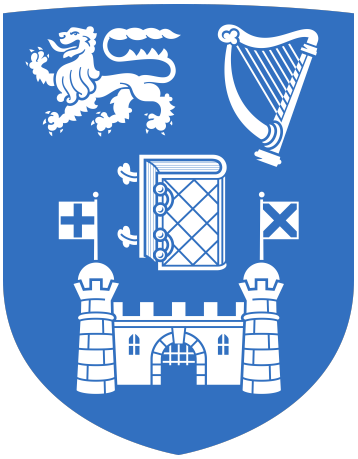
The University of Dublin



Trinity College

Using the Ethereum Blockchain as a General Purpose Identity Mechanism

Gregory Buckley

B.A. (Mod.) Integrated Computer Science

Final Year Project May 2017

Supervisor: Dr. Donal O'Mahony

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland

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.

Gregory Buckley

Date

Permission to lend

I agree that the Library and other agents of the College may lend or copy this report upon request.

Gregory Buckley	Date

Abstract

Over the last number of years, the need for an secure system to maintain a users's digital identity has increased, with an increasing number of services being offered online and through wireless systems. Banking systems, email and social security are moving to a digitalised infrastructure where a reliant and secure identity system is needed. The Public Key Infrastructure allows users to connect their 'identities' to a public key, but establishing a user's key in the first place and dealing with lost and stolen keys are some of the factors which hold back the potential of digital identities.

The introduction of the Ethereum blockchain provides a means for creating decentralised applications, where data can be stored and manipulated securely between users. This project investigates the potential of deploying an identity mechanism using the Ethereum blockchain, by learning the drawbacks from identity mechanisms currently used today, and investigating current developments being made in handling identity on the blockchain.

Acknowledgements

I would firstly like to thank my supervisor, Dr. Donal O'Mahony, for all the help and encouragement he has given me over the course of the project. I would also like to thank all the staff and students at Trinity College, who have made my last four years of studying both challenging and rewarding. I would finally like to thank my parents, for all the help and support they offered throughout my years of education to reach this point.

Contents

Chapter 1

Introduction

1.0.1 Project Motivation

Over the past thirty years, the number of people and businesses online has grown dramatically. In order for the Internet to be successful as a medium for communication, it is vital to have secure and authenticated communication, where users can be assured of the identity of who they are in communication with.

Identity mechanisms have been created as a means to establish a digital online identities for a user. Achievement of this concept is reflected by its implementation in third party certificate authorities [1] and the Pretty Good Privacy [2] design, which makes use of the Web Of Trust, all of which are discussed later in this dissertation. Nevertheless, while these identity mechanisms are frequently used, they both possess faults in their viability for a solution towards security in network communication. Central authorites rely on complete trust in a third party to be us, which can lead to a single point of failure in the system if the authority is breached. Pretty Good Privacy contrasts by offering a decentralised approach, but its system in creating a trust network between user's can be ineffective, as discussed later in the report.

The introduction of the Blockchain in the past ten years has sparked interest as a means

of creating a distributed, decentralised database which is protected from malicious attacks. The rise in successful cryptocurrencies, such as Bitcoin, has proven that the technology can be successfully used to solve some of the worlds biggest technological problems such as creating a secure, instant method of transferring currency online. New identity mechanisms have been designed on the Bitcoin Blockchain to take advantage of the decentralised design, but most have had shortcomings due to the fact that the Bitcoin Blockchain was designed initially with only digital currency in mind.

Ethereum has been introduced in the past two years, with the goal to surpass all the existing limitations of the Bitcoin Blockchain with a design made with the intention of creating decentralised applications. This dissertation looks into the background of the existing identity mechanisms used today, the proposals of using the blockchain as an alternative identity mechanism design and investigating whether the Ethereum blockchain has the potential to create a viable solution for the future.

1.0.2 Project Aims

With the currenent mechanisms used to handle digital identity being flawed, I hope to create a more viable solution by taking advantage of the recently developed Ethereum Blockchain.

The aims of this project are as following:

- To study the behaviour of identity mechanisms currently in place.
- To study the Bitcoin blockchain and the introduction of Ethereum.
- To investigate and analyse current approaches for identity management on the blockchain.
- To design and implement an improved solution for the creation of a decentralised approach to issue and store user identities on the Ethereum blockchain.
- To evaluate and propose improvements for this solution.

Chapter 2

Background

This chapter introduces the techniques used in cryptography, which is a vital componant to how identity mechanisms operate as it manages the encryption and authentication of data. Certificate authorities and Pretty Good Privacy will be investigated as they are the most commonly used mechanisms for managing digital identity today. Finally, the background and architecture of the Ethereum blockchain will be discussed, so that it can later to investigated to see if it may be used to create a viable identity solution for the future.

2.1 Cryptography

Cryptography serves as the basis for which data can be encrypted and validated over a network [3]. The two most popular encyption methods are Symmetric Encryption and Asymmetric Encryption, which are used in the authentication of user identities and secure transmission of messages. This chapter describes how these encryption methods are implemented, how hash functions are used to manipulate data to a fixed size, and how digital signatures are used for authenticating messages and in identity certificates as a means of authenticating user identities.

2.1.1 Symmetric Encryption

Symmetric encryption involves the establishment of a shared secret key between two parties in order to encrypt data. The same secret key is used for decryption of the message. Prior to establishing this key, a secure channel is necessary for both parties [4].

2.1.2 Asymmetric Encryption

Asymmetric encryption entails the use of two different, but mathematically related keys. Each party creates a public key, as well as a private key. The sender encrypts the message with their private key, and the receiver may decrypt the message by using the public key of the sender. The involved parties in the process must have knowledge of each other's public keys.

In the case of the distribution of private keys in symmetric encryption, a secure channel is required. Whereas in asymmetric encryption, only an authenticated channel is necessary, as the public distribution of a user's public key cannot lead to an attack to impersonate another user [4].

2.1.3 Hash Functions

A hash function is a function that can be used to map data of arbitrary size to data of a fixed size. Hash functions are commonly used in cryptography due to it being a one way function which cannot be reversed. This makes hashing suitable for password validation as the hash of a password can be checked against the hash of a user's input, removing the obstacle of storing passwords in plain text. Similarly, authentication and digital signatures can profit from this method of data manipulation. An ideal hash function is contains a number of properties:

- Quick computation of a hash value for any given message.
- Infeasible to reverse a message from the hash value.
- Small changes in a message should change the hash value
- Unlikely for two messages to have the same hash value.

Many hashing methods exist, but the SHA-256 [5] algorithm is one of the strongest and most commonly used hash functions, and can be used in Blockchain technologies and signature applications.

2.1.4 Digital Signatures

A digital signature is a numeric value produced to demonstrate the authenticity and integrity of a digital message [6]. Digital signatures make use of asymmetric cryptography and hashing. They are very frequently used in digital identity mechanisms, as they provide a method for parties to attest to the identity of a user. For an entity to produce a digital signature for another user, the entity can combine their own private key with the hash of all the fields in user's certificate.

For example; a user, Alice, requires a digital signature for her authentication in a network, which can be produced by user Bob. For Bob to create a signature for Alice, Bob can encrypt the message "Alice's public key is 0x123456789" using his private key, allowing Alice to store this message alongside her public key. Via decryption of the signature with Bob's public key, Alice is authenticated in the network by other users in receipt of her public key.

2.2 Certificate Authorities

Certificate authories are the most common method used today for the authentication of web domains and user identities. This section looks into how this mechanism is implemented, and the shortcomings which exist with it.

2.2.1 Background of Certificate Authorities

Certificate Authorities (CA) are third parties, used for validating the authenticity of public keys. A certificate stores a user's public key. A certificate authority is responsible for the issuing, signing and verifying of these certificates. For a user to authenticate themselves, a request for a certificate is made to a CA, which is signed by the CA's private key. If another user receives a message containing this certificate, the certificate authority's signature is used to verify the authenticity of the sender of the message. For this process to be achieved, both parties need to trust the third party *and* have a copy of their public key. [1]

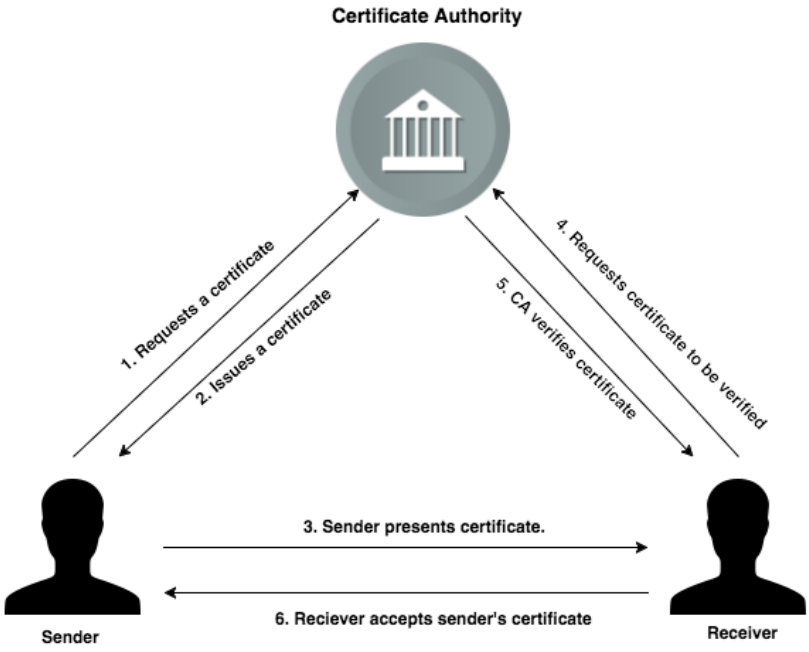


Figure 2.1: Overview of the interaction with a certificate authority.

Public key certificates are data structures, which relates public key values to users. This relation is created by having a trusted CA digitally sign each certificate. The structure for a X.509 V3 certificate [1] is specified below:

Figure 2.2: Overview of the build up of a X.509 certificate.

2.2.2 Issuing of Certificates

The method by which a CA verifies a user's identity, prior to signing their certificate, depends on the security of the certificate policy used. Many policies exist, with the most frequently used being Domain Validation (DV) or Extended Validation (EV).

