Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

A Reliable and Universal Cloud Wallet

Master Thesis

Pau de la Cuesta i Sala

Tutor: Juan Hernández Serrano

A thesis submitted in partial fulfillment of the requirements for the
Master in Advanced Telecommunication Technologies
MATT

Department of Telematics Engineering
Universitat Politècnica de Catalunya

Barcelona, Autumn Semester 2020/2021

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem Statement . . . 3
- 1.3 Methodology . . . 4
- 1.4 Cybersecurity Approach . . . 5

2 Related Work **6**
- 2.1 Ethereum . . . 6
- 2.2 Cyberattacks . . . 7
  - 2.2.1 Brute-force Attacks . . . 7
  - 2.2.2 Impersonation Attacks . . . 7
- 2.3 Key Stretching and Key Derivation Functions . . . 8
- 2.4 PBKDF . . . 9
- 2.5 Scrypt . . . 9
- 2.6 Password-Authenticated Key Exchange . . . 10

3 High-Level Architecture **12**

4 Frontend Application **13**
- 4.1 Browser Extension vs SPA . . . 13
- 4.2 Client Library . . . 13
- 4.3 Example Application . . . 14

5 Backend Server **16**
- 5.1 Infrastructure . . . 16
- 5.2 Public API . . . 16
- 5.3 Secrets Vault . . . 17
- 5.4 Security Aspects . . . 18
  - 5.4.1 Key Stretching . . . 18
  - 5.4.2 Authentication . . . 19
- 5.5 OpenAPI Specification . . . 19

6 Client-Server Interactions **21**
- 6.1 Sign Up . . . 21
- 6.2 Login . . . 22
- 6.3 Create Resource . . . 23
- 6.4 Import Resource . . . 24
- 6.5 Sign Message . . . 25

7 Conclusions **28**

Bibliography **29**

List of Figures

1	Verifiable claim components	2
2	i3 Market Backplane Architecture	4
3	Project Gantt Chart	5
4	Blockchain cryptographic relationship	6
5	Man-in-the-Middle Diagram	8
6	Key Stretching deriving a strong cryptographic key	9
7	OPAQUE Diagram	11
8	High-level diagram of the application	12
9	Cloud Wallet App main page	14
10	Cloud Wallet App key selection page	15
11	Cloud Wallet App signing page	15
12	API Schemas excerpt from Cloud Wallet Server documentation website	19
13	Server OpenAPI Specification 3 Website	20
14	Sign Up diagram	21
15	Login diagram	22
16	Create Resource diagram	23
17	Import Resource diagram	24
18	Sign Message with Key ID	26
19	Sign Message without Key ID	27

List of Tables

1	Estimated cost of hardware in $ to crack a password	10
2	Estimated derivation time on Browser and Node.js of different working factors N for *scrypt-pbkdf* implementation	18

Abstract

This document explains the design, implementation and deployment of a secure Ethereum wallet backed up by a vault of secrets on the cloud. It also introduces key concepts for this task in the field of cryptography and cybersecurity.

This project is done in collaboration with the European H2020 *i3-Market project.* Intelligent, Interoperable, Integrative and deployable open source MARKETplace with trusted and secure software tools for incentivising the industry data economy [3].

Within this project, the development of a vault of secrets, such as a secure wallet on the cloud, is essential if we desire a trusted data-exchange environment backed up by the blockchain.

The developed solution, named Cloud Wallet, consists of three pieces of code, each of which, serves a specific purpose within the complete project:

- Frontend Application (CloudWalletApp)
- Client Library (CloudWalletClient)
- Backend Server (CloudWalletServer)

On the one hand, the frontend app allows the user to intuitively create and import Ethereum accounts in order to sign raw messages or Ethereum transactions. While the client takes care of the setup and communication with the server (among others) the app works as a UI wrapper made with the React framework for this client. To sum up, the app manages user interactions, renders the website and uses CloudWalletClient to manage blockchain and interact with the server.

On the other one, CloudWalletServer safely stores an encrypted representation of account credentials so that the client used on the application can rely on a secure credentials cloud backup to restore them back to client in case of incidences (i.e. lost device). Additionally, it uses Key Derivation Functions to deliberately slow password-check processes and make them expensive in terms of CPU and memory in order to discourage pre-computation attacks.

This project has been developed in conjunction with Alberto Miras Gil with our university and tutor's approval. Even though we have worked on the same project, while Alberto has focused his work on the Blockchain aspects developed exclusively at the frontend client, I have developed the security aspects of the complete solution, implemented on all frontend app, client and backend server.

Acknowledgements

Let me first extend this acknowledgement to every person who supported me during my academic career as engineer.

I would like to thank my tutor Juan Hernández Serrano for introducing me to the amazing project he is involved into. We have been able to successfully develop it, specially in this very unusual year of mobility restrictions due to the COVID-19 pandemic.

Furthermore, a great appreciation to my colleague Alberto Miras Gil, with whom I have also collaborated in the design and making of this project. Thanks for your relentless determination and expertise at every step of this project, specially on Blockchain topics.

Also, to conclude, special gratitude to my friends and family for their support and help during the development of this thesis.

