UPORT: A PLATFORM FOR SELF-SOVEREIGN IDENTITY

DRAFT VERSION (2016-10-20)

Dr. Christian Lundkvist, Rouven Heck, Joel Torstensson, Zac Mitton, Michael Sena

INTRODUCTION

History

There are many problems with the current state of identity systems. Digital identity is fragmented and siloed between various service providers, prohibiting a holistic view, and delivering poor user experience necessitating repetitive registrations and logins with usernames and passwords. This results in insecure systems where people use the same password for many of their sites. The centralized servers of identity providers like Google and Facebook are honeypots of data, so they're economically valuable for hackers to attempt to crack. The upcoming reliance on billions of internet-of-things devices makes it untenable to have all those devices controlled by a centralized identity provider, since a breach of this provider would prove catastrophic to not only digital but also physical infrastructure.

Public/private key cryptography and decentralized technologies like blockchains offer a promising solution to the problems mentioned above. These technologies push ownership of identity away from centralized services to the edges - to individuals - so that the identities themselves are in control. This is commonly referred to as *self-sovereign identity*. This approach decentralizes data and computation and pushes them to the edges, where it is less economically valuable to hackers because it would require a lot of effort to hack many individual identities one-by-one.

However, introducing new technologies to end-users is difficult. Public-key cryptographic tools like PGP have been around for 25 years, but their use in digital identity systems have seen little use due to their unintuitive and complex user experience, and the fact that usernames and passwords work well enough for most people. Blockchain technologies are interesting in that they *require* the use of cryptographic keys to sign messages for each interaction of the blockchain. Thus the rise of cryptocurrencies like Bitcoin and general blockchain architectures like Ethereum have sparked new interest in making public key cryptography usable to regular consumers and users in order for them to interact with these systems.

Interactions with blockchain based systems necessitate usable public-key cryptography, and up to this point the key management solutions (commonly called "wallets") have been difficult to use for non-technical users. Interestingly the blockchain can itself help make public-key cryptography more usable and secure by acting as a decentralized public key infrastructure (PKI). The blockchain can be viewed as a decentralized certificate authority that can maintain the mapping of identities to public keys. Smart contracts can furthermore add sophisticated logic that helps with key revocation and recovery, lessening the key management burden for the end user.

Introduction to Uport

Uport is a secure, easy-to-use system for self-sovereign identity, built on Ethereum. The uPort technology consists of three main components: smart contracts, developer libraries, and a mobile app.

The mobile app holds the user's keys. Ethereum smart contracts form the core of the identity and contain logic that lets the user recover their identity if their mobile device is lost. Finally the developer libraries are how third party app developers would integrate support for uPort into their apps.

uPort identities can take many forms: individuals, devices, entities, or institutions. Uport identities are self-sovereign, meaning they are fully owned and controlled by the creator, and don't rely on centralized third-parties for creation or validation. A core function of a uPort identity is that it can digitally sign and verify a claim, action, or transaction - which covers a wide range of use cases.

An identity can be cryptographically linked to off-chain data stores. Each identity is capable of storing the hash of an attributed data blob, whether on IPFS, Azure, AWS, Dropbox, etc., which is where all data associated with that identity is securely stored. Identities are capable of updating this file themselves, such as adding a profile photo or a friend, or they can also grant others temporary permission to read or write specific files.

Since they can interact with blockchains, uPort identities can also control digital bearer assets such as cryptocurrencies or other tokenized assets.

Proposed Use Cases

A self-sovereign identity system will have many use cases, here a few of them are presented:

uPort allows end-users to: own and control their personal identity, reputation, data, and digital assets; securely and selectively disclose their data to counterparties; access digital services without using passwords; digitally sign claims, transactions, and documents; control and send value on a blockchain; interact with decentralized applications and smart contracts; and encrypt messages and data.

uPort allows enterprises to: establish a corporate identity; easily onboard new customers and employees; establish an improved and transitive Know-Your-Customer process; build secure access-controlled environments with less friction for employees; reduce liability by not holding sensitive customer information; increase compliance; maintain a network of vendors; establish role-specific, actor-agnostic identities (i.e. CTO) with specific permissions.