Domain Validated Certificates

A Domain Validated (DV) certificate involves a user requesting a certificate, with the provision of their email, name and domain [7]. The CA performs a `whois` lookup on the

domain to check whether a user's given name and email match the registered information. Additionally, the CA may send an email to the address provided to confirm it is controlled by the user in question. It is possible for this process to be completely automated, and such implementations have arose such as ”Let's Encrypt, ”Comodo” and ”Verisign”, which offer DV certificates for free [8].

Extended Validated Certificates

An Extended Validated (EV) certificate requires a more detailed background search of the requesting user, and requires human interaction [7]. EV may require a phone call, signed documents and a face-to-face meeting with the CA to verify a user's identity. The CA would confirm that the entity owns, or has rights to use to the domain, which may be included in the certificate. The cost for receiving a certificate rises with a more thorough validation process by the CA.

2.2.3 Key Revocation

Each certificate has a *validation period* for when it can be used. Loss of private key, change in users status or key compromisation may lead to the certificate being revoked by a CA. A certificate authority is responsible when a certificate is compromised. There are several methods such as 'Online Certificate Status Protocol' (OCSP) [9] and Certificate Revocation Lists (CRLs) [1] which are used to declare the revocation status of a certificate. CRLs are the most common approach.

A CRL is a list identifying all certificates which have been revoked by a CA. When a user interacts network to identify other users, the user does not only check the certificate signature and validity, but also checks the CRL to ensure that the certificate in question has not been revoked. The CRL is updated periodically by the CA. Once a certificate is revoked, it appears on the updated CRL. To contain this growth of the CRL, the certificate remains on the CRL until the certificate expires.

2.2.4 Key Recovery

In the event that a user loses their private key, potentially due to loss of device possessing the key or accidental deletion, the user loses access to encrypt and decrypt new and old messages using their private key. The user has the option to allow the CA to backup their private key. This leads to a scheme where a user may later recover their lost key from the CA. While this is achievable, it is not commonly used as this leaves the possibility of the CA being able to impersonate the user or even be falsely accused of doing so.

2.2.5 Problems with Certificate Authorities

The trusting of a certificate authority is vital. However, there is no reason for one to believe that a CA is trust-worthy, other than the CA's ability to sign user certificates. This leads to a large number of CAs in industry, and it becomes increasingly difficult to identify the trustworthiness of each CA which a user encounters.

Lack of Assurance in Certificate Quality

A more trustworthy CA requires more extensive background checks before issuing a certificate. This does not prevent users from sacrificing trust for cost by acquiring certificates from more inexpensive authorities which have the same functionality. With the cost of creating a certificate, in the perspective of a CA, being low, many companies today issue certificates in large quantities at a low price [10]. This lack of security standard leads to a great imbalance of quality in the certificate ecosystem.

Problems of Authentication

Certificates generally associate public keys with a user's name. If one user receives a certificate from another user, named John Smith, and the CA confirms the validity of the certificate, this does not ensure that the John Smith named in the certificate is in fact the

John Smith which the user knows. Further information about a user may be stored on the certificate to differentiate this John Smith from others on the CA, but it may be likely that the recipient is unsure of this information about the person.

Certificate Validity Period

As discussed, all certificates must contain a validity period, which once expired, loses all of its uses. This is implemented to limit growth size of CRLs and to increase security by limiting the time that a certificate may be used by an attacker. Unfortunately this becomes abused by CAs who take advantage of the need to reissue a certificate to clients and charge additional fees.

Single Point of Failure

One fundamental issue regarding the use of certificate authorities, is that the centralised nature of the design leads to a single point of failure at the certificate authority. This leads to total trust and dependence resting on the stability and credibility of the CA [10].

Attacks have occurred, with the highest profile attack occurring in June 2011 [11]. DigiNotar, was a large Dutch certificate authority which was a primary certificate provider for domains owned by the Dutch government. After a breach, an attacker was able to issue more than 500 fraudulent digital certificates on behalf of DigiNotar. The certificates issued were for some of the top Internet companies such as Mozilla, Skype and Google. An attack of this nature led to the attacker being able to use these certificates on their own sites. This would create the risk of users believing they are communicating with Google, when actually they are not. DigiNotar lost all credibility following this attack and being unable to recover filed for bankruptcy two months later.

2.3 Pretty Good Privacy and the Web of Trust

Pretty Good Privacy (PGP) was first introduced in 1991 by Phil Zimmermann has become a standard for email security [12]. The goal of PGP was to bring cryptography to the masses of people using the Internet in an accessible manner.

2.3.1 Confidentiality

PGP combines both symmetric encryption and asymmetric encryption to allow messages to be sent between users confidentially. The sender of a message creates a random symmetric key, and uses the symmetric key to encrypt the message to be sent. The sender then encrypts the key with the public key of the receiver. After both the encrypted message and key are sent to the receiver, the key can be decrypted with the receiver's private key, and the message decrypted by the symmetric key. [2]

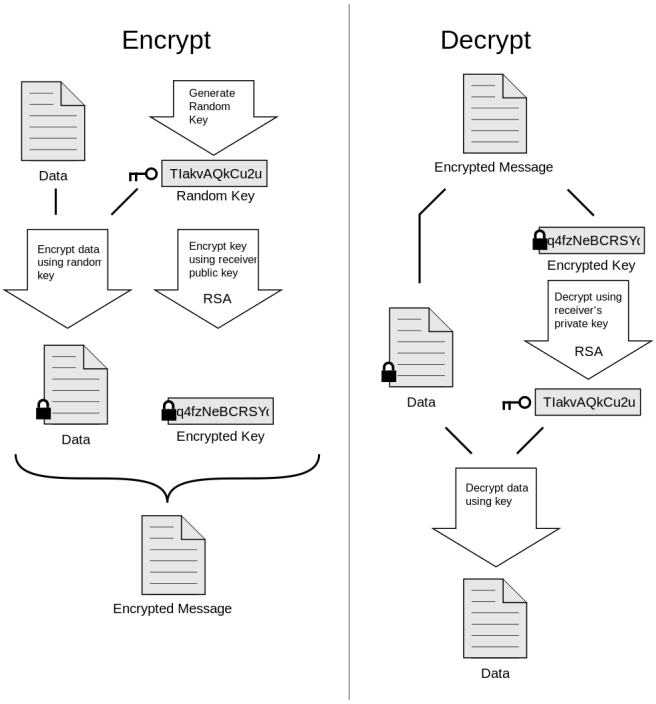


Figure 2.3: The process of encryption/decryption using PGP [13].

2.3.2 The Web of Trust

In contrast to the centralised approach of certificate authorities, PGP offers an alternative design by opting for a ”Web of Trust” architecture [14]. The central authority is removed by shifting authentication to a network of users. These users possess the ability to sign other users' public keys, and view the signatures which other users have received. This builds a web of individual public keys, connected by links formed by these signatures.

Paul Zimmerman describes how the Web of Trust may be used in the manual for PGP version 2.0 in 1992:

As time goes on, you will accumulate keys from other people that you may want to designate as trusted introducers. Everyone else will each choose their own trusted introducers. And everyone will gradually accumulate and distribute with their key a collection of certifying signatures from other people, with the expectation that anyone receiving it will trust at least one or two of the signatures. This will cause the emergence of a decentralized fault-tolerant web of confidence for all public keys. [15]

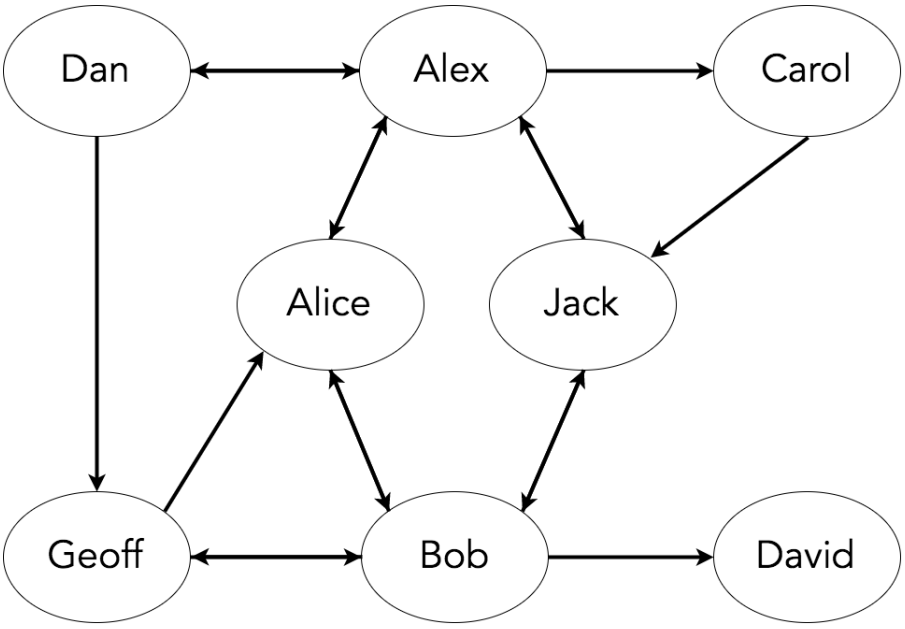


Figure 2.4: Multiple trust connections creates a mesh network between all users.

2.3.3 Key Signing Parties

A key signing party [16] is an event where users can meet in person, to present their public key to others, and to have their key signed once the other person is confident that the key is authentic and belongs to that person. As the PGP infrastructure does not depend on a certificate authority to provide background checks to authenticate users, signing parties act as a solution for users to become authenticated. The presenting of adequate identity documentation is required by participants at a key signing party prior to having their key signed.

2.3.4 Establishing Trust

The figure below shows Alice who is looking to communicate with Bob for the first time. Bob has had his colleague, Jack, sign his certificate which Jack knows is authentic. Alice knows Jack, and trusts Jack as an introducer. Alice can see that Jack has signed Bob's certificate, therefore Alice can assume that Bob's certificate is authentic. It is possible to have multiple signatures on a certificate, which may help in proving a certificate to be more trustworthy. However, had Alice had no 'introducer' to Bob, it would be difficult for Alice to prove the authenticity of Bob.