Revision history and approval record

Revision	Date	Purpose
0	02/11/2020	Document creation
1	14/11/2020	Document revision
2	20/11/2020	Document final revision

Name	Email
Pau de la Cuesta Sala	pau.de.la.cuesta@alumne.upc.edu
Juan Hernández Serrano	j.hernandez@upc.edu

Written by:		Reviewed and approved by:	
Date	14/12/2020	Date	20/12/2020
Name	Pau de la Cuesta Sala	Name	Juan Hernández Serrano
Position	Author	Position	Tutor

1 Introduction

In recent years, digital world has been rapidly introduced into most aspects of human's everyday life, some experts state our civilization has dived right into the information age [2]. We can observe how Internet-enabled devices are increasingly being used at schools, workplaces, hospitals, universities. The Internet did not only pave the way for establishing long distance connectivity but it also moved part of our identity as individuals to this digital world.

As John Perry Barlow mentioned some years ago on his famous manifest *A Declaration of Independence of the Cyberspace* in opposition of government legislation on the Internet and in defense of net neutrality and a borderless digital world of universal access:

> *"Cyberspace consists of transactions, relationships, and thought itself, arrayed like a standing wave in the web of our communications. Ours is a world that is both everywhere and nowhere, but it is not where bodies live. [...] We will create a civilization of the Mind in Cyberspace"* [1]

Unfortunately, the current Cyberspace is quite different from the one Barlow described. Due to a generalized use of digital services, some problems in terms of identity privacy appeared online, and so did new protection mechanisms to restore and achieve confidentiality, integrity and availability for all online events.

Since the first blockchain 'Bitcoin' went public, thousands of 'alt-coins', or alternative coins, have been created to represent different digital assets with different algorithm and security levels [8]. The so called distributed ledger technologies (DLT) involved in cryptocurrencies aim, as the name states, to work as a ledger for digital assets in which there is not a trusted centralized party. Its most well-known subtype is blockchain, which registers these transactions of digital assets with a structure formed by a chain of blocks. These blocks are cryptographically encrypted and consensually approved so that this chain of blocks is immutable, meaning, any transaction committed to the blockchain can not be altered *a posteriori*.

i3 Market is a European project aiming to provide a trusted and secure data sharing mechanism for organizations using Ethereum blockchain to store data transactions. Cloud Wallet is created as a tool to interact with the i3 Market backplane which will be described later. The user is in control of his Ethereum accounts as the keys are stored on the App and has additional features such as signing messages or transactions. The system is resilient in case the device running the App is lost or stolen because Cloud Wallet securely stores an encrypted version of public/private keys on the Cloud Wallet Server, only allowing a new App instance with the valid credentials to correctly decrypt the keys when restoring these accounts to a new device.

1.1 Motivation

Not every user of Internet-enabled devices knows, or even understands, how or what the device is actually doing in order to interact with the millions of online services a lot of organizations provide across the world.

This gap creates a serious problem for the less knowledgeable users, which end up delegating this responsibility to their online service providers. Moreover, it has

become more usual for online services to require real-world data in order to function, thus creating a direct link between the digital identity and the real one.

The centralized paradigm for data ownership does not allow end users to effectively own and decide about their data. This may lead to severe privacy flaws such as: data mishandling by non-encrypted services, espionage by several trackers used by online service providers and even location tracking.

Even though there have been several legal initiatives to protect users online privacy (i.e. European GDPR, California Consumer Privacy Act, etc...), these legal initiatives are mostly focused on user consent but do not tackle the root cause or prevent the flaws described above as they are inherent to the centralized paradigm, simply because, this digital identity and its management does not reside on users' devices.

In this context and opposing the centralized paradigm, another prominent initiative is *self-sovereign identity* [9], a new approach to identity management tasks which rely on the blockchain working as an identity registry, it uses public/private key-pair authentication mechanism and Decentralized Identifier (DID) to bind an identification namespace to a public key thus defining an identity attached to the ownership of the public/private key-pair.

The decentralized paradigm is based upon entities (i.e. individuals, organizations, devices, etc...) which interact in a trustless ecosystem (such as the Internet) by means of claims (i.e. age, nationality, accomplishments, etc...). While these claims may vary in content, they always refer to a *subject*, are issued by an *issuer*, held by a *holder* and in order to give trust to the trustless environment, can be verified by a *verifier*.

Verifiable claims are essentially cryptographically signed and non-repudiable claims. Claims are statements about subjects, they are verifiable because a significant set of trusted attestators prove the verifiable claim is either issued by themselves or can prove the correctness of the claim with respect to the subject. This system provides scoped access to verifiable pieces of information about a subject in the shape of claims, a powerful feature which can be used to provide restricted access to the concrete information needed for a service provider (acting also as verifier) and allow the provided service to authenticate the claimed subject.

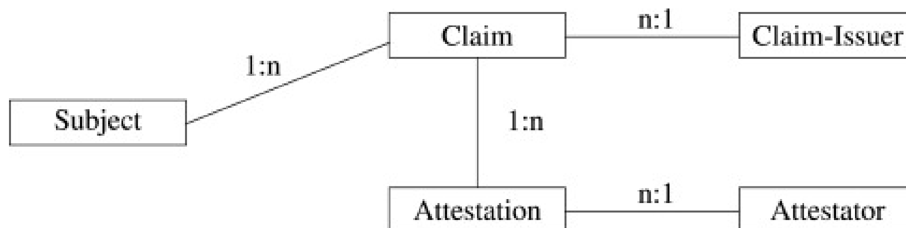

Figure 1: Verifiable claim components

For instance, an online company on a security audit sharing their data mining raw sources to an external auditor could make an access pass to the data stream by attesting a claim issued by the auditor to access the data residing in a third party storage provider (i.e: AWS S3, Dropbox, etc...). The auditor would show that pass and the data storage provider would verify the identity of the issuer and the

attestator and then return the data as long as the attestator was the data stream owner, as it would be considered a valid verified claim. In this case:

- Subject: Data ID for the thirdparty storage provider
- Claim-Issuer: External Auditor
- Claim: Time-Limited Access to Subject
- Attestator: Company being audited
- Verifier: Storage provider

A simpler example, a user at a shop needs to prove the legal age required to buy alcoholic beverages, the user would issue a non-expirable statement claiming his legal age and ask an attestator, the government, to sign it in order to prove that claim any time in the future. This user should only provide that verifiable claim in the shop and the shop would know that a trusted third party, in this case the government, attests that claim.

