network.

Account Recovery

The proposed system supports identity recovery using a network of peers, which is robust to single user compromise. An approach is also shown to recover private identity data if the keys are distributed correctly between these trusted peers.

Self-Sovereignty

The self-sovereign approach to identity management is fundamental to this research, and providing full autonomy over a user's identity and associated data helps to accomplish this. By giving the user sole identity access and the power to delegate authority to peers, users are no longer dependent on third parties for managing or storing their data. The blockchain element is crucial for this complete data decentralisation, as it is one of the few technologies that fundamentally supports it.

7.2 Future Work

Although this paper presents a comprehensive design and proof-of-concept implementation of a self-sovereign identity system; there are still improvements that can be made as the technology matures.

7.2.1 Metropolis Release

The Ethereum blockchain is due to release a protocol upgrade soon, which will bring about improvements in transactions and smart contract operation.

Work on allowing smart contracts to pay the transaction fees for users could be included, which would remove the requirement for users to have an initial Ethereum

balance before joining the system. This is currently tackled by uPort by using a *currency faucet* to pay these fees [67], but by doing so, it introduces an element of external dependency in the system.

RSA signature verification could also be added to this release, and it has been discussed in a recent proposal [68]. This would remove the dependency on the client for identity verification and allow it to be done transparently on the blockchain.

Bibliography

- [1] S. Sharma and M. Sharma, “Self, social identity and psychological well-being,” *Psychological Studies*, vol. 55, no. 2, pp. 118–136, Jun. 2010.
- [2] L. J. Camp and M. E. Johnson, *The Economics of Financial and Medical Identity Theft*. Springer Science & Business Media, Mar. 2012.
- [3] T. Straub, “Usability Challenges of PKI,” PhD Thesis, Technische Universitt, Darmstadt, Apr. 2006. [Online]. Available: <http://elib.tu-darmstadt.de/diss/000682>
- [4] S. Riley, “Its Me, and Heres My Proof: Why Identity and Authentication Must Remain Distinct,” Feb. 2006. [Online]. Available: <https://technet.microsoft.com/en-us/library/cc512578.aspx>
- [5] F. B. Schneider, “System Security – Something You Know, Have, or Are.” [Online]. Available: <https://www.cs.cornell.edu/courses/cs513/2005fa/NNLauthPeople.html>
- [6] S. Nasiripour and K. Scannell, “UK banks hit by record \$2.6bn US fines,” Dec. 2012. [Online]. Available: <https://www.ft.com/content/643a6c06-42f0-11e2-aa8f-00144feabdc0>

-
- [7] Financial Conduct Authority, “FCA fines Deutsche Bank 163 million for serious anti-money laundering controls failings,” Jan. 2017. [Online]. Available: <https://www.fca.org.uk/news/press-releases/fca-fines-deutsche-bank-163-million-anti-money-laundering-controls-failure>
- [8] C. Allen, “The Path to Self-Sovereign Identity,” Apr. 2016. [Online]. Available: <http://www.lifewithalacrity.com/2016/04/the-path-to-self-sovereign-identity.html>
- [9] European Parliament, “EU GDPR.” [Online]. Available: <http://eugdpr.org/eugdpr.org.html>
- [10] Identity2020 Systems, Inc, “ID2020.” [Online]. Available: <http://id2020.org/>
- [11] B. Fitzpatrick, “Distributed Identity: Yadis,” May 2005. [Online]. Available: <http://lj-dev.livejournal.com/683939.html>
- [12] A. S. G. S. Gilbertson, “OpenID: The Webs Most Successful Failure,” Jan. 2011. [Online]. Available: <https://www.wired.com/2011/01/openid-the-webs-most-successful-failure/>
- [13] OASIS Security Services, “SAML Specifications | SAML XML.org,” Mar. 2005. [Online]. Available: <http://saml.xml.org/saml-specifications>
- [14] Z. Dennis, “An Updated Look At Choosing Between OAuth2 and SAML,” Jul. 2015. [Online]. Available: <https://www.mutuallyhuman.com/blog/2015/07/17/an-updated-look-at-choosing-between-oauth2-and-saml>
- [15] E. Hammer-Lahav, “The OAuth 1.0 Protocol,” Apr. 2010. [Online]. Available: <https://tools.ietf.org/html/rfc5849>