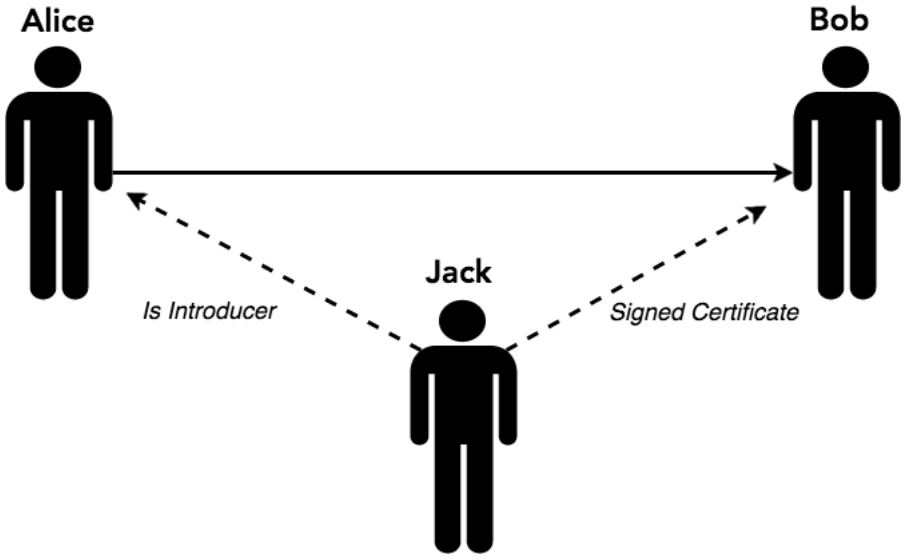


Figure 2.5: Following signing Bob's key, Jack takes position as an introducer for Alice to authenticate Bob.

2.3.5 Trustworthiness of Introducers

To allow the system to become more reliable in determining the trustworthiness of users, PGP allows for the rating of introducers to be split into three categories depending on the confidence attached to an introducer.

- *Full*: Full trust is one which you are fully confident in the introducer to only sign keys which are definitely trustworthy.
- *Marginal*: Marginal trust entails that the introducer may be able to introduce you to a new user, but you may not be confident to always be trustworthy.
- *Untrustworthy*: Untrustworthy introducers cannot be trusted in any case to introduce to new users.

Categorisation of a user into one of these categories is not standardised, and how to group an introducer into one of the categories is left entirely to the user themselves. PGP docu-

mentation does however offer guidelines to assist users in the vetting of introducers [17], however this is still a poor substitute for the CA security and certificate policies.

2.3.6 Trustworthiness of Users

Public keys themselves can be split into three categories in regards to their trustworthiness:

- *Undefined*: It is not possible to tell whether the key is valid or not.
- *Marginal*: The public key may be valid, but it is uncertain.
- *Complete*: The public key is valid, and it is certain to be authentic.

With different levels of confidence placed on users, the Web of Trust offers a mechanism for setting a number of both marginal trusts needed and complete trusts needed before a public key can become valid [17].

2.3.7 Problems with the Web of Trust

PGP and the Web of Trust show a successful method to encrypt messages through email and various other applications. The decentralised architecture and removal of third party certificate authorities remove the single point of failures, reliance of trust on a CA and necessity for a valid lifetime of a public key [14]. However, while it may not have the same shortcomings of the CA architecture, PGP and the Web of Trust has flaws of its own.

Key Loss

While with Certificate Authorities, the loss of a user's private key is not detrimental due to the ability for the Certificate Authority to securely backup a user's key leading to no permanent loss of access to their certificate. Due to the decentralised nature of PGP, no key recoverability exists. If a user loses access to their private key, then the user must

generate a brand new key pair. Following key loss within the Web of Trust, a user loses all signatures from their previous key and must build a new reputation from scratch.

Furthermore, loss of a private key prevents a user from being able to decrypt new messages and most email clients only store old messages in their encrypted form, so a user loses access to their old messages as well.

Rating of a User's Trustworthiness

The ambiguity of the nature of rating trust on users leads to an imbalance between the varying views which different users may perceive in rating the trustworthiness of a public key or an introducer.

Entry Barrier For New Users

One of the largest social obstacles in the design is in the large entry barrier which new users face. Upon creation of a new key pair, not having any signature on one's public key likely leads to not being readily trusted by other user's systems which may require a certain number of introducers before authenticating a user. The user must meet other users in person to authenticate themselves which may not always be possible. If a user is abroad and may never have an opportunity to meet those which they know, then it is unlikely that they will ever be able to become trusted.

2.4 The Blockchain

2.4.1 The History of the Blockchain

The Blockchain is shared database made up of a continuously growing list of records known as *blocks*. The Bitcoin Blockchain was proposed by Satoshi Nakamoto in 2008 and has proven to be the most successful implementation of the blockchain to date [18]. The Bitcoin blockchain provides a decentralised system with secure transactions of money, immune from the *double spending problem* [19].

2.4.2 Decentralisation on the Blockchain

The decentralised design proposed by Blockchain implies that all records on the chain may be downloaded and shared across many computers. Each block contains its own hash value, the hash value of the block prior to it and a batch of transactions which are hashed and stored within a Merkel tree [20]. The Merkel tree is the structure in a block which provides cryptographic proof of the transactions in the block and their order.

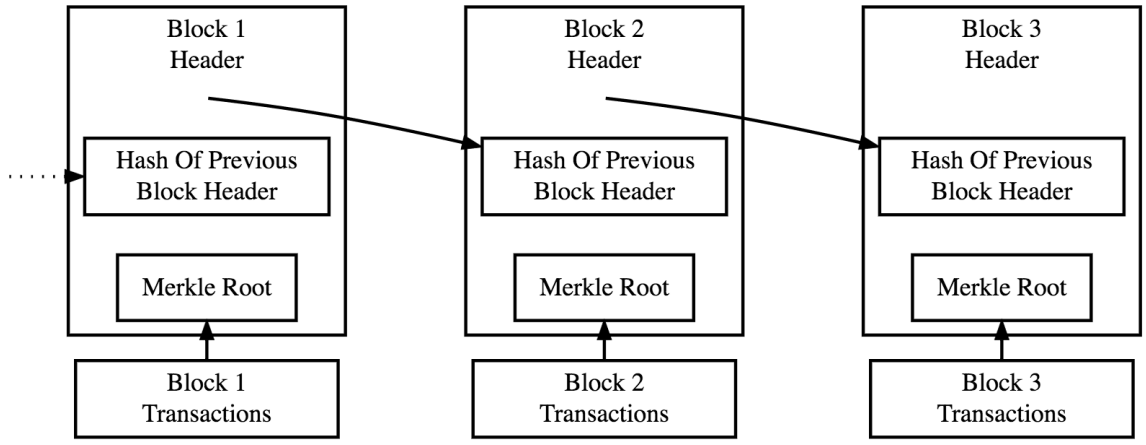


Figure 2.6: Overview of the connection between blocks on the blockchain.

The connection of a block to the previous block in the chain allows for the assurance of the integrity of each block which can be traced back to the original genesis block. Therefore, once data has been recorded in the blockchain, the data within a block cannot be altered as the block and every block in the history of the blockchain would have to be altered on all computers in the network. This makes the blockchain resistant to malicious attacks which would be possible in a centralised storage of data [21].

2.4.3 Addition of transactions to the Blockchain

A transaction must be verified before its addition to the blockchain in a process known as *mining* [22]. Bitcoin makes use of mining to assure that transactions are not fraudulent and that individual bitcoins are not spent more than once (double spending) [19]. A consensus on the network must be reached in order to agree on the order of transactions on a newly mined block. While an ideal scenario would be for nodes to each vote on the order of transaction, this is not possible due to the potential of the Sybil attack [23], whereby a single entity joins with multiple nodes to achieve multiple votes to influence the network to favour their own interests.

2.4.4 The Proof-Of-Work Scheme

This problem of generating a consensus for the order of transactions is solved by making the process of mining computationally expensive. Miners work to complete a proof-of-work algorithm in hope of earning Bitcoins upon success. In the proof-of-work algorithm, miners work to calculate a value which once hashed, with a hashing algorithm such as SHA-256 [5], generates a number beginning with k zero bits. If the number of zero bits to be calculated is low, then the computation time to complete is quick and may only take several seconds. However, as the number of zeros increase, as does the time and computational power required to complete the puzzle. Satoshi Nakamoto explains that the number of lead-

ing bits to calculate differs to compensate for the increasing hardware speed and number of miners on the network over time. The difficulty is determined by aiming to achieve a set number of blocks mined each hour. If the blocks are being successfully mined too quickly, then the difficulty of the puzzle increases. [18]

Once a proof-of-work calculation is solved, the successful miner distributes the solution to all nodes on the network to prove the computational effort given to find a solution. Due to the nature of the hashing algorithm, while computationally difficult to solve, the solution achieved can be very easily verified. Once other miners on the network verify the authenticity of the solution, the block and all of its transactions, are added as the newest block on the blockchain, and miners move to work on the next block.

Once more than 51% of the nodes on the network are honest, then the chain is immune to attack. Therefore, this system is in place where authentic and secure transactions are added to the blockchain without any trust needed between nodes. [23]

2.4.5 Limitations Of The Bitcoin Blockchain

The Bitcoin blockchain has been implemented by various groups for uses other than currency exchange. The two methods of building such an application are either to construct an independent network of mining nodes *or* to construct a protocol on top of the Bitcoin blockchain.

Independent network construction

The building of an independent network has been achieved on such applications such as Namecoin [24]. This choice has been moderately successful although difficult to implement. The difficulty in this being a worthwhile choice is that applications who chose this route need to create an independent blockchain and test and build all necessary state transition and network code. Most applications are generally too small to warrant their own

blockchain, and therefore with a relatively low node count on the network they are vulnerable to the Sybil attack [23].

Protocol construction

The alternative choice to build on top of the Bitcoin blockchain too has flaws. Since the bitcoin application uses the depth of a block as validation that the block is valid, it may take too long for newer applications which need faster assurance. By sharing the same blockchain, the new applications cannot exclude the other transactions within the blockchain from sight which likely flood the chain and are unnecessary to them. [25]

Lack of State

Many applications require multiple stages of execution (i.e. states) to complete complex tasks. Bitcoin provides minimal scripting for applications, and transactions on an application can only return two values: 1 if the verification of a transaction is successful and 0 otherwise [25]. In the bitcoin blockchain, coins can either be spent or unspent and no capability exists for scripts to contain any other internal state beyond that. This makes the capabilities for more Turing-complete applications limited where multi-stage contracts cannot be created which makes many applications impossible to build [25].