- Subject: Shop User
- Claim-Issuer: Shop User
- Claim: Subject's legal age
- Attestator: Government
- Verifier: Shop owner

1.2 Problem Statement

Cloud Wallet solves a series of problems inherent to the centralized paradigm. Moreover, it is developed to fit into the European framework for an open-source, secure and reliable digital data market called i3 Market [3].

The i3 Market project aims to provide a trustworthy and secure data sharing mechanism between organizations of different sizes. It is a solution towards present and future federation of diverse European data markets which is characterized by being:

- **Trusted**: secure, self-governing, consensus-based and auditable.
- **Interoperable**: semantic-driven
- **Decentralized**: scalable

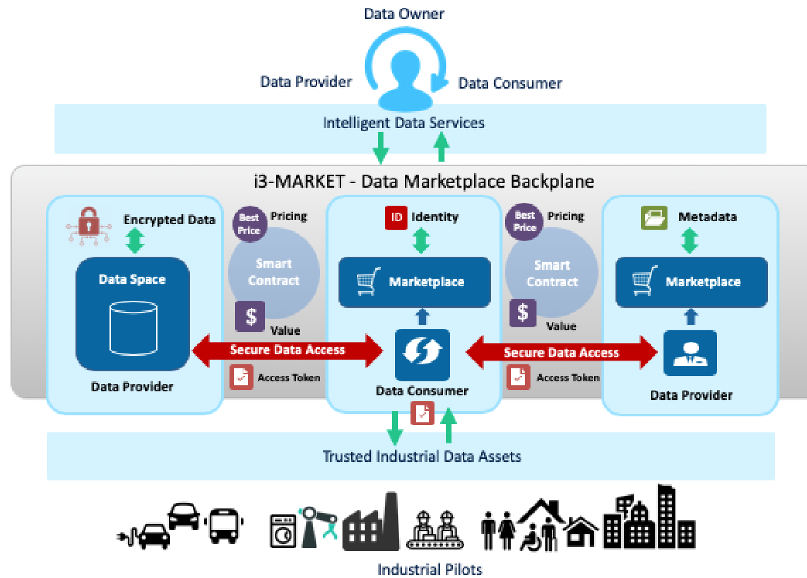

Figure 2: i3 Market Backplane Architecture

As seen above in Figure 2, i3 Market relies on the blockchain in order for it to fulfil the aforementioned *trusted* characteristic.

Cloud Wallet allows its users to effectively own and manage their sensitive data assets by providing a secure storage for their blockchain accounts' credentials. Blockchain accounts are unlocked by public/private keypairs which provide the owner of these keys the capability of, for instance, sign verifiable and auditable transactions of data assets and commit them to the blockchain to remain immutable.

In addition to blockchain transactions for the data market, any other type of text-based message can be encrypted and signed using these accounts keypairs hosted at the secure Cloud Wallet application.

Finally, a lost key prevention design allows its users to securely recover these keypairs in case of loss or thief. The mechanism is secure with state-of-the-art cybersecurity defense techniques later explained in this document.

1.3 Methodology

The development of this project is characterized by 4 different phases. On Figure 3 the Gantt chart of the project is shown:

- Design.
- Backend Implementation.
- Frontend Implementation.
- Documentation.

This project began phase 1 and 4 by mid-July 2020. In the first phase, the general structure and project planning were developed as well as a previous implementation tool study to decide which technological tool to use in order to build our solution.

Phase 2 was focused on the back end server of the Cloud Wallet, specifically, on its architecture and implementation. Phase 3 was focused on the same aspects for the front end app of the Cloud Wallet solution.

Phase 4 was focused on documentation, the drawing of sequence diagrams of the solution and documentation followed by the writing of this document and the preparation of support media for the presentation day.