-
- [16] M. Mimoso, “Office 365 Vulnerability Exposed Any Federated Account,” Apr. 2016. [Online]. Available: <https://threatpost.com/office-365-vulnerability-exposed-any-federated-account/117716/>
- [17] A. Low and S. Rosenblatt, “Serious security flaw in OAuth, OpenID discovered,” May 2014. [Online]. Available: <https://www.cnet.com/uk/news/serious-security-flaw-in-oauth-and-openid-discovered/>
- [18] W. Diffie and M. Hellman, “New Directions in Cryptography,” *IEEE Trans. Inf. Theor.*, vol. 22, no. 6, pp. 644–654, Nov. 1976.
- [19] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
- [20] The Associated Press, “Hacking in the Netherlands Took Aim at Internet Giants,” *The New York Times*, Sep. 2011. [Online]. Available: <http://www.nytimes.com/2011/09/06/technology/hacking-in-the-netherlands-broadens-in-scope.html>
- [21] A. Ayer, “Misissued/Suspicious Symantec Certificates,” Jan. 2017. [Online]. Available: <https://www.mail-archive.com/dev-security-policy@lists.mozilla.org/msg05455.html>
- [22] S. Farrell and R. Housley, “An Internet Attribute Certificate Profile for Authorization,” Apr. 2002. [Online]. Available: <https://tools.ietf.org/html/rfc3281>
- [23] The Free Software Foundation, “The GNU Privacy Handbook,” 1999. [Online]. Available: <https://www.gnupg.org/gph/en/manual.html#AEN335>

-
- [24] A. Brennan, “The Keysigning Party HOWTO,” Jan. 2008. [Online]. Available: http://cryptnet.net/fdp/crypto/keysigning_party/en/keysigning_party.html
- [25] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008. [Online]. Available: <https://bitcoin.org/bitcoin.pdf>
- [26] D. Ron and A. Shamir, “Quantitative Analysis of the Full Bitcoin Transaction Graph,” in *Financial Cryptography and Data Security*. Springer, Berlin, Heidelberg, Apr. 2013, pp. 6–24.
- [27] F. Reid and M. Harrigan, “An Analysis of Anonymity in the Bitcoin System,” in *Security and Privacy in Social Networks*, Y. Altshuler, Y. Elovici, A. B. Cremers, N. Aharony, and A. Pentland, Eds. Springer New York, 2013, pp. 197–223.
- [28] J.-H. Hoepman, “Distributed Double Spending Prevention,” in *Security Protocols*. Springer, Berlin, Heidelberg, Apr. 2007, pp. 152–165.
- [29] V. Buterin, “Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform,” *GitHub*, 2014. [Online]. Available: <https://github.com/ethereum/wiki/wiki/White-Paper>
- [30] J. Benet, “IPFS - Content Addressed, Versioned, P2p File System,” Jul. 2014. [Online]. Available: <http://arxiv.org/abs/1407.3561>
- [31] Filecoin.io, “Filecoin: A Cryptocurrency Operated File Storage Network,” Jul. 2014. [Online]. Available: <http://filecoin.io/filecoin.pdf>
- [32] “Swarm - Incentivised Peer-to-Peer Storage and Content Distribution.” [Online]. Available: <http://swarm-gateways.net>