2.5 The Ethereum Blockchain

2.5.1 History of Ethereum

Ethereum is a new blockchain introduced in late 2013 by Vitalik Buterin [25]. The goal of Ethereum is to aid the creation of decentralised applications on top of the one blockchain with rapid development time and high security. This is achieved by the creation of a more robust scripting language than Bitcoin, which is built on top of the Ethereum blockchain. With this new scripting capabilities, users can quickly deploy applications and create smart contracts with much more ease than with Bitcoin thanks to the Turing-completeness, blockchain awareness and state control of Ethereum's scripting.

2.5.2 Ethereum Accounts

In Ethereum, state is maintained by the inclusion of accounts. Each account is referenced by a 20-byte public address and state is handled by the transfer of currency and information between accounts. An Ethereum account is comprised of the account's current Ether balance (Ether being the currency for Ethereum), contract code (if present) and account storage [25].

There are two types of accounts in Ethereum:

- *Externally owned account*: which are controlled by a private key.
- *Contract account*: which are controlled by its contract code.

2.5.3 Smart Contracts

Contract accounts are a fundamental aspect of Ethereum which aid in the creation of smart contracts. These smart contracts are key in the creation of decentralised applications. Any time a contract account receives a message, it's code is run, allowing the account to read

and write to its internal storage and send messages to other contract or externally owned accounts.

A well written smart contract describes all possible outcomes of that contract. Smart contracts are deterministic, and yield the same output for the same input. Since the contract resides on the blockchain, the code can be inspected by anyone on the network and since all transactions on the blockchain are signed, all participants have a cryptographically verifiable trace of the actions of the contract [21].

Further benefits of the code being on the blockchain are that users may inspect the code and identify its outcomes prior to engaging with the contact. They have certainty of execution since no one entity has full control of the contract, and all processes are verifiable since all interactions are digitally signed.

An example of the potential of a smart contract can be seen in Ethereum based applications, such as Slock.it [26], which offers a decentralised apartment renting application. This application uses smart contracts to allow one party to send Ether to another, in return for the user's public key gaining access to an apartment for a defined time period, specified in the contract via a NFC device and a smart door lock connected to the contract.

2.5.4 Transactions in Ethereum

The term ”transaction” is used in Ethereum to refer to the signed data package that stores a message to be sent from an externally owned account. The figure below shows the structure of an Ethereum transaction.

Figure 2.7: The fields which make up the structure of an Ethereum transaction.

The first three are standard fields expected in any cryptocurrency. The data field is optional, but can be used to pass data as parameters to a contract. The term ”gas” is used as a method of payment for running transactions in Ethereum. The more computational steps and storage required by a transaction, the higher the gas cost to run it.

This mechanism is used to force attackers to pay for every resource that they consume. The value of gas is then converted to Ether, which is taken out from the account’s ether balance. Transactions to read from a contract however, are free and cost no ether.

The `STARTGAS` field of the transaction shown above is used to prevent accidental or hostile infinite loops. Each transaction sets a limit to the maximum gas which it can use, so that a user does not lose their total ether balance. If a user interacts with a contract, which in turn forwards a transaction to another, then the `STARTGAS` value specified is also forwarded on to the next transaction. If a transaction runs out of gas, then no state change occurs, other than miners being rewarded in Ether [27].

Chapter 3

State-of-the-Art

Following the problems that result from a centralised design of using certificate authorities for identity management, the blockchain offers the potential to move to a decentralised architecture.

Several groups have proposed successful solutions built on both the Bitcoin and Ethereum blockchain, such as Namecoin and Uport.

3.1 Namecoin

Namecoin was first introduced 2011 [24], and proposed a decentralised DNS registry for `.bit` web domains. Namecoin was the first coin to fork the Bitcoin blockchain to create a new digital currency to take advantage of Bitcoin's features, but to also build on top features for domain registration.

The advantages brought by registering a `.bit` domain includes the removal of the need to register with a third party, which requires high payment and the potential problems brought on by governments, as it a makes the removal or seizure of a domain name not possible.

Problems

However, a number of issues lie in the design of the Namecoin architecture. Firstly, Namecoin offers no method of key recovery after the loss of private key. Since the design only allows new domain names being registered to be unique, an issue arose whereby attackers claimed thousands of domain names including popular names such as Google.bit and Yahoo.bit [28]. Since Namecoin made the cost of registration so low, as of 2015, it had 120,000 domain names registered with only 28 containing non-trivial content [28].

Namecoin also stands vulnerable to a Sybil attack due to its creation of a standalone blockchain apart from Bitcoin, with a relatively low amount of miners. Namecoin, similarly to Bitcoin, has been threatened by large entities attempting to claim 51% of the network in the past, which would prove fatal to a domain registry with a focus to escape the censorship of large entity control [24].

3.2 Uport

Uport is a system for creating self-sovereign identities on the Ethereum blockchain [29]. This application allows users to own and control their personal identity, reputation, data, and digital assets securely. Uport takes advantage of the additional capabilities that Ethereum offers, most importantly in the inclusion of Smart Contracts. The core of the user's identity is their uPort identity, which is a 20-byte hexadecimal string. This identifier is defined as the address of a smart contract known as the Proxy contract. This Proxy contract can forward on transactions that are sent to it, to interact with other smart contracts on the Ethereum blockchain.

Controller Contract

The most noteworthy aspect of the Uport design is the creation of the Controller contract [29]. The Controller contract maintains access control to a user's Proxy contract.

Each Proxy contract in the system contains an ”owner” value, representing the Controller contract which has access to the Uport identifier. Using their private key, users use the Controller contract to authenticate themselves to the Proxy contract, and have it act on their behalf to other applications.

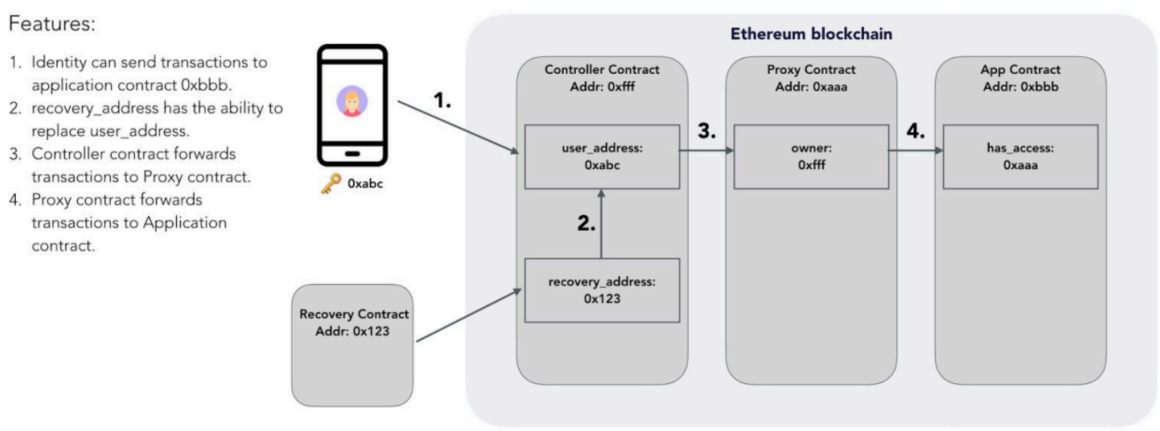


Figure 3.1: The Controller contract is used to forward user interactions to the Proxy contract and onwards to applications.

Key Recovery

This design and the introduction of a third contract, the Recovery Contract, introduces the capability of key recovery. If a user loses their private key, then they are unable to authenticate themselves to the Controller contract and subsequently lose access to their Uport identity. By predefining delegates, who can be trusted friends or family of a user, to a controller contract while control is still maintained, then a recovery system can exist whereby the trusted recovery delegates can specify a new user address for the Controller contract. This mechanism allows a user to retain the same Uport identifier over time [29].

Mobile Application

Uport uses a mobile application to store a user's private key securely. The key cannot be exported off the device. As a user wishes to use their Uport identifier to sign an interaction with an desktop application, Uport issues a QR code in the desktop application which can be scanned with the user's camera and confirmed using their fingerprint on the phone.

Uport does not store any user information on the Proxy Contract as Uport is designed to store information in off-chain data stores. It is here that data can be encrypted and stored, enabling a user to store data such as bank details and other private information.

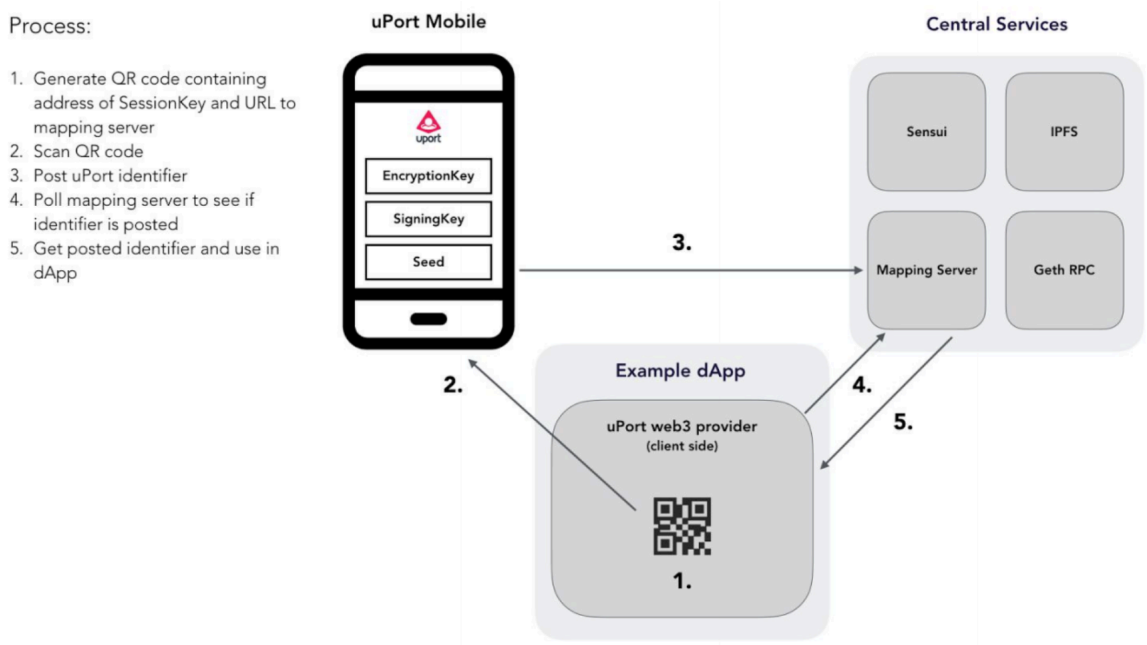


Figure 3.2: An overview of the processes of interacting using the Uport mobile application.