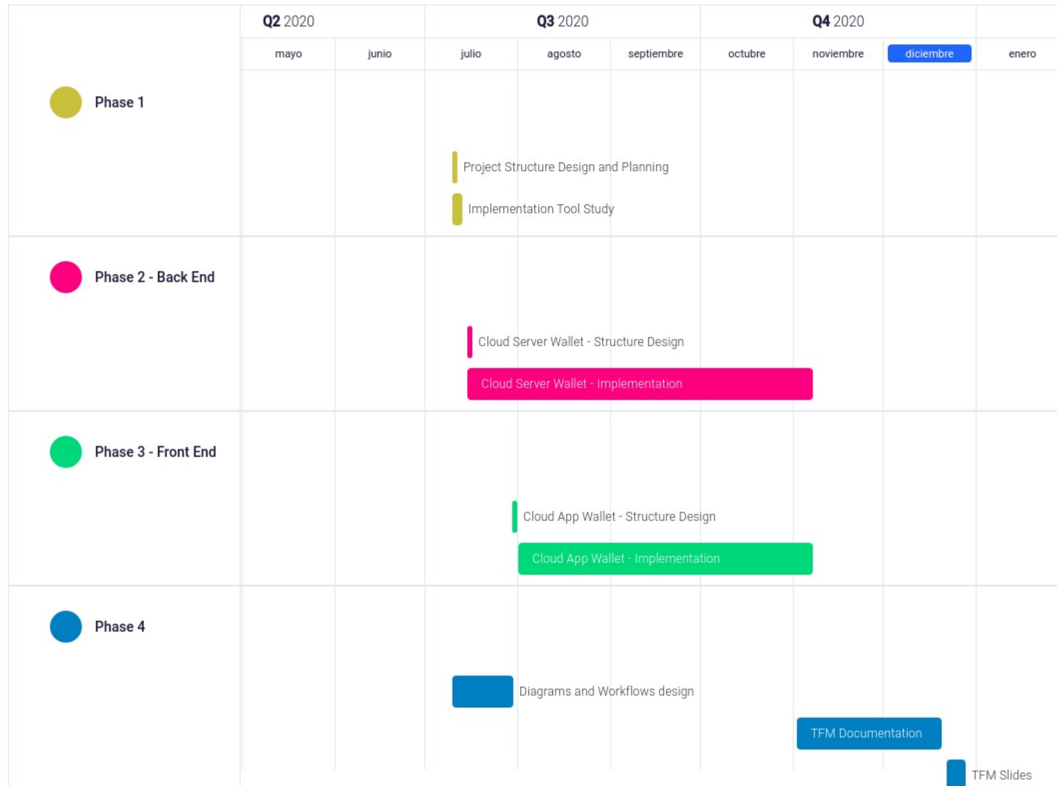

Figure 3: Project Gantt Chart

1.4 Cybersecurity Approach

Generally, online services rely on user's ability to secure their account by means of a robust password. Nonetheless, this is generally not the case because the longer these passwords are, the easier it is for a user to forget them. Another usual behaviour is using the same weak password across different services, in case of a data breach on one service where weak passwords are revealed, it would be easier for an attacker to test these same passwords in other online services.

Cloud Wallet does not rely on users robust passwords, but rather assumes user passwords are generally weak, and makes them robust by means of password-based key derivation functions. Our approach relies on making brute-force attacks unfeasible by deliberately making the computations slow and expensive both in terms of CPU and memory. While a valid user will see no difference as of performance, an attacker without the proper password will have to spend a vast amount of resources to be able to crack Cloud Wallet passwords.

2 Related Work

In this section we are going to describe some concepts used in Cloud Wallet in order to better understand the solution proposed.

2.1 Ethereum

A blockchain is a kind of peer-to-peer decentralized and distributed database where the data can not be altered once stored. It is a shared, trusted and public ledger that saves immutable and encrypted transactions across different nodes. It is made of blocks linked between them by cryptographic means. It is because of the cryptographic relationship between blocks that the chain can not be modified.

- **Decentralized**: No centralized trusted party.
- **Cryptographically secure**: Cryptography is used to improve the security and the integrity of the network.
- **Immutable**: Each block has a cryptographic relationship with its predecessors making it computationally infeasible to modify once accepted and stored.
- **Distributed**: Each node has a full copy of the ledger.

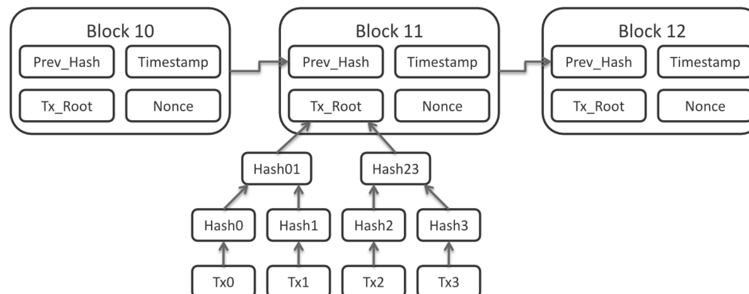

Figure 4: Blockchain cryptographic relationship

As a decentralized system, in this case a distributed ledger, single point of failure is avoided because every node has a copy of the ledger or blockchain; so it behaves like a decentralized database for digital assets. As there is not a centralized source of truth, no user is trusted more than another and consensus mechanisms are used in order to agree on adding or not new blocks to the shared ledger. The most used consensus algorithm is called Proof of Work (PoW), there is a hardware processing competition among nodes to validate new entries, called mining. Once a miner wins the competition, the candidate block is accepted and stored across the network (thus, appending a new block to the chain) and the miner receives a reward for the computational work.

Ethereum is often described as “the world computer”, a globally-decentralized computing infrastructure that executes programs called smart contracts. Smart contracts decide through their logic whether or not a certain transaction is valid and what is the resulting modification in the ledger.

More technically speaking Ethereum is a deterministic but practically unbounded state machine, consisting of a globally accessible singleton state and a virtual machine that applies changes to that state. The system's state is shared and synchronized among the Ethereum nodes by using a blockchain.

In order to incentivize Ethereum use and to pay for the costs of executing smart contracts, Ethereum also has a it own cryptocurrency, which is called ether.

2.2 Cyberattacks

There are many kinds of attacks, both online and offline, to exploit system's vulnerabilities. In this section we are going to see the two main types of cyberattacks which could present a threat to Cloud Wallet.

2.2.1 Brute-force Attacks

This type of attack aims at correctly guessing some user's password by means of an exhaustive test of candidates.

Pre-computation attack, also known as dictionary attack, is a specific brute-force attack in which an attacker prepares long lists of candidate passwords and its associated hashes. In case of a database exposure, a sufficient long list with a vast amount of candidate pre-computed hashes would almost instantaneously find the correct password.