TECHNICAL OVERVIEW

Ethereum and Smart Contracts

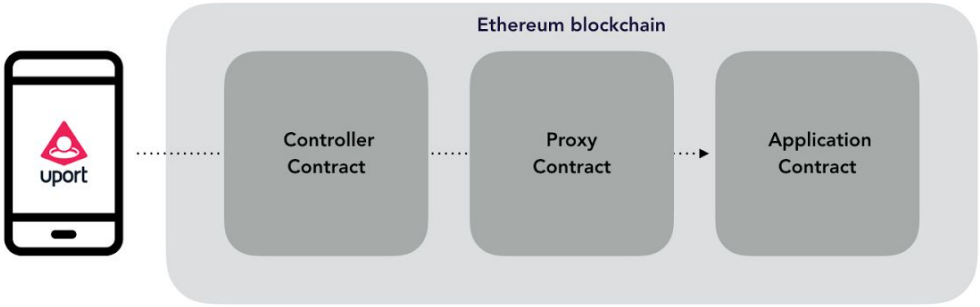
Ethereum is a blockchain architecture with an associated state database, capable of storing programs and their state. These programs are commonly referred to as *Smart Contracts.* A smart contract can be deployed by any Ethereum user and it has a function-based interface. Once deployed the smart contract can be referenced by its *address*, which is a cryptographic identifier. A user can call a smart contract function by sending a transaction with this address as the destination, and with the data payload of the transaction containing the function signature and input parameters. Calling a function causes the miners of the network to execute the program in a trust-minimized way and update its state. A smart contract can hold and send the native value token Ether, and can furthermore call functions of other smart contracts.

For further reading on Ethereum and Smart contracts, see the Ethereum white paper.

uPort Technical Overview

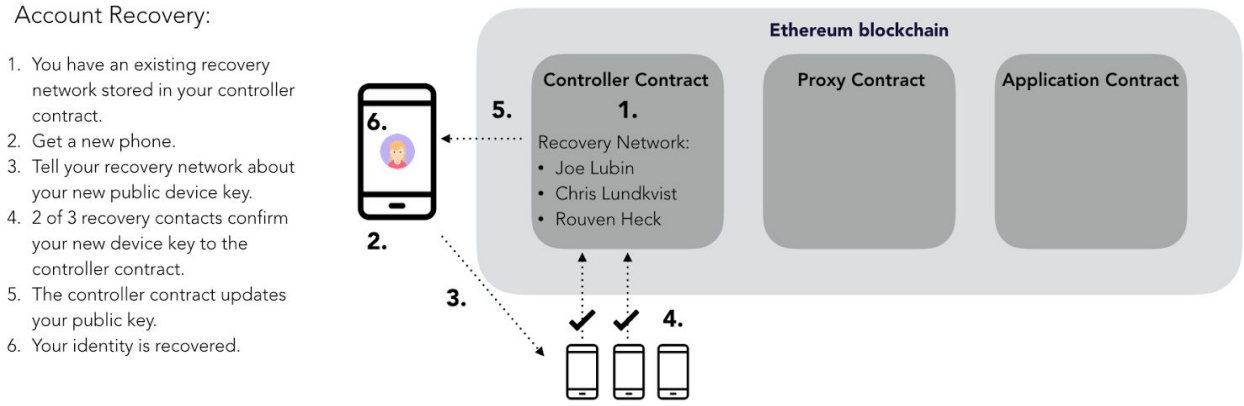
At the core of a uPort identity is the uPort identifier, a 20-byte hexadecimal string that acts as a globally unique, persistent identifier. This identifier is defined as the address of an Ethereum smart contract known as a Proxy contract. The Proxy contract can relay transactions and it is through this mechanism that the identity interacts with other smart contracts on the Ethereum blockchain.

When the user wants to interact with a particular application smart contract, they send a transaction through the Proxy contract, via a Controller contract, which contains the main access control logic. The Proxy contract then forwards this transaction to the application smart contract. This architecture allows the application to view the Proxy contract address as the interacting entity. The Proxy contract thus introduces a layer of indirection between the user's private key - stored on their mobile device - and the application smart contract.



The purpose of having a Proxy contract as the core identifier is that it allows the user to replace their private key while maintaining a persistent identifier. If the user's uPort identifier instead was the public key corresponding to their private key, they would lose control over their identifier if they were to lose the device where the private key is held.

In the case of device loss, the Controller contract maintains a list of recovery delegates that can help the uPort user recover their identity. These delegates can be individuals, such as chosen friends and family members, or institutions, like banks and credit unions. A quorum of delegates is required to let the user recover their identity and connect it to a new device.