-
- [33] Ethersphere, “IPFS & SWARM Comparison,” Mar. 2017. [Online]. Available: <https://github.com/ethersphere/go-ethereum>
- [34] Blockstack Inc., “Blockstack: A New Decentralized Internet,” May 2017. [Online]. Available: https://blockstack.org/blockstack_whitepaper.pdf
- [35] J Nelson, M Ali, R Shea, and M.J Freedman, “Extending existing blockchains with virtualchain,” Workshop on Distributed Cryptocurrencies and Consensus Ledgers, Jun. 2016.
- [36] Kyle Torpey, “The 3 Reasons Why Onename Switched from Namecoin to Bitcoin,” Nov. 2015. [Online]. Available: <https://www.coingecko.com/buzz/3-reasons-onename-switched-from-namecoin-to-bitcoin>
- [37] Onename, “Introducing Blockchain ID, Your Digital Identity,” May 2015. [Online]. Available: <http://blog.onename.com/blockchain-id/>
- [38] The Digital Society, “Estonian e-Residency,” 2014. [Online]. Available: <https://e-estonia.com/e-residents/about/>
- [39] “Identit.ee - Innovating e-Residency and Identity.” [Online]. Available: <http://www.identit.ee/>
- [40] Bitnation, “Estonia E-residency Program & Bitnation Dao Public Notary Partnership,” Nov. 2015. [Online]. Available: <https://bitnation.co/blog/pressrelease-estonia-bitnation-public-notary-partnership/>
- [41] S. M. Smith and D. Khovratovich, “Identity System Essentials,” Mar. 2016. [Online]. Available: <http://www.evernym.com/wp-content/uploads/2017/02/Identity-System-Essentials.pdf>

-
- [42] Sovrin Foundation, “Sovrin - Frequently Asked Questions.” [Online]. Available: <https://www.sovrin.org/about/faq.html>
- [43] ShoCard Inc., “ShoCard: Travel Identity for the Future,” May 2016. [Online]. Available: <https://shocard.com/>
- [44] P. Braendgaard, “Response: uPort Centralisation,” Apr. 2017. [Online]. Available: <https://medium.com/@pelleb/db634cb3d48f>
- [45] Massachusetts Institute of Technology, “MIT PGP Key Server.” [Online]. Available: <https://pgp.mit.edu/>
- [46] Microsoft, “Key Management - Microsoft TechNet.” [Online]. Available: <https://technet.microsoft.com/en-us/library/cc961626.aspx>
- [47] P. Brooks and P. Leyland, “Key Revocation - PGP,” Jan. 1995. [Online]. Available: <http://www.ac.uk.pgp.net/pgpnet/secemail/q4/node19.html>
- [48] J. Chow, “Technical Introduction to Events and Logs in Ethereum,” Jun. 2016. [Online]. Available: <https://media.consensys.net/a074d65dd61e>
- [49] Ethereum Foundation, “Web3.js - Ethereum Javascript API.” [Online]. Available: <https://github.com/ethereum/web3.js>
- [50] —, “Ethereum JSON RPC API.” [Online]. Available: <https://github.com/ethereum/wiki/wiki/JSON-RPC>
- [51] “INFURA :: Ethereum Blockchain Infrastructure.” [Online]. Available: <https://infura.io/>

-
- [52] “ethereumjs-tx - A simple module for creating, manipulating and signing ethereum transactions.” [Online]. Available: <https://github.com/ethereumjs/ethereumjs-tx>
- [53] “ethereumjs-util - A collection of utility functions for Ethereum.” [Online]. Available: <https://github.com/ethereumjs/ethereumjs-util>
- [54] “MetaMask - Brings Ethereum to your browser.” [Online]. Available: <https://metamask.io/>
- [55] “testrpc - Fast Ethereum RPC client for testing and development.” [Online]. Available: <https://github.com/ethereumjs/testrpc>
- [56] Ethereum Foundation, “Whisper Protocol - Ethereum Wiki.” [Online]. Available: <https://github.com/ethereum/wiki/wiki/Whisper>
- [57] “BIP-0039 - Mnemonic code for generating deterministic keys.” [Online]. Available: <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>
- [58] “BIP-0032 - Hierarchical Deterministic Wallets.” [Online]. Available: <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>
- [59] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, “Secure Multiparty Computations on Bitcoin,” in *2014 IEEE Symposium on Security and Privacy*, May 2014, pp. 443–458.
- [60] Y. G. Desmedt, “Threshold cryptography,” *European Transactions on Telecommunications*, vol. 5, no. 4, pp. 449–458, Jul. 1994.
- [61] A. Acquisti, A. Friedman, and R. Telang, “Is there a cost to privacy breaches? An event study,” *ICIS 2006 Proceedings*, p. 94, 2006.