An off-chain server, known as the ”Chasqui” server [29], is responsible for a QR code generation to connect the Uport identifier with desktop applications, however data sent to this server is temporarily not encrypted which may pose as a privacy concern.

Problems

Uport successfully manages key recovery and creates a viable solution for using an Ethereum identifier to interact with other applications. However, the design does not remove all aspects of centralisation, as it relies on communication with a Uport controlled server and an off-chain data storage, to store user's details. This can be seen as a single point of failure, if corrupted.

Another problem in Uport's design is that new users in the system have their gas fees paid for them by Uport, to create and set up an account. This opens a potential for an attacker to flood the Uport ecosystem with multiple fake accounts.

Chapter 4

Design

The solution proposed by this dissertation is to build a general purpose identity mechanism on the Ethereum blockchain. Following the research into how identity is currently managed by the means of the issuing and verification of certificates by trusted third parties, it is clear that this poses too many flaws to become a reliable and sound solution for the future.

The introduction of the Blockchain provides a method to create a decentralised application where no trusted third party is in control. As discussed, Namecoin was the first application to make a valuable attempt at creating a new solution using the Bitcoin Blockchain, but it is the introduction of smart contracts featured on the Ethereum blockchain that can surpass previous designs. Uport has created a design which take advantage of the added features which Ethereum provides by offering recovery from key loss and expanded storage capabilities of user information. Uport however only serves as an identity address, storing no information on the contract, therefore it relies on off-chain data storage.

The design proposed in this dissertation aims to learn from these existing designs, and further enhance them. A final solution aims to contain the following features:

- Remove all need for trust in a single entity.
- Contain no single points of failure.

- Maintain identity control after key loss.
- Introduce a Web of Trust mechanism.
- Usability for authentication in everyday applications such as email.
- Remove any need for off-chain servers.

4.1 Overview

The system is comprised of a user interaction between a link of three contracts, the Controller contract, the Identity contract and the Recovery contract. Figure 4.1 illustrates this architecture below. The system aims to be completely reliable on storage and computation on the blockchain, and not need external centralised servers or applications to authenticate and store certificates.

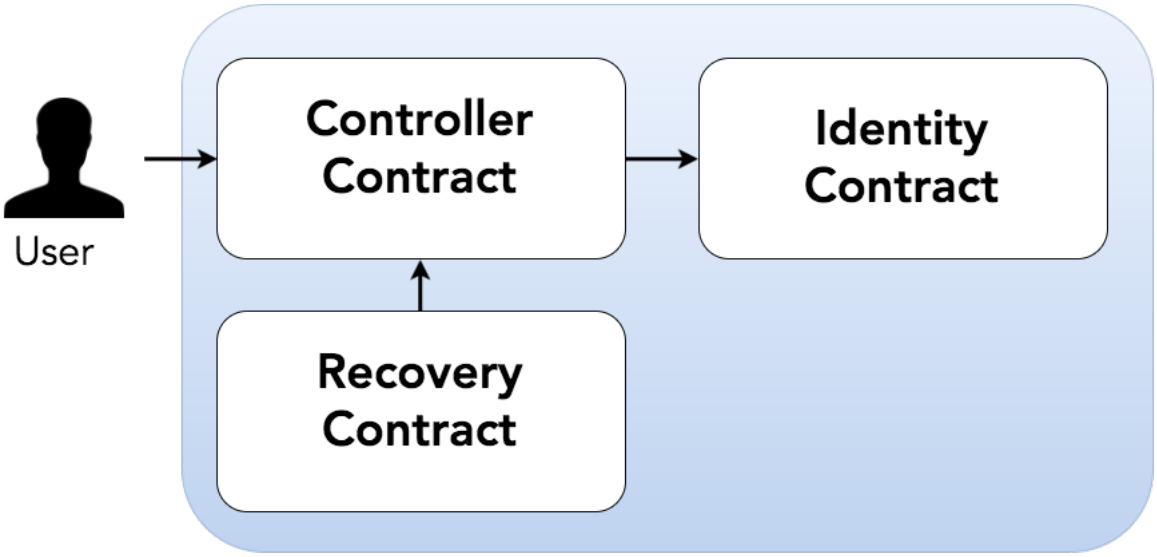


Figure 4.1: The overview architecture of the design.

The following sections will go into detail the functionality of each of these contracts, and the reasons for why this design was chosen.

4.2 Creation of Certificate

The core of the design is the Identity contract, which stores both the certificates of users and the email log, which will be discussed later. The system offers an approach to creating self-sovereign identities, where users may create certificates by themselves without the need of a third party authority. An overview of the structure of certificates stored on the Identity Contract is shown in Figure 4.2.

A hash table structure is a list containing all user certificates within the system.

Figure 4.2: Certificate contain the information of a user on the system.

A user creates a certificate which is assigned a unique public ID by the contract. The ID is a reference to the certificate which can be used by sent to other users. With Ethereum's extended capabilities for storage within a contract, a user may add additional details such as their name, company and any other details which they would like to share or aid them to be more recognisable to other users.

4.3 Key Loss Handling

Loss or theft of a user's private key has proven to be a major flaw in designs which use certificate authorities and PGP. Uport introduces the idea of assigning trusted delegates for a user's identity. This feature is included in this design with the inclusion of the Controller contract and Recovery contract.

Controller Contract

All user interactions within the system are sent through the Controller contract. Each certificate is owned not by the user's Ethereum address, but by a Controller contract address which the user owns. The reason for this is that if a user loses their private key, then it is possible for them to generate a new Ethereum account and create a new Controller contract without having to create a new certificate. In this event, the recovery mechanism begins and the ownership of a certificate moves from the old Controller contract to the new Controller contract. Having the controller contract separate further allows the Identity Contract to stay persistent over time while new more updated versions of the controller Contract can be introduced.

Moving Certificate Ownership to a New Controller

For a user to regain ownership of a certificate, control needs to be allocated in an authenticated manner to the new Controller contract. The Recovery contract contains a list of trusted delegates assigned by the user. These delegates can specify the address for a new Controller contract to have access to the user's certificate. After a user generates a new recovery contract, the user may display the address of this contract to their delegates. The most secure way would to meet in person to show this address. This delegate system removes any capability of attack, as an attacker would have to secure all the private keys of the delegates to take control of the identity. The delegates can send a message to the old

controller contract, informing of a need for the certificate to switch to a new controller. Once a majority of the trusted delegates vote on the same address, then the old Controller contract changes the owner value on the certificate to be that of the new Controller contract. The recovery contract too then points to this new c Controller contract. The user now has full access to their certificate.

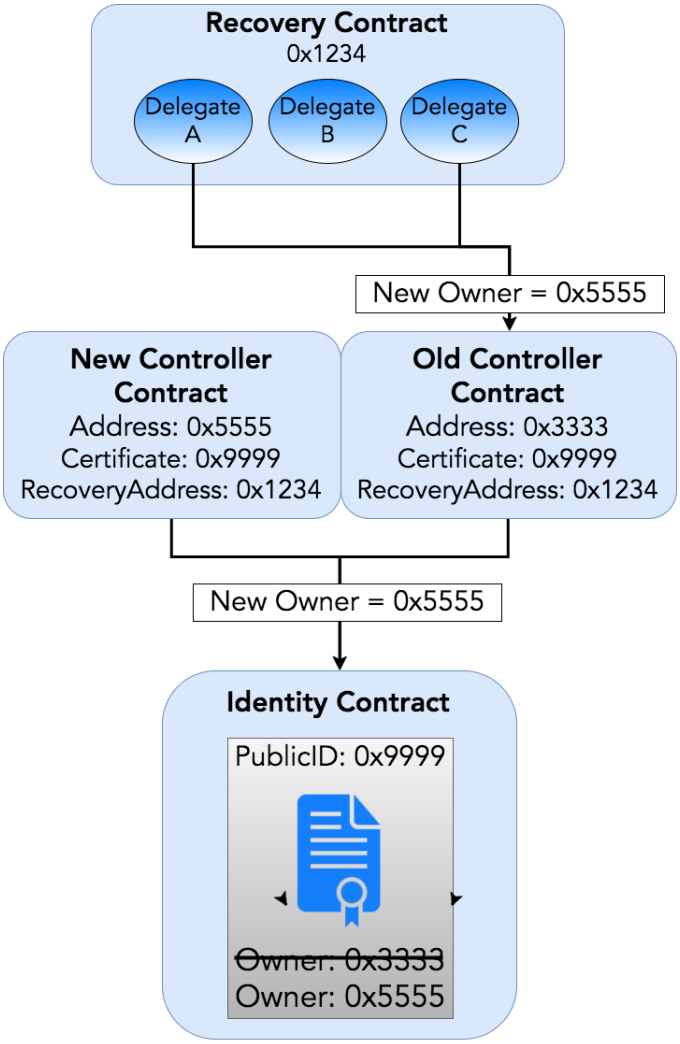


Figure 4.3: Overview of the processes in assigning a new controller to a certificate.

4.4 Trust Network

The introduction of the Web of Trust in PGP created a network where users use introducers to validate authenticity of other users. The design proposed here aims to take advantage of this ideology and allow users using the application to build their own trust network.

Keychain

The first feature which allows this is the creation of a 'keychain' on a user's certificate. This keychain is a hash table stored within the certificate where user's may add the public ID's of other certificates which they trust on the network.