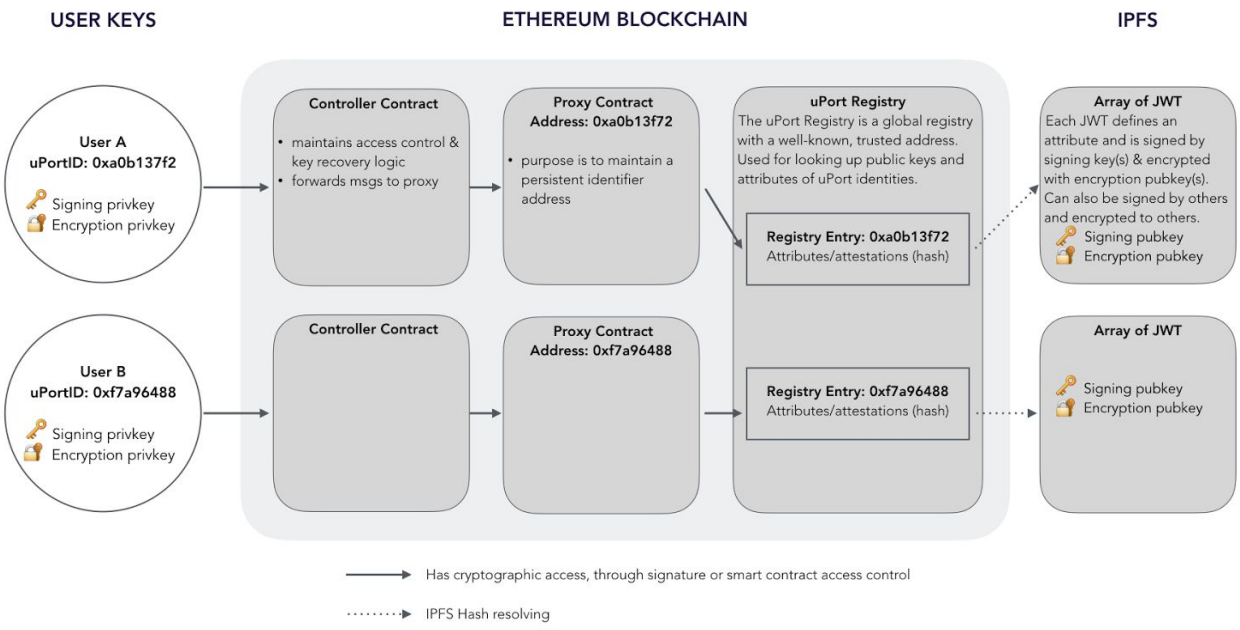


We can also use uPort for non-blockchain identity-related use cases. This is achieved by cryptographically binding an external data structure to the uPort identifier using a Registry contract. The Registry contract contains a mapping from a uPort identifier to an IPFS hash. IPFS is a decentralized system for storing, linking and transporting data. The hash guarantees the integrity of the data structure, and the cryptographic binding to the identifier is defined by smart contract access control: only the uPort proxy is authorized to update the Registry contract.

The data structure corresponding to the IPFS hash can contain profile information like Name, Profile picture, etc. It can also contain data such as public keys in order to support a decentralized public key infrastructure. The data structure used is a collection of JSON schemas originally from schema.org and enhanced in collaboration with Blockstack. Each JSON schema can be digitally signed with a private key to create a JSON Web Token. This token can then be used as an off-chain attestation.

An attestation is a very general structure. It can be used as proof that a certain identity makes a claim about another identity. It can also be a self-signed certificate stating that a public key belongs to a specific identity. Furthermore, an attestation can be used to provide a two-way link to a service like Twitter, allowing the user to leverage their existing social network.

If the uPort user possesses an encryption key, the attributes and attestations linked to the Registry can also be encrypted. This way the profile data can be held private by default and by encrypting it to the public encryption keys of other identities, the data can be shared with these identities.



Technical Components

In this section we present in more detail the main components of the uPort system.

Smart Contract Components

- The *Proxy Contract* is a minimal contract, used to forward transactions and its address is the core identifier of a uPort identity.
- The *Controller Contract* maintains access control over the Proxy contract, and allows for additional functionalities.
- The *Recovery Quorum Contract* facilitates identity recovery in case of key loss.
- The *Registry Contract* maintains cryptographic bindings between a uPort identifier and the off-chain data attributes associated with it.

Data Components

- *Attestations* are signed data records containing profile attributes and/or verifiable claims, stored off-chain.
- *Selective disclosure* is a mechanism for encrypting data that adds a layer of privacy to attributes and attestations.

Developer Components

- A *Developer library* allows for simple integration of uPort into decentralized applications or existing digital services.

Mobile Components

- A *Mobile application* stores the identity's private key, which is used to control the identity and sign attestations, in the smartphone's secure enclave.

Server Components

- *Chasqui - messaging server*
- *Sensui - gas fueling server*
- *Infura RPC*
- *Infura IPFS*

SMART CONTRACT COMPONENTS

Proxy Contract

A large part of the uPort system rests on the concepts that in the Ethereum EVM, a contract address and the hash of a public key can both be the origin of a message sent to a smart contract, and that the target smart contract sees no difference between these two cases. This makes it possible to forward almost any interaction with Ethereum through a *Proxy Contract* which lies between the initial signed transaction and its intended target. One of the primary benefits of such a scheme is the ability to add features like key recovery or spending limits, which are not possible using a simple private/public key pair. Another benefit is that the proxy address can stay the same as keys are recovered or updated, thus allowing a user to maintain an immutable identifier capable of acquiring a reputation over time as third parties attest to its authenticity and actions.