Custom Hardware attack, on the other hand, relies on custom hardware specifically crafted to hash candidate passwords at a high rate. So, instead of pre-computing hashes, the attacker uses machines under his control (such as ASICs or FPGAs) which are specially designed to crack passwords.

2.2.2 Impersonation Attacks

Another relevant type of online threat are impersonation attacks. An attacker tries to deceives a user by making him believe it is communicating with another trusted user.

Man-in-the-Middle (MITM) attack is a specific impersonation attack in which the attacker secretly eavesdrops and relays the communication between two trusted parties while these interaction happens. While the original connection between the two users is not used anymore, all the traffic is relayed by the attacker and can potentially be altered.

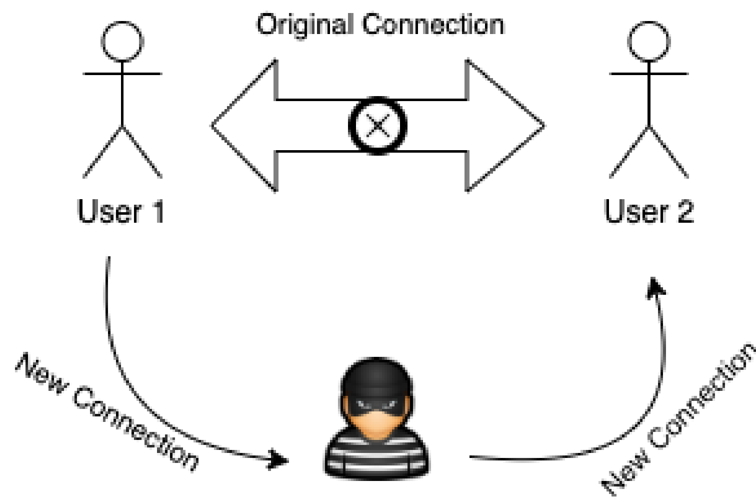

Figure 5: Man-in-the-Middle Diagram

2.3 Key Stretching and Key Derivation Functions

Key stretching is a set of techniques to create a strong cryptographic key from a low entropy input password or passphrase. Usually using salts to prevent pre-computed dictionary attacks, it outputs keys of different lengths by deriving the original key with a salt and password, passphrase or similar. Key stretching can be accomplished by using key derivation functions as it is described in Figure 6.

A Key Derivation Function (KDF) is a function which constructs a cryptographic key from a password using Pseudo-Random Functions (PRF). The PRFs add randomness to the key derivation process making the result efficiently indistinguishable from a truly random generated key.

Most common implementation for PRFs are keyed-Hash Message Authentication Codes (HMAC) that produce a cryptographic digest of the input based on a key and a cryptographic hash function. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and the size and quality of the key.

Quoting the author of Scrypt [11]:

> *"Since all modern key derivation functions are constructed from hashes against which no non-trivial pre-image attacks are known, attacking the key derivation function directly is infeasible; consequently, the best attack in either case is to iterate through likely passwords and apply the key derivation function to each in turn."*

Knowing that the best strategy to crack a password is iteratively hashing the input candidates until a match is found, there rises the necessity of discouraging an attacker to test keys excessively fast. By using key stretching, longer keys take longer time to process and that is of paramount importance to discourage massive brute-force attacks.

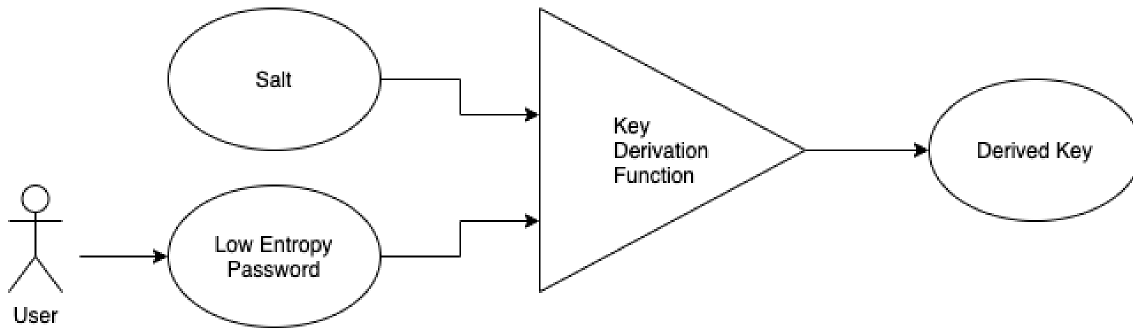

Figure 6: Key Stretching deriving a strong cryptographic key

2.4 PBKDF

Password-based key derivation function (PBKDF1 and PBKDF2) are two implementations of Key Derivation Functions defined in RFC2898 [6].

We can control the iteration count by means of the parameter c, adding more iterations will make the function run slower. It may seem secure enough but it still is not, attackers could take advantage of the easy and cheap implementation of this function and use Application-specific integrated circuits (ASIC) or graphical processing units (GPU) to brute-force as many iterations c as it is required as long as enough CPU and RAM can be provisioned [11].

- **P**: Password as an octet string
- **S**: Salt as a PRF-generated octet string
- **c**: Iteration count
- **dkLen**: Derived key length, positive integer, at most $(2^32 - 1) * hLen$, where hLen is the length in octets of the pseudorandom function output

It outputs a derived key dk of length $dkLen$.