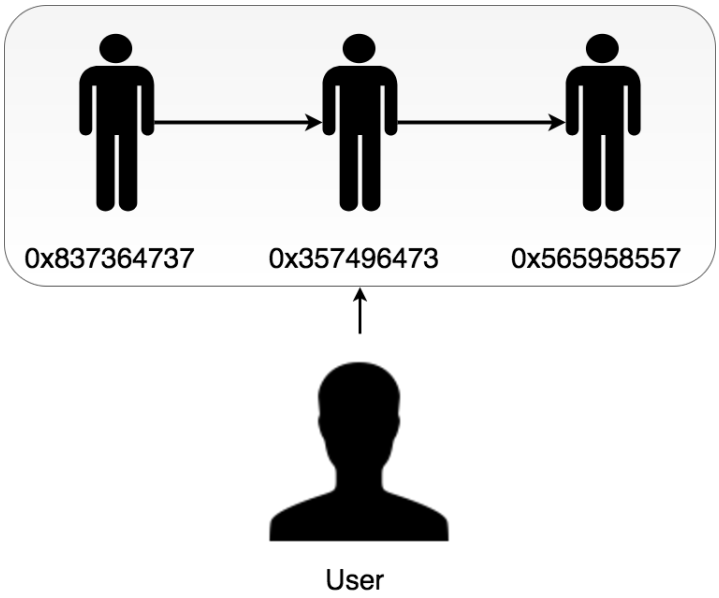


Figure 4.4: A keychain structure allows users to store the public IDs of other users that they trust.

Signature Storage

Another structure stored in a user's certificate contains a list of public IDs which a user has placed trust. This is similar to how in PGP one could view the signatures stored on a

another user's certificate. This feature allows users to view the mutual connections which they have with another user and then be able to decide whether to believe their authenticity. Since a user passes transactions via their private key and through the controller contract, it makes it cryptographically impossible to be able to forge a signature of another user. This yields high potential since a user will not lose access to their Identity contract, a larger trust network can be created without risk of having to to begin gathering signatures again. A user may gather signatures from banks, government and other well known entities to further strengthen their certificate's reputation.

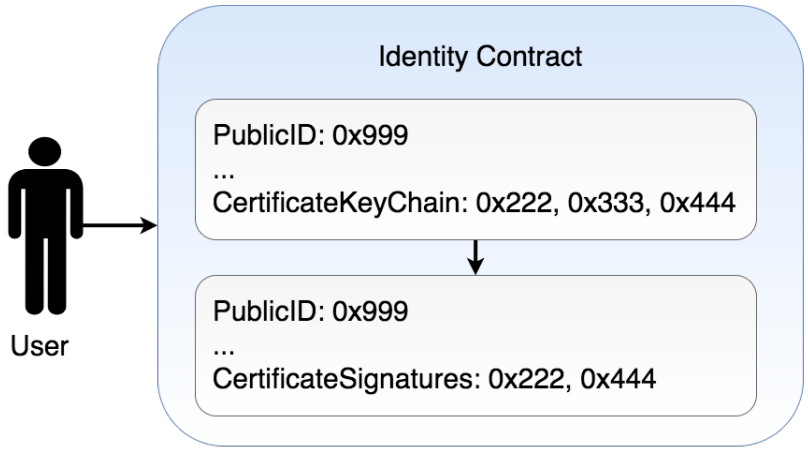


Figure 4.5: User's can check mutual connections with users on the Identity Contract.

4.5 Authentication of Email

The email log is used to link a user's certificate to an email that they send to another user. Prior to sending an email, a user generates the hash of the text to be sent. By sending a transaction to the Identity contract requesting for the creation of a new email log, the Identity contract creates a log containing the following details which are sent back to the sender:

- *Message ID*: The unique ID generated to reference the message in the hash table structure.
- *Sender ID*: The public ID of the sender of the email.
- *Receiver ID*: The public ID of the recipient of the email.
- *Message Hash*: The SHA-256 hash value of the text of the email.

The sender then is able to insert these values to the top of the email to be sent. Upon receiving this email, the recipient has the ability to lookup the email log using the unique message ID given and verify the details of the email.

Again, it is impossible for an attacker to create a log using another user's public ID as the senderID in an email log. The recipient can be assured of the authenticity of the email they have been sent and that it has not been manipulated by an attacker since they are able to verify the hash of the text in the email to the hash value shown on the Identity contract.

The process of creating a log in the Identity contract can be shown below in figure 4.6.

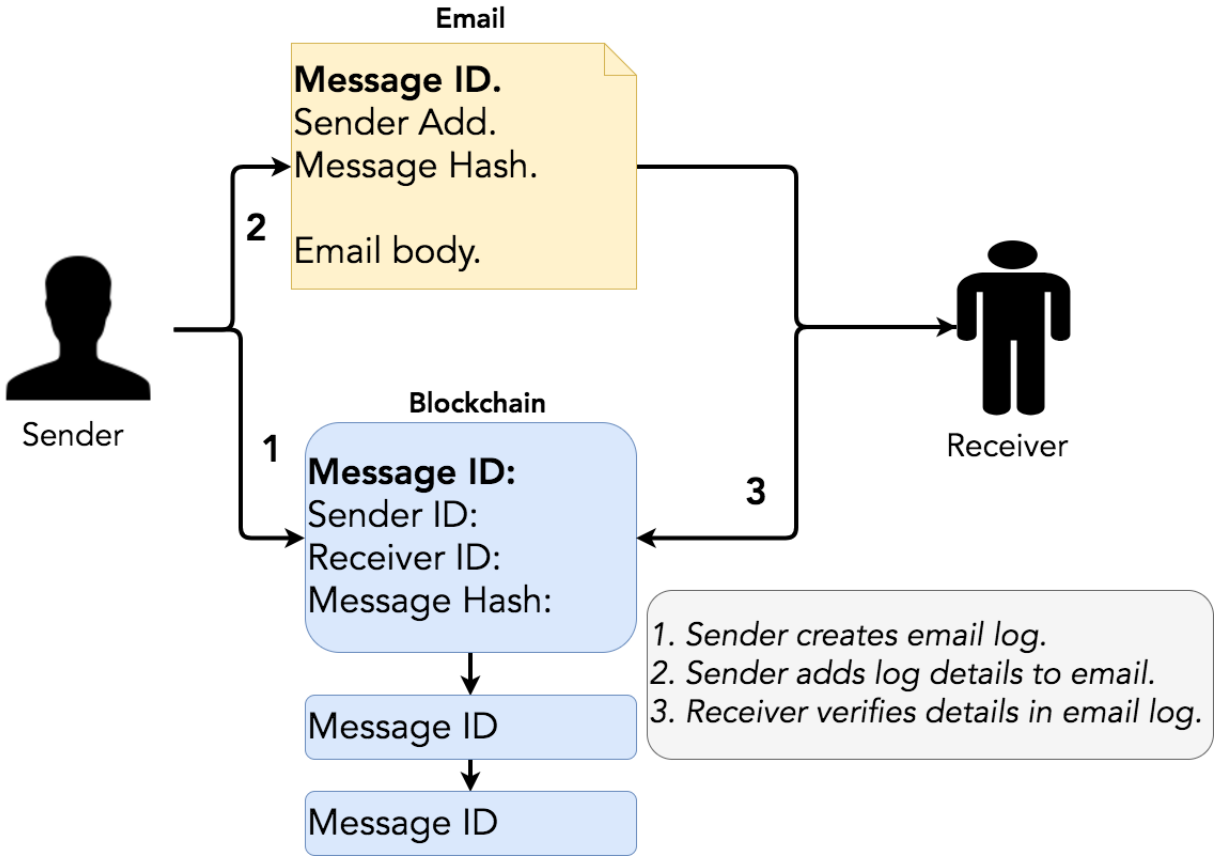


Figure 4.6: The email log permits a solution of logging email details authentically for recipients to verify the sender of an email.

Chapter 5

Implementation

This section provides an overview of the technical merit which was achieved during the completion of the project. The development tools used during the implementation, as well as a description of the processes undergone during the creation of the backe nd and the front end of the application.

5.1 Development Tools

During the development process, various technologies come together for the creation of a decentralised application that successfully communicates with the Ethereum Blockchain. Development environments, test frameworks, test blockchains and APIs are some of the various tools which were investigated to begin development. Figure 5.1 below shows the interaction of various tools used to send transactions from application's front end to the Ethereum blockchain.

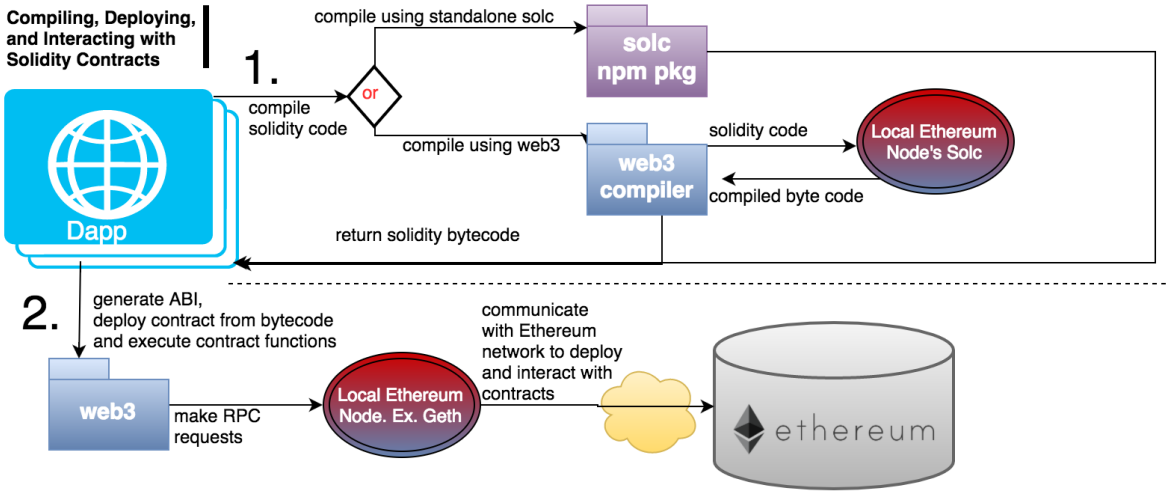


Figure 5.1: Overview of the development tools used to develop smart contracts and interact with the Ethereum Blockchain [30].