The proxy is standardized, with two main features. The owner of the Proxy Contract can:

1. Forward an Ethereum transaction to an external address
2. Swap out the owner for a different one

The proxy contract Solidity code is very simple, and is presented here for reference:

```
contract Owned {
    address public owner;
    modifier onlyOwner(){ if (isOwner(msg.sender)) _ }
    modifier ifOwner(address sender) { if(isOwner(sender)) _ }

    function Owned(){
        owner = msg.sender;
    }

    function isOwner(address addr) public returns(bool) { return addr == owner;
}

    function transfer(address _owner) onlyOwner {
        owner = _owner;
    }
}

contract Proxy is Owned {
    event Forwarded (address indexed destination,uint value,bytes data);

    function forward(address destination, uint value, bytes data) onlyOwner {
        if (!destination.call.value(value)(data)) {throw;}
        Forwarded(destination, value, data);
    }
}
```

Controller Contract

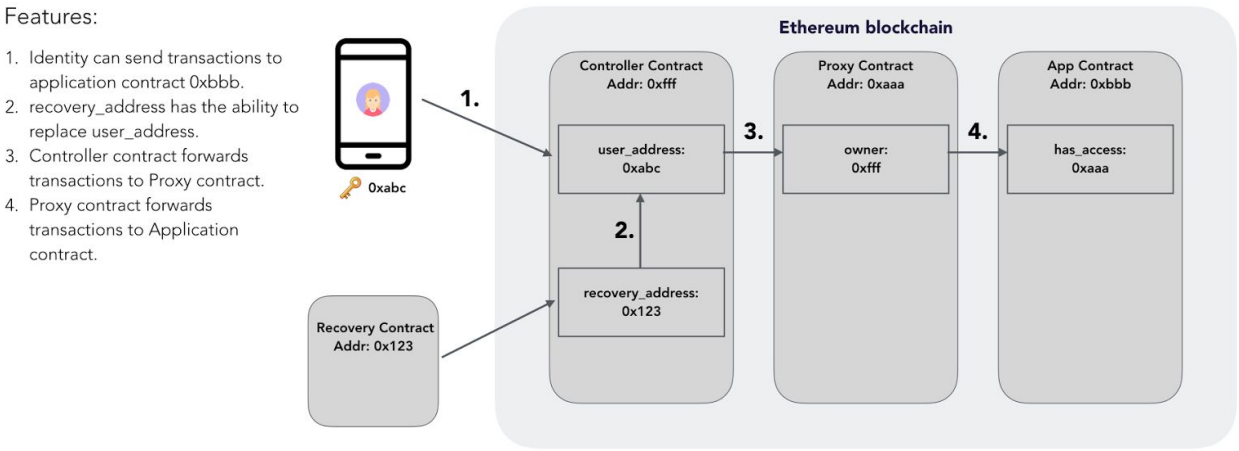
As the uPort software matures, we want users to be able to update their smart contract logic without changing their core uPort identifier, which is tied to reputation, assets and history. With that objective in mind, the Proxy Contract was designed to be extremely simple. Another contract, the *Controller Contract*, was designed to act as the formal owner of the Proxy Contract. The Controller Contract maintains core access control features that allow the user to authenticate themselves (using their private key) to the Proxy Contract, and have it act on their behalf. As better controllers are eventually developed, users will be able to replace their controller with a new one without affecting the services linked to their identity.

The first version of the *Controller Contract* is RecoverableController.sol. This contract has two main addresses associated to it: the *user address* and the *recovery address*. The user address corresponds to the private device key of the user and is the standard way of interacting with the contract. The recovery address is used to help the user recover control if they lose their key. We use a separate contract to represent the recovery address; see the Recovery Quorum section below.

Specifically the following interactions are defined:

1. The user address can forward transactions to the Proxy Contract
2. The user address can relinquish control of the Proxy Contract to a new Controller Contract (timelocked)
3. The user address can replace itself with a new user address (timelocked)
4. The recovery address can replace the user address with a new address.

Interactions 2 and 3 are “timelocked”, meaning they take a certain amount of time to go into effect after they are requested. This extra security measure gives the user additional time to recover their account using the recovery address in the event they have had their key stolen, and is under attack by a malicious entity.