2.5 Scrypt

Scrypt was born as an alternative of PBKDF2, it is also a Key Derivation Function (KDF) but unlike PBKDF2, scrypt offers resistance to Application-specific integrated circuits (ASIC) and graphical processor units (GPU) brute-force attacks [11].

It can be parametrized to make custom hardware attacks very expensive by requiring a vast amount of memory.

- **N**: Is the working factor (CPU/memory cost) *Default: 16384*
- **r**: Is the blocksize for sequential reading performance fine-tuning. *Default: 8*
- **p**: Is the parallelization factor. *Default: 1*

The tuning of these parameters modifies the time and resources spent for the calculations and can be adapted for different environments and adversaries. Given the different KDFs and parameters lead to different amounts of time and resources spent while deriving keys, we can see a cost estimation in dollars for an attacker to brute-force a password on Table 2.5. These times have been estimated with a 2,5GHz Core 2 processor [10] and take into account two main usages of KDF, interactive logins (≤1s) and file encryption(≤5s).

KDF	6 letters	8 letters	8 chars	10 chars
MD5	1	1	1	1.1k
PBKDF2 (100 ms)	1	1	18k	160k
scrypt (64 ms)	1	150	4.8M	43B
PBKDF2 (5.0 s)	1	29	920k	8.3B
scrypt (3.8 s)	900	610k	19B	175T

Table 1: Estimated cost of hardware in $ to crack a password

2.6 Password-Authenticated Key Exchange

OPAQUE is an asymmetric Password-Authenticated Key Exchange (aPAKE) protocol[5]. OPAQUE provide a secure method to authenticate two parties without needing certificates and their Public Key Infrastructure (PKI), it is able to securely verify users' identity just with a password (*password-only*).

In most of online services, servers have a database of hashed passwords obtained from hashing the users' registered passwords with user-specific nonce values, called salts. When a user logs in the system the username and password are sent in cleartext on a secure TLS client-server connection. Then, the server hashes the received password with the salt stored for that user, and checks whether it matches the stored hashed password.

Unfortunately, TLS security may be undergone in several situations. Some examples are: rogue servers operating with leaked private keys; client software that does not verify certificates correctly; users that accept invalid or suspicious certificates, certificates issued by rogue CAs; servers that share their TLS keys with others, e.g. CDN providers or security monitoring software, information (including passwords) that traverses networks in plaintext form after TLS termination; and more[14]. OPAQUE offers two main advantages over password-over-TLS and current aPAKE authentication protocols, respectively:

- May not use TLS even though it is recommended to combine both. Registration is the only step that requires an authenticated channel (out-of-band, PKI-based...)

- Resolves pre-computation attacks vulnerability, even if an attacker could access the server database, it would not be able to perform a brute-force attack as it does not store password hashes but only user salts and client-encrypted blobs of data.

This new protocol is meant to be *oblivious* as it is constructed with Oblivious PRF functions (OPRF). While these *oblivious* concept may seem new and flashy, it

has actually been referenced and studied since 1981 by Rabin on his 1-2 Oblivious Transfer protocol based upon RSA cryptosystem [12]. OPAQUE authentication protocol uses DH-OPRF composed of an OPRF and a Diffie-Hellman key-exchange protocol. An OPRF is a pseudorandom function family (PRF) H with an associated protocol between a server that holds a key for User kU and a client with a password $passwd$. It is oblivious because at the end of the protocol execution, the client learns the PRF output $H_{kU}(passwd)$ and nothing else, and the server learns nothing about the $passwd$.

The user runs OPRF with the server to compute $decryptKey = H_{kU}(passwd)$, then runs the Diffie-Hellman key exchange protocol with the server by using the key-pair decrypted with $decryptKey$. By using this previously generated key, the server is able to verify users identities without the need of PKI while not learning anything about $passwd$.

In Figure 7 the OPAQUE protocol is depicted with some of its parameters: a hash function $AuthEnc$ (e.g., SHA2 or SHA3 function) with 256-bit output at least, a cyclic group G of prime order q, a generator g of G, and a hash function H mapping arbitrary strings into G (where H is modelled as a random oracle). After OPRF, Diffie-Hellman Key Exchange would be executed with the learned keys.