Solidity

Solidity is a JavaScript-like statically-typed object oriented programming language designed to be used for the creation of smart contracts. It is one of the four languages which can be used to be computed through the Ethereum network, with the other's being Serpent, LLL and Mutan, however Solidity is currently the primary language used [31].

Each smart contract contains a .Sol file which defines and runs the contracts behaviour upon receiving a message from another account. The solidity code is stored as bytecode within the contract.

TestRPC

TestRPC is a Node.js based Ethereum client which simulates a full Ethereum blockchain locally in the developing machine to make development and testing much faster. Testing and developing on the live Ethereum blockchain is not recommended. When run, the TestRPC provides Ethereum accounts onto the network which can each be used for sending transactions to the application being developed.

Developing and testing in early stages of an application would cost unnecessary amounts of ether. Time would also be wasted waiting for blocks to be mined making testing slow and cumbersome. The TestRPC was used during the development of the application proposed in this project.

Web3

The Web3 API is used in development as a link from the front end application to the Ethereum Blockchian. The API allows the application to send messages to read and write to the smart contracts deployed on the blockchian. During the development of this application the Web3 library was used to connect to the TestRPC blockchain.

Truffle

Truffle is a development environment and testing framework used during the development of applications for the Ethereum blockchain.

Truffle offers built in smart contract compilation, deployment and automated testing. It includes an interactive console for direct communication with contracts and automated testing. Truffle is installed with Web3.js included, and with detailed documentation aids in fast and intuitive development.

5.2 Frontend of the Application

While the focus of this project is to investigate the potential of Ethereum to be used as identity management, keeping a front end design in mind is important as it represents ultimately the most viable user interaction with a smart contract since a command line only approach is not intuitive for most users.

As email is one the applications where authentication is needed the most for everyday users, creating a seamless design where users can interact with the blockchain without leaving their email client is important. Once a smart contract is deployed, any front end application can be developed as long as correct message calls are sent to the contract.

Once a smart contract is deployed, any type of front end application can be developed as long as correct message calls are sent to the contract.

Browser extensions such as offered on Google Chrome offer the ability to run programs from within a web browser. Extensions are written in HTML, CSS and Javascript, identical technologies which are used in the development of most web pages. Browser extensions may be run through the browser's taskbar, and may be interacted with without leaving the email client web page.

The logic of the application is written using JavaScript, which can use the Web3 framework to interact with the Ethereum Blockchain. A user may log into their Ethereum account on the extension application. The application is designed to be used with user email where they can use the interface to complete the following features of the contract:

- View the certificates on their keychain.
- View connections with other users.
- Log and validate email messages onto the Identity Contract.

Figure 5.2: Creating a new email log on the blockchian by making use of the browser extension.

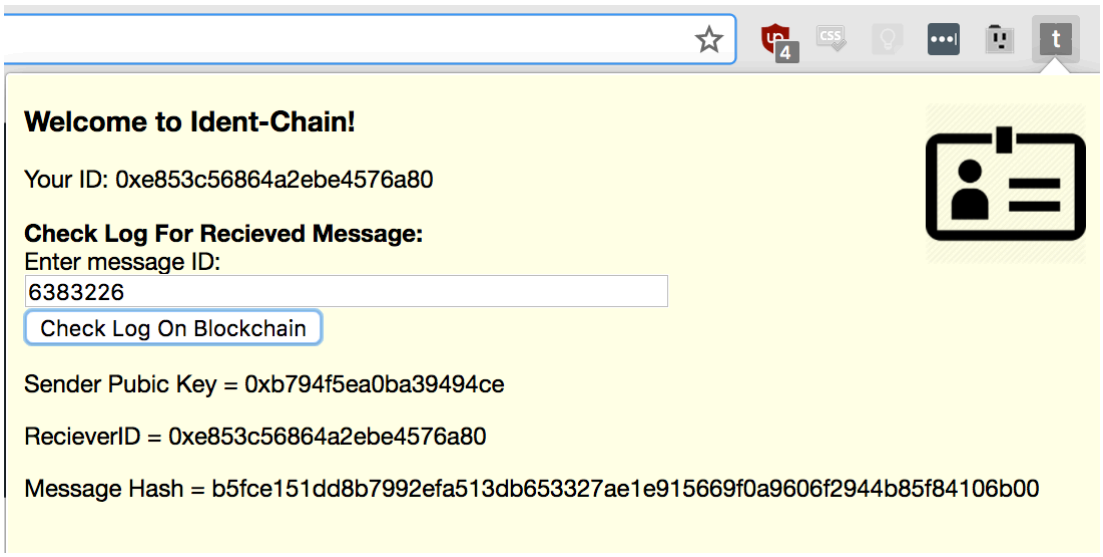


Figure 5.3: Upon receiving an email, the extension can be used to lookup the message on the email log within the Identity Contract.

A Chrome extension offers the ability allows users to easily communicate to the Identity Contract without leaving their email client. As shown in figure 5.2, the extension can be used after completion of an email. A user enters the hash and recipient of the email, and send a message to the Identity Contract to create a new email log. The Message ID is returned, and the user can easily copy and paste these details to the top of the email to

be sent. Intuitive design and ease of use is important to aid in the expanse of an application, with the Web of Trust mechanism strengthening over time with more users becoming connected.

5.3 Back-end of the Application

The back-end of the application contains all the logic used for the various smart contracts in the design. To this point, the Identity Contract has successfully been completed which contains certificate creation, the Web of Trust functionality and the email logging system. Truffle was used as a development environment where the contracts were created in Solidity and deployed thorugh Web3 calls into the TestRPC. Truffle permits the creation of JavaScript test files, where Web3 calls can be created to test the various functionality of the smart contracts deployed in the TestRPC.

Two structures are defined in Solidity on the Identity Contract; a certificate structure and an email log structure. The contract contains two hash tables which store the list of user certificates and email logs. The PubID and the MessageID are used respectively as keys to reference patricular structures in the hash table.

5.4 User Interaction to the System

The controller contract acts as a means for key recovery should a user's private key become lost or compromised. All messages sent from the user to the Identity Contract are forwared through this controller to the Identity Contract.

A core feature of Ethereum lies in the inclusion of the msg.sender() function. Upon receipt of a message, a contact may use the msg.sender() function to access the address of the account which has sent the message to the contract. This makes it impossible for attackers to forge messages to contracts from accounts other than their own.

To show an example of the process of a user interaction with the Identity Contract, Figure 5.4 below highlights the operations which occur when a user creates a new certificate on the system.

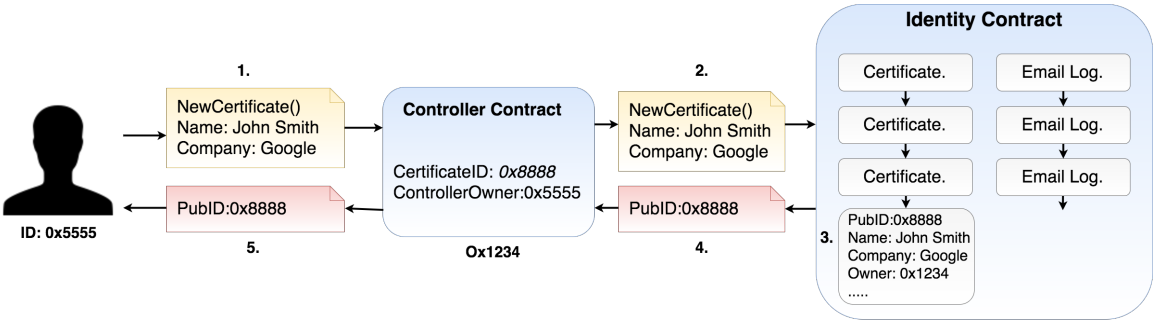


Figure 5.4: Creating a new certificate onto the Identity Contract.

1. The user sends a message to the Controller Contract containing the NewCertificate() function and additional parameters to store in the certificate as specified in section 4.2.

2. The Controller Contract verifies that the messege it received is from the Ethereum address 0x5555, as specfied in its ”Owner” value.

3. The Controller Contract forwards the message to the Identity Contract.

4. The Identity Contract creates a new certificate in its certificate table, filling in the attributes as specified in the message. The certificate owner address is set to be the address of the Controller Contract which sent the message.

5. The Identity Contract returns the new certificate’s Public ID value to the Controller Contract to store the certificate ID which it is related to.

6. The Controller Contract returns the certificate PubID value to the User as confirmation of the creation of a certificate, and so the user may use their PubID in future to verify their certificate to other users in person so that their trust network can grow.

To lookup a certificate's details or an email log in the certificate hash tables, a user may make a call to the contract passing the public ID of the certificate or message ID respectively of the object in question. The Identity Contract will return all details stored onto that object.

5.5 Difficulties Encountered During Development

Difficulties were encountered during development, which is common with the introduction of new technologies. One of the largest difficulties was a shortage of documentation and tutorials for the various development tools online. While online forums such as Reddit [32] provided a good medium for discussion on relevant Ethereum topics, the continuous updates to the Ethereum Blockchain led to many redundant discussions and tutorials online. Several libraries and development tools such as Truffle also continued to be updated throughout the research, with official documentation not being kept up-to-date with the latest releases in the case of Truffle. This led to a delay in implementing the application which ultimately was a factor in not achieving full implementation in joining the front end Chrome extension with the application.ggf

5.6 CD

A CD containing all the code which has been written, and all necessary libraries is attached at the end of this paper.

Chapter 6

Results and Evaluation

The implementation outcome was an overall success to demonstrate the viability of the design proposed. The Identity contract which holds the most functionality has been completed with the Recovery contract and Controller contract yet to be developed. The Google Chrome Extension has been built, but currently the interaction to the TestRPC has not been completed, but the implementation can be used to demonstrate a potential user interface for the application.

Several factors need to be reviewed to determine what aspects of the design are successful, and whether any obstacles exist for the deployment of this system.