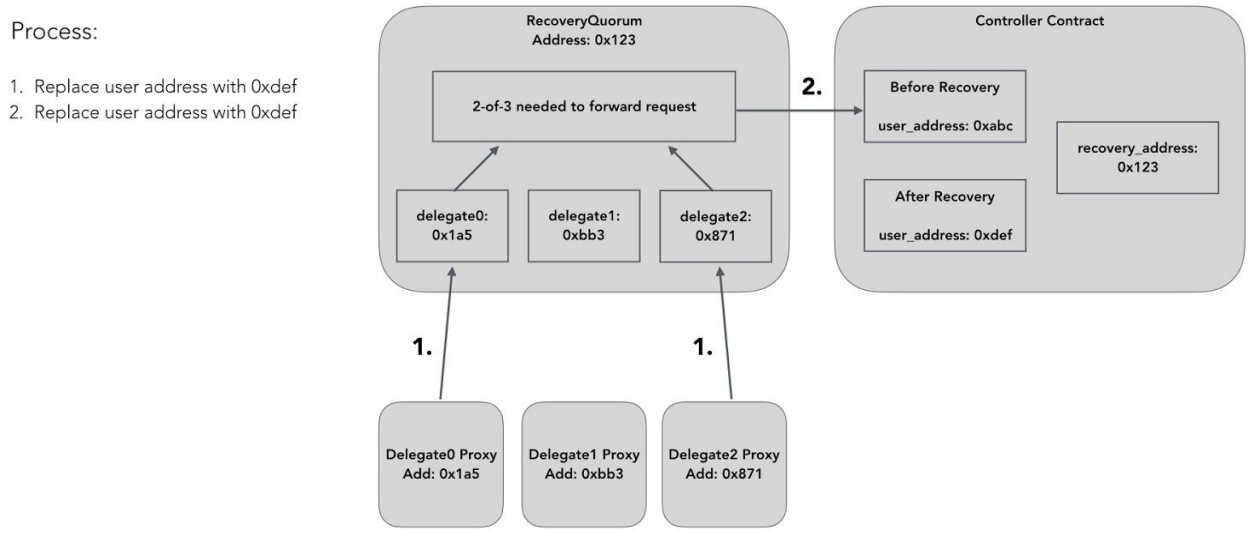
Recovery Quorum Contract

In the above description of RecoverableController.sol the recovery address has the ability to swap out the main user address of the Controller. We currently give this access to a contract called RecoveryQuorum.sol. This is a multisig contract that is controlled by the user’s friends or other trusted entities, known as *recovery delegates*. The recovery delegates can specify a new user address for the Controller Contract. This feature gives the uPort user a means of recovering their identity in the event they lose their device. After the user gets a new device and they communicate the newly generated device key to their recovery delegates, the recovery delegates can then replace the corresponding user address with this new one.

The Recovery Quorum contract offers the following interactions:

1. A recovery delegate can sign a vote to change the user address - which goes into effect after more than half of the recovery delegates have voted for the change.
2. The user address can add/remove a recovery delegate (timelocked)

Interaction number 2 implies that an attacker who compromises the user's device key can replace recovery delegates with her own and take full control over the identity. However since the interaction is timelocked, existing recovery delegates have a grace period where they can replace the user key as well as and/or block the addition and removal of recovery delegates, which would effectively thwart the attack.



Registry Contract

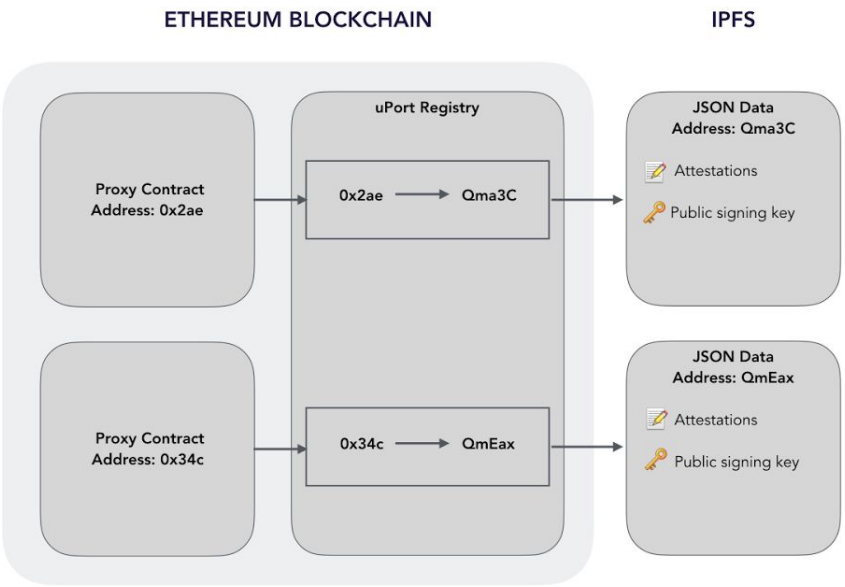
The *Registry Contract* maintain cryptographic bindings between a uPort identity and an off-chain data structure (IPFS, etc).

This smart contract is the main entry-point for accessing the attributes associated to a uPort identity.

The uPort Registry Contract acts as a logically centralized but physically decentralized registry or lookup table mapping each uPort identitifier to an IPFS hash linking to a structure containing the user's attributes, profile data and attestations (see below for more information on attestations).

IPFS is a decentralized object storage and retrieval system. The hashing guarantees the integrity of the data, while also allowing retrieval from IPFS peers that hold the data. Instead of storing data in IPFS we can also use more traditional services like Dropbox, Microsoft OneDrive, etc.

Through the cryptographic access controls built into the Ethereum blockchain, we can guarantee that only the owner of the uPort identity (i.e. the holder of the device key) has the right to modify the corresponding registry entry.