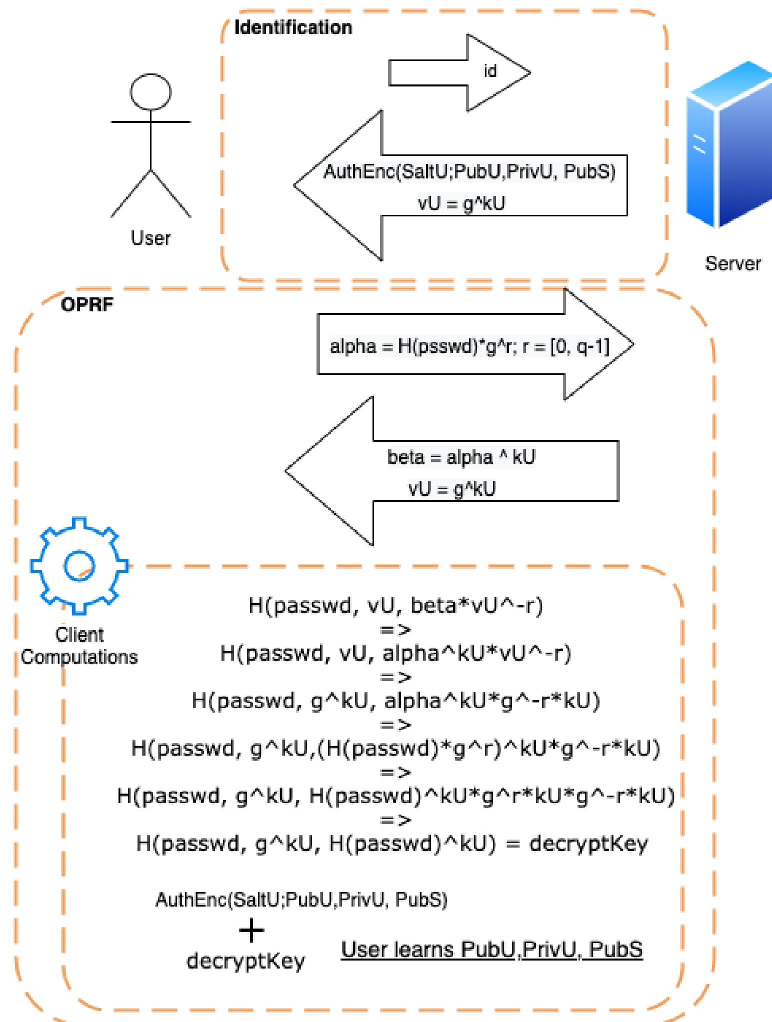

Figure 7: OPAQUE Diagram

3 High-Level Architecture

In this section, a high-level explanation of Cloud Wallet architecture will be done.

As can be seen in Figure 8, the *Cloud Wallet* architecture is based on two main blocks:

- **App**: working as cryptocurrency wallet which was done by Alberto Miras. A brief explanation will be given later in this document.
- **Server**: working as a vault of secrets which I designed and implemented. An extended explanation will be given later in this document.

In the diagram, we can also see other blocks that interact with *Cloud Wallet*:

- End User: client who is browsing the *Cloud Wallet* user interface.
- i3-Market SDK: the *i3-Market* ecosystem will interact with our application by requesting the signature of blockchain transactions.
- Blockchain: the application will have to keep in touch with the blockchain in order to be able to use some of its cryptographic functions and to retrieve some information about the users' blockchain accounts (as we will see later on in other chapters).

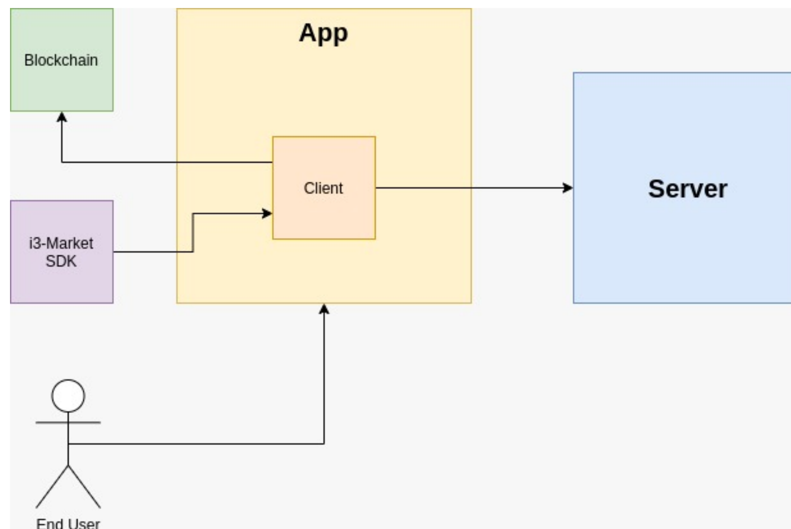

Figure 8: High-level diagram of the application

4 Frontend Application

The frontend application works as the visual interface for users to interact with the Cloud Wallet. It contains the *cloud-wallet-client* which, in turn, is the part interacting with the blockchain and the Server, i3 Market SDK itself will interact directly with this client. The frontend application is a wrapper for *cloud-wallet-client*.

4.1 Browser Extension vs SPA

An initial tool study was done in order to evaluate the suitability of two well-known platforms to distribute the frontend application. The two options were either building a Browser Extension (BE) or a Single Page Application (SPA).

In either case, React web framework was the most suitable framework to use considering it is open-source and maintained by both the community and several important companies, also because of its fast bootstrapping tool *create-react-app* and due to its seamless integration with the rest of software used in the i3 Market project, which is hevalyly develop with Javascript. Moreover, React can easily switch from one to the other option, so in case of a change in the distribution platform, most part of the software would still be usable.

We finally decided to build a Single Page Application contrarily to the Browser Extension option due to maintainability and security reasons.

Browser Extensions can potentially become a threat from a privacy perspective. They have access to more browser APIs than websites, even capable to modify and capture sensitive information from the browser storage and configuration. In case of a remote malware hijacking our frontend BE, a higher impact can be done because of this access to more capabilities than the application on a SPA.

Also, it is harder to maintain code for some specific extension APIs that differ across browsers, it is more maintainable to develop a web application built as a SPA with community-tested polyfill software that outputs the same visualization of User Interface (UI) across all browsers.

4.2 Client Library

CloudWalletClient is a Javascript client library of the system. This local wallet connected to the blockchain and the CloudWalletServer allows the user to securely store its Ethereum accounts, which are public/private keypairs.