Viability of User Authentication

The trust implementation on the application contains all of the benefits of PGP's Web of Trust while being further strengthened since a user would never have to regenerate new key pairs after key loss. This leads to a larger trust network being created over time. However, similar issues arise which feature in PGP's approach in the verifying a user's authentication without meeting the user in person. It is difficult to be guaranteed of an introducer's assurance of another user's identity, and since certificates are self-sovereign, nothing prevents attackers from creating a certificate with the same details of another person.

Difficulty arises for new user's to the system. By creating a new certificate, without any signatures from other users it is difficult to become trusted initially. A new user must have to meet other users in person to build up a reputation, but this may be an issue for user's living in remote areas.

Email Log Verification

Given that a user is certain of the ID of another user's certificate, then the authenticity of an email log on the system is assured since it is impossible for an attacker to forge a senderID value which is not their own.

The SHA-256 hash value stored on an email log is used to verify that an email has not been altered on receipt and to verify the link from messageID to the email received. It is highly unlikely that two emails containing different text could produce the same hash value [5], so therefore it is unlikely for an attacker to be able to alter an email sent to a user without producing a different hash value.

An issue which may arise however is that upon creating an email, a user must wait for their transaction to the Identity contract to be mined before they may enter the details onto their email. Waiting several minutes to receive this information before sending an email may be deemed too slow for many users.

Cost of Transactions

The cost of a transaction in Ethereum is determined by how computationally difficult the operation is to perform.

The creation of new certificates and email logs currently both average a computation cost of 35000 gas, which is currently equivalent to $0.003 per transaction [33]. While the application requires continuous payment per transaction as opposed to the unlimited use offered after being issued a certificate from a certificate authority, this application is likely cheaper for most users as the cost of more trusted certificate authorities ranging from $200

- $1500 per year. PGP offers a free to use service, however, the additional features such as key loss recovery offered by this application would likely be worthwhile for most users. The average person on the US sends 41 emails a day [34], leading to a total average cost of using the email log feature to be $44.89 per annum, much lower than the fee of a trusted certificate authority.

In October 2016, the Ethereum blockchain was updated and gas prices for transactions increased. [35] This was done to combat attacks where attackers flooded the chain with low cost transactions. It is not possible to predict whether the gas prices will increase dramatically again, but it should be noted that the price for using an application as proposed here may not be constant over time.

Scalability

Currently, the Ethereum documentation [25] does not specify the maximum storage within a smart contract. However, should the maximum capacity of a smart contract be reached, then it is possible to design the smart contract to store the user identities on multiple contracts, where the retrieving of a user's certificate me search between the possible contracts.

Improvements to the Design

During development, it became clear that some aspects of the design could be improved for future deployment. Implementing the email log and the Web of Trust functionality from within the Identity contract is not completely necessary, and could be moved to be stored within its own contract. While no additional functionality is gained by doing this, it is noteworthy as the Identity contract could be designed in a way similar to Uport, and have the Identity contract act on behalf on the Ethereum contract in the interaction with other smart contracts on the blockchain. The Web of Trust and email log could be interpreted as their own applications, which are solely passed identites from the Identity contract via transactions.

Chapter 7

Conclusion

The goal of this project was to investigate how user's digital identities is managed by implementations currently in place. The two most common approaches in use are through certificate authorities and PGP's Web of Trust.

The need for complete trust in certificate authorities has led to a single point of failure and vulnerability to attack. The lack of uniform background checks by certificate authorities, prior to issuing user certificates, has furthermore led to an imbalance of certificate quality in circulation.

PGP introduces a decentralised architecture known as the Web of Trust where users become responsible for signing each other's keys. A trust network is created where introducers are used to authenticate other users. This design leads to flaws where private key loss led to complete loss of ownership of an identity and ability to decrypt messages sent to the user. The ambiguity of the notion of trust of introducers and unknown user's creates an imbalance of trust in connections within the network.

The Ethereum blockchain provides a means to create decentralised applications run between user's within a network. The ability to create a decentralised application with functionality surpassing PGP and the Bitcoin blockchains limitations shows that a promising identity mechanism can be built with Ethereum.

7.1 Contribution

Analysis of the Gaps in Current Identity Mechanisms on the Blockcahin

This project investigated Namecoin and Uport, which are both the current most sucessful identity mechanisms designed on the blockchain. The limitations of Bitcoin are seen to prevent the wide application of Namecoin, however Uport offers a sound solution which provides a means to handle the recovery of user identity after key loss, and interaction of a digital identity to external applications. However, the Uport design relies on off-chain data stores to store user information, and must connect to a Uport controlled server for interaction with applications, which does not create an ideal decentralised approach.

Decentralised Identity Storage on the Blockchain

The design proposed in the project removes any need for third party control and off-chain data storage in the issuing and handling of user identities. The Identity contract implemented sucessfully stores all user information. Smart contract store certificate No need for off chain storage

Implementation of Trust Network

The Web of Trust network, which was introduced by PGP, has successfully been implemented in this system. User identities can build a reputation over time, and view mutual connections with other user's to aid in the authentication of of new interactions.

Implementation of an Email Logging Application

An email authentication implementation has been put in place, where users attach their digital identity to emails where the recipient can verify its authenticity upon receipt.

7.2 Future Work

As discussed, the solution is not fully functioning from a user's point of view. In order to deploy this system to industry, the backend contracts need to be integrated with the live Ethereum blockchain. Additionally, the front end Chrome Extension must support interaction with the blockchain via the Web3 API.

7.3 Final Remarks

The solution proposed by this dissertation proves how the introduction of the Ethereum blockchain offers an alternative design to manage user digital identities, surpassing current implementations. It will be interesting to see how the growth of Ethereum will be over the coming years. With the major growth of Internet of Things devices predicted, digital identity may become more important than ever before, and a decentralised approach offered by Ethereum may be the answer to this challange.

Bibliography

[1] R. Housley, W. Polk, W. Ford, and D. Solo, "Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile," tech. rep., 2002.

[2] P. R. Zimmermann, *The official PGP user's guide*. MIT press, 1995.

[3] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[4] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Annual International Cryptology Conference*, pp. 537–554, Springer, 1999.

[5] H. Gilbert and H. Handschuh, "Security analysis of sha-256 and sisters," in *Selected areas in cryptography*, pp. 175–193, Springer, 2004.

[6] N. Koblitz, *A course in number theory and cryptography*, vol. 114. Springer Science & Business Media, 1994.

[7] R. Biddle, P. C. Van Oorschot, A. S. Patrick, J. Sobey, and T. Whalen, "Browser interfaces and extended validation ssl certificates: an empirical study," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, pp. 19–30, ACM, 2009.

[8] G. Gebhart and S. Schoen, "Is let's encrypt the largest certificate authority on the web," 2016.

[9] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "Online certificate status protocol-ocsp," 1999.

[10] C. Ellison and B. Schneier, "Ten risks of pki: What you're not being told about public key infrastructure," *Comput Secur J*, vol. 16, no. 1, pp. 1–7, 2000.

[11] K. Zetter, "Diginotar files for bankruptcy in wake of devastating hack," *Wired magazine, September*, 2011.

[12] A. Abdul-Rahman, "The pgp trust model," in *EDI-Forum: the Journal of Electronic Commerce*, vol. 10, pp. 27–31, 1997.

[13] "Pgp diagram.." `https://upload.wikimedia.org/wikipedia/commons/thumb/4/4d/PGP_diagram.svg/500px-PGP_diagram.svg.png`, note = Accessed: 2017-04-05.

[14] M. Richardson, R. Agrawal, and P. Domingos, "Trust management for the semantic web," in *International semantic Web conference*, pp. 351–368, Springer, 2003.

[15] P. Zimmermann, "Pgp user's guide, volume i: Essential topics," *Available on the WWW via ftp://ftp. pegasus. esprit. ec. org/-pub/arne/pgpdoc1. ps. gz*, 1994.

[16] C. Ellison and S. Dohrmann, "Public-key support for group collaboration," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 4, pp. 547–565, 2003.

[17] J. Golbeck, B. Parsia, and J. Hendler, "Trust networks on the semantic web," *Cooperative information agents VII*, pp. 238–249, 2003.

[18] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[19] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, and S. Čapkun, "Misbehavior in bitcoin: A study of double-spending and accountability," *ACM Transactions on Information and System Security (TISSEC)*, vol. 18, no. 1, p. 2, 2015.

[20] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, *et al.*, "On scaling decentralized blockchains," in *International Conference on Financial Cryptography and Data Security*, pp. 106–125, Springer, 2016.

[21] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[22] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better—how to make bitcoin a better currency," in *International Conference on Financial Cryptography and Data Security*, pp. 399–414, Springer, 2012.

[23] J. R. Douceur, "The sybil attack," in *International Workshop on Peer-to-Peer Systems*, pp. 251–260, Springer, 2002.

[24] H. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An empirical study of namecoin and lessons for decentralized namespace design," in *Workshop on the Economics of Information Security (WEIS)*, Citeseer, 2015.

[25] V. Buterin *et al.*, "Ethereum white paper," 2013.

[26] G. Prisco, "Slock. it to introduce smart locks linked to smart ethereum contracts, decentralize the sharing economy," *Bitcoin Magazine. Nov-2015 [Online]. Available: https://bitcoinmagazine. com/articles/slock-it-to-introduce-smart-locks-linked-to-smart-ethereum-contracts-decentralize-the-sharing-economy-1446746719.[Accessed: 20-May-2016]*, 2015.

[27] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.

[28] F. Jacobs, "Providing better confidentiality and authentication on the internet using namecoin and minimalt," *arXiv preprint arXiv:1407.6453*, 2014.

[29] J. T. Z. M. M. S. Dr. Christian Lundkvist, Rouven Heck, "Uport: A platform for self-sovereign identity draft version," 2016.

[30] "Karldottechl 5 essential ethereum dapp tools.."

[31] B. M. Galan, "Decentralized application platform ethereum,"

[32] "r/ethereum, the front page of the web 3.." `https://www.reddit.com/r/ethereum/`, note = Accessed: 2017-04-05.

[33] E. B. E. Etherscan, "Etherscan gas price history,"

[34] P. Sara Radicati, "Email statistics report, 2014-2018," 2014.

[35] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts," tech. rep., Cryptology ePrint Archive: Report 2016/1007, https://eprint. iacr. org/2016/1007, 2016.