DATA COMPONENTS

Attributes and Attestations

The uPort Registry cryptographically links profile data or *attributes* to a uPort identifier. This data can exist either as a plain JSON structure, or as a signed JSON web token, which we denote as an *attestation* or *claim*. We currently follow the standard of blockchain-profile-js for the JWT token formats and extensions of the schema.org/Person schema.

An example of an attribute:

```
{“name” : “Christian Lundkvist”}
```

An example of an Attestation:

```
{ token:
'eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NksifQ.eyJhdHRyaWJ1dGUiOnsibmFtZSI6IkNocmlzdGl
hbiBMdW5ka3Zpc3QifSwic3ViamVjdCI6IjB4N2RiNTM1ZmFjOGEyNTg2OWE3YWViNWY3ODQyYzcxZD
```

```
U5ODI0ODcyMyIsImlzc3VlciI6IjB4Y2I4NTE4ZTFhYjQyMmM4NjQ1ODQ3ZTYzYmQ1N2Q4YTcwYWFhY
jY0YyIsImlzc3VlZEF0IjoiMjAxNi0wMS0yNlQyMjo0ODoyOS42MzlaIiwiZXhwaXJlc0F0IjoiMjAx
Ny0wMS0yNlQyMjo0ODoyOS42MzlaIn0.LO_rHNSq_Piow3kLTqks86BVsYLIWpUoN7LMBXqD8q3YR2I
v6q9BWjvtPKao34HlZKqPDZeXtjOVFPXmH5eRMg',
  data:
   { header: { typ: 'JWT', alg: 'ES256K' },
     payload:
      { attribute: {“name” : “Christian Lundkvist”},
        subject: {“uportId”: “0x7db535fac8a25869a7aeb5f7842c71d598248723”,
"publicKey":"020313f206e46575c98ed35658378fc76abfbad99900a05b0093a743a5d79dff67
"}
        issuer: {“uportId”: “0xcb8518e1ab422c8645847e63bd57d8a70aaab64c”,
"publicKey":"02e8a9b0aeee81f80ca32eb09d97319d61b5df9485bdf6f726465155ca778f69f1
"},
        issuedAt: '2016-01-26T22:48:29.639Z',
        expiresAt: '2017-01-26T22:48:29.639Z' },
     signature:
'LO_rHNSq_Piow3kLTqks86BVsYLIWpUoN7LMBXqD8q3YR2Iv6q9BWjvtPKao34HlZKqPDZeXtjOVFP
XmH5eRMg' },
  encrypted: false,
  publicKey:
'02e8a9b0aeee81f80ca32eb09d97319d61b5df9485bdf6f726465155ca778f69f1' }
```

Signing user attributes allows for other identities to verify and attest to the validity of the profile data of the identity. This can be useful for KYC where a bank can attest to the validity of profile data or other attributes of their customer. The bank customer can then use the attestation as a portable KYC token in order to access other financial services.

Other examples of attributes/attestations can be public key certificates, allowing the uPort identity to act as its own self-sovereign certificate authority, provisioning keys for applications, devices or services and revoking them as necessary.

Another important example is the notion of *linked profiles*. This term refers to using existing social media services to bootstrap your identity. For instance, in order to link my Twitter profile to my uPort identity we would create an attribute such as

```
 {
      "@type": "Account",
      "service": "twitter",
      "identifier": "ChrisLundkvist",
      "proofType": "http",
      "proofUrl":
"https://twitter.com/ChrisLundkvist/status/720015561703493632"
}
```

The above attribute contains a Twitter identifier, `ChrisLundkvist`. Signing the attribute with the uPort signing key creates a claim that the uPort identity controls the twitter account with the handle ChrisLundkvist. The attribute also contains a `proofUrl` which links to a Twitter post stating that this Twitter user also controls this particular uPort identity. Thus this attestation creates a two-way link connecting the uPort identity with the Twitter profile.

Selective Disclosure

One of the most challenging issues for blockchain-based applications is privacy. Blockchain security relies on all participants being able to validate the state of the chain, and therefore the state needs to be public.

The uPort system can be used in a public fashion where attributes and attestations are unencrypted and in the public for everyone to see, similar to a public Twitter profile. However, we envision many applications where the user wants to keep their attributes private. *Selective disclosure* adds a layer of privacy to uPort by allowing the user to encrypt some or all attributes by default and choose who to share the data with.

The system relies on each uPort identity having a public encryption key. The disclosure of an attribute works by having the user encrypt an attribute with a symmetric encryption key. This key is then individually encrypted using the public encryption key of each identity that is allowed to read this attribute.

The final detailed specification is not yet finalized, but the main idea is the following:

Let `A` be an attestation. Suppose we wish to share this attestation with only identities `X`, `Y` and `Z`. We first generate a random symmetric key `k`, and encrypt `A` symmetrically. We denote this ciphertext `sym(k, A)`.