-
- [62] “Replay Attacks - Microsoft Developer Network.” [Online]. Available: <https://msdn.microsoft.com/en-us/library/aa738652.aspx>
- [63] A. Merriam, “How can a contract run itself at a later time?” Jan. 2016. [Online]. Available: <https://ethereum.stackexchange.com/a/87>
- [64] J. R. Douceur, “The Sybil Attack,” in *Peer-to-Peer Systems*. Springer, Berlin, Heidelberg, Mar. 2002, pp. 251–260.
- [65] D. Williams, “Byzantine consensus suitable for decentralized networks using cryptographic randomness,” May 2015. [Online]. Available: <https://crypto.stanford.edu/seclab/sem-14-15/williams.html>
- [66] A. Jsang, R. Ismail, and C. Boyd, “A survey of trust and reputation systems for online service provision,” *Decision support systems*, vol. 43, no. 2, pp. 618–644, 2007.
- [67] “Does uPort require Ether? | uPort Support.” [Online]. Available: <http://support.uport.me/frequently-asked-questions/product/does-uport-require-ether>
- [68] “EIP0074 - Support RSA signature verification.” [Online]. Available: <https://github.com/ethereum/EIPs/issues/74>

Appendix A

Contract Source Code

A.1 Identity Contract

```
contract Identity {  
  
    // Contract Storage  
    address owner;  
    string ipfs_hash;  
    address recovery;  
  
    // Ownership Checks  
    modifier onlyOwner(){  
        if (msg.sender == owner) _;  
    }  
    modifier onlyOwnerOrRecovery(){  
        if (msg.sender == owner || msg.sender == recovery) _;  
    }  
  
    // Functions  
    function Identity() {  
        owner = msg.sender;  
        recovery = new Recovery(this);  
    }  
    function setRecovery(address _recovery) onlyOwner {  
        recovery = _recovery;  
    }  
}
```

```

function setIPFSHash(string _ipfs_hash) onlyOwner {
    ipfs_hash = _ipfs_hash;
}
function setContacts(address[] _contacts) onlyOwner {
    Recovery recovery_c = Recovery(recovery);
    recovery_c.setContacts(_contacts);
}
function addRecovery(address _recovery, address _key) onlyOwner {
    Recovery recovery_c = Recovery(_recovery);
    recovery_c.addRecovery(_key);
}
function transferOwner(address _owner) onlyOwnerOrRecovery {
    owner = _owner;
}
function getDetails() returns
    (address _owner, string _ipfs_hash, address _recovery) {
    _owner = owner;
    _ipfs_hash = ipfs_hash;
    _recovery = recovery;
}
}

```

A.2 Recovery Contract

```

contract Recovery {
    address uuid;
    address[] contacts;
    mapping(address => address) recoveries;
    mapping(address => uint) proposed_keys;
    modifier onlyUuid(){
        if (msg.sender == uuid) _;
    }
    modifier onlyContact(){
        for(uint i = 0; i < contacts.length; i++)
            if(contacts[i] == msg.sender) _;
    }

    function Recovery(address _uuid) {
        uuid = _uuid;
    }
}

```

```
function setContacts(address[] _contacts) onlyUuid {
    contacts = _contacts;
}
function getContacts() returns (address[] _contacts) {
    _contacts = contacts;
}
function addRecovery(address _key) onlyContact {
    if(recoveries[msg.sender] != _key
        && proposed_keys[recoveries[msg.sender]] == 0){
        recoveries[msg.sender] = _key;
        proposed_keys[_key] += 1;
    }
    if (proposed_keys[_key] >= (contacts.length / 2)){
        Identity identity_c = Identity(uuid);
        proposed_keys[_key] = 0;
        identity_c.transferOwner(_key);
    }
}
function getRecoveries(address _key)
    returns (uint num_done, uint num_total) {
    num_done = proposed_keys[_key];
    num_total = contacts.length / 2;
}
}
```

A.3 Project Repository

The source code for the proof-of-concept system outlined in this paper can be found on GitHub at <https://github.com/zachd/ethereum-identity-research>.

The final commit hash is 82eeef2c64a2e3dbc0c286c81a8f215d7ee2a8d0.