Now we assume that `X`, `Y` and `Z` each have a public encryption key. Let `asym(U,V,d)` denote the asymmetric encryption between identities `U` and `V` of some data `d`. Specifically

```
asym(U,V,d) = sym(DH(U,V),d)
```

Where `DH(U,V)` is a symmetric key generated from the public key of `U` and the private key of `V` using a Diffie-Hellman key exchange.

Create a random and ephemeral public/private key pair `R`, and create the data blob made up of

```
( sym(k,A), asym(X,R,k), asym(Y,R,k), asym(Z,R,k) )
```

This data blob can now be decrypted only by the identities `X`, `Y` and `Z` in order to retrieve `A`, and we store this data blob among our attestations where it can be retrieved. Note that since the key `k` is small, the overhead of encrypting to several identities is small, even though the attestation `A` might potentially be large.

This general protocol is relatively standard, and is used by encryption protocols like Minilock and by the Keybase saltpack protocol.

DEVELOPER COMPONENTS

Developer Libraries

We make a javascript library, `uport-lib`, available to developers to allow them to easily integrate support for uPort into their applications. This library allows the developer to integrate uport login events and signing of transactions using a standard web3 provider. We also provide another library, `uport-persona`, which helps the developer with easily querying the attributes and profile data of a uPort identity from within their dapp.
Future enhancements will include ways to log in to traditional web services using your uPort as well as to sign documents and other data.

The uport developer libraries can be found here:

https://github.com/ConsenSys/uport-persona
https://github.com/ConsenSys/uport-lib

MOBILE COMPONENTS

Mobile Application

The mobile application is the way the end user interacts with their uPort, and it's the primary means of managing the user's private keys. The main idea is for the user's key to be held in the secure enclave of their device and accessed via local biometric authentication whenever the key is used to sign. The key remains on the device and there is no means of exporting the private key off the device.

User experience is one of the main design goals of the mobile app interaction model. When interacting with a decentralized application on Ethereum there are two main actions:

1. "Connect": Provide the uPort identifier (or in general an Ethereum address) to the dapp
2. "Verify/validate/authorize" an interaction: Signing a transaction with the private key

The interaction is modeled after design patterns found in popular applications like WhatsApp and WeChat. The desktop version of these apps show a QR code, and scanning this QR code with the mobile version of the app will log the user in. This exact user flow is used by the uPort mobile app for the "connect" flow in a decentralized app.

When the user needs to confirm an interaction with the blockchain (i.e. sign a transaction) another QR code is shown. The user scans this code and is presented by a confirmation screen where they can confirm the interaction using their fingerprint. This is similar to the BankID system used in Sweden for interacting with banks and verifying transactions.

If a dapp is run in a mobile browser the interaction model is slightly different. Instead of displaying a QR code the dapp will instead ask to launch the uPort app. The user will then be redirected to the uPort app where they can either authorize the release of their identifier or sign a transaction. Once the action is taken the user is taken back to the mobile browser for continued interaction with the dapp.

We are deliberately aiming to create a user experience where the user doesn't need to know about public/private key cryptography. The mental model of the user should be that their smartphone & uPort app can be used to interact with dapps, log into websites, verify transactions, sign documents in their name etc, and if they lose their phone they can ask their selected friends to help them recover their identity again.

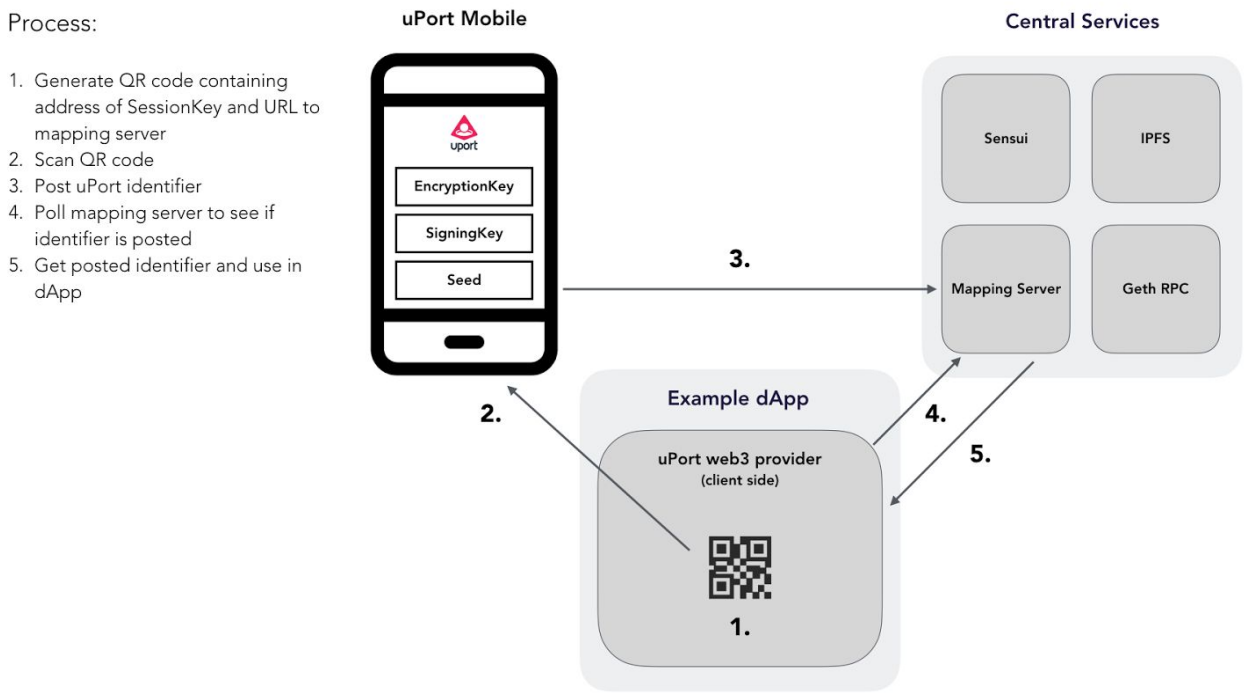
SERVER COMPONENTS

Chasqui (Messaging server)

The messaging server, Chasqui, is in charge of communication from a desktop-based decentralized application frontend to and from the mobile app. A uPort user will initially connect to a dapp by scanning the "connect with uPort" QR code. This code contains a session ID that is shared between the mobile app and the dapp frontend. When connecting with uPort the dapp frontend shows the QR code and will then poll the Chasqui server for a uPort identifier posted to the session ID. The mobile app will post the identifier and this is then immediately picked up by the dapp frontend.

Similarly when sending a transaction via the mobile app the dapp will poll the Chasqui server for a transaction hash. Once the transaction hash has been posted the dapp can immediately show that a transaction has been sent and update the UI accordingly.

Note that the Chasqui server is not used when the decentralized application is run in a mobile browser.



Sensui (Fueling server)

The fueling server, Sensui, helps new users of Ethereum overcome the initial hurdle of needing to purchase Ether to pay the fees needed to use the network. Sensui works to alleviate this problem by paying the gas fees for the user, permitting the user to create a new uPort and get up and running immediately.

The Sensui server works by utilizing the account based infrastructure of the Ethereum network. Suppose a user wants to send a transaction but they have no Ether in their sending account. The user signs the transaction as they normally would, then sends the signed transaction to the Sensui server. The server verifies that the sending account does not have enough Ether to pay for the gas, sends enough Ether to the account to pay the fees, and finally sends the user-signed transaction to the network.

Infura Ethereum RPC

The Ethereum RPC endpoint provider, Infura, allows uPort to communicate with the Ethereum network through the standard RPC interface that Infura provides. This allows the use of the uPort mobile app until there are mature mobile light-clients available for Ethereum.

Infura IPFS

Infura also allows the uPort mobile app to connect to an IPFS node that allows for communication with the IPFS network.

FUTURE ENHANCEMENTS

The uPort architecture will go through a number of enhancements in the future. Here are some planned future additions:

In the current architecture all recovery delegates of a user is publicly available on the blockchain. This could pose a security risk since an attacker could decide to attack a user's delegates in order to compromise their identity. In the future we will use a system of single-use recovery keys that are not tied publicly to specific identities.

Currently the mobile app only holds a single identity. In the future the user will have the ability to have multiple personas for different purposes.

The data stored temporarily on the Chasqui server is not encrypted at the moment which can be a privacy concern. The intention is to make all communications over this server end-to-end encrypted so that the server only stores encrypted information.

The Sensui server will in the future employ a more sophisticated architecture to pay fees for the user, which will involve smart contract logic built into the Controller Contract.

CONCLUSION

This whitepaper has presented uPort, an identity systems that aims to be a flexible and easy-to-use method of interacting with decentralized applications as well as off-blockchain identity related tasks. The system aims to abstract away the public key cryptography from the end user to make the user experience intuitive. A mobile app holds the user's private key and a smart contract address acts as their identifier. We use a novel identity recovery mechanism to let the user select friends from their contact list which gives a quorum of these friends the ability to recover the identity of the user if their mobile device is lost.

REFERENCES

[1]https://github.com/WebOfTrustInfo/rebooting-the-web-of-trust/blob/master/topics-and-advance-readings/PGP-Paradigm.pdf

[2] http://ipfs.io/

[3] https://github.com/ethereum/wiki/wiki/White-Paper

[4] https://github.com/blockstack/blockchain-profile-js

[5] https://minilock.io/

[6] https://github.com/keybase/saltpack

[7] https://github.com/ConsenSys/uport-persona

[8] https://github.com/ConsenSys/uport-lib

[9] https://web.whatsapp.com/

[10] https://web.wechat.com/

[11] https://www.bankid.com/en/