This library is conformed by three modules:

- Authentication Module: store and validate credentials or interact with User entity on Server
- Blockchain Module: interactions with Blockchain.
- Resource Module: interactions with Resource entity on Server

It exposes some public methods which other applications can use in order to interact with the system:

- Create new accounts

- Import existing accounts
- Sign text-based messages and Ethereum transactions
- Check accounts balance
- Save and Recover accounts from CloudWalletServer

4.3 Example Application

CloudWalletApp is a React web application which offers a basic and intuitive UI wrapper for the CloudWalletClient library. It is a visual resource to access the public methods exposed by the library. The example application can be found at *https://cloudwallet.gold.upc.edu*

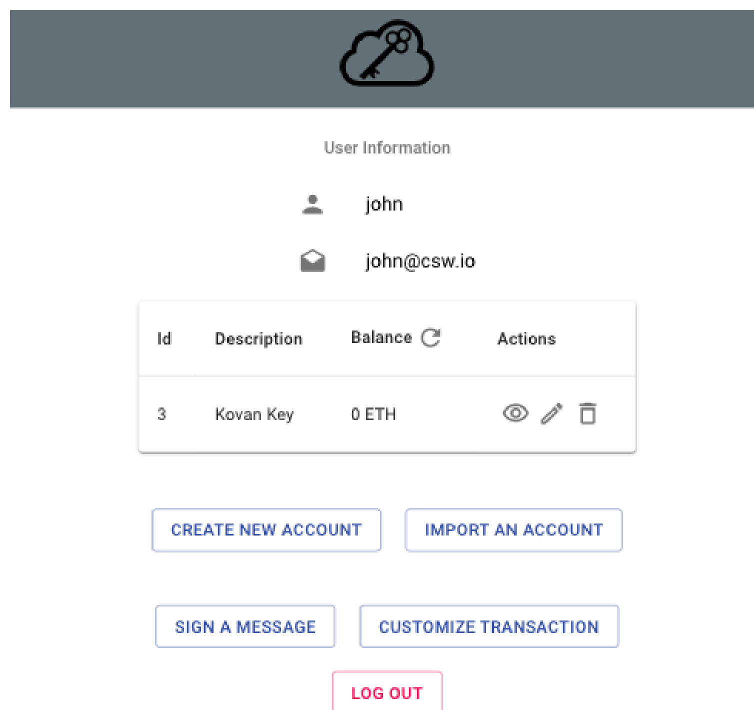

Figure 9: Cloud Wallet App main page

On Figure 9, a screenshot of the main page for an authenticated user. It visually allows a user to view public key, edit descriptions or delete resources for all his keys. Moreover, several available actions are clearly displayed on the buttons at the bottom.

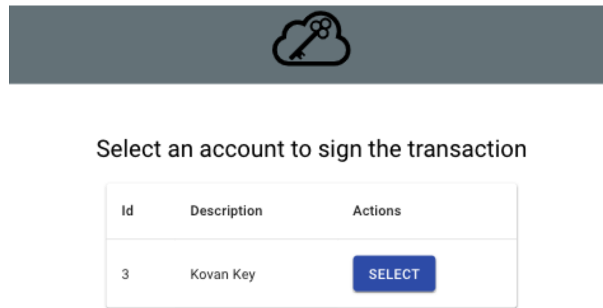

Figure 10: Cloud Wallet App key selection page

On Figure 11, the signing page is shown with the parameters needed to create a new transaction. Destination account, amount to transfer and the name of the chain to use. i3 Market may operate over different blockchain networks, the SDK provides the parameter to build a proper transaction. The signing key can be selected at the previous page referenced on Figure 10.

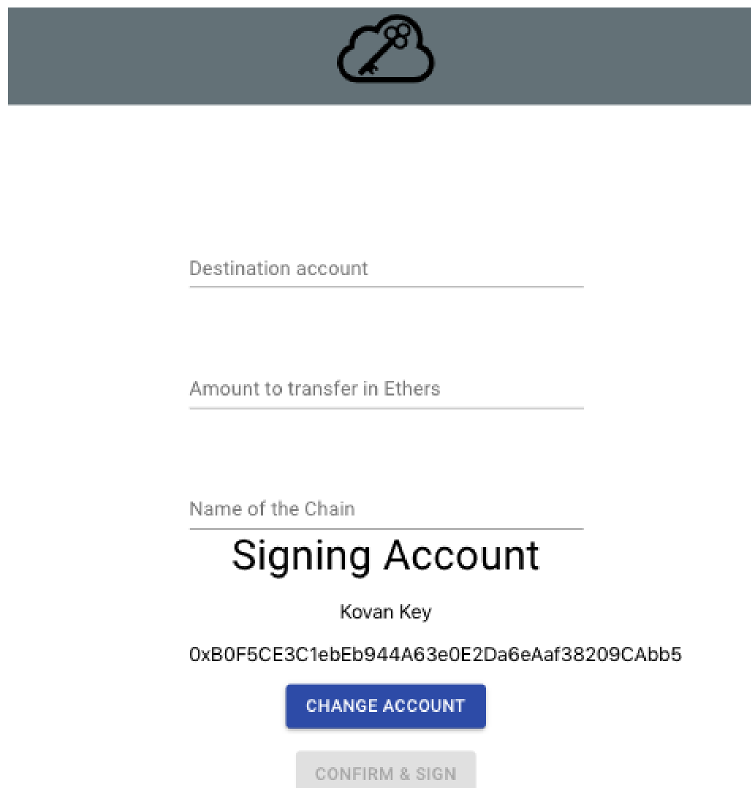

Figure 11: Cloud Wallet App signing page

5 Backend Server

CloudWalletServer is a backend server working as a secrets vault implemented in NodeJS, specifically using the NestJS framework. It allows Cloud Wallet users to safely store their public/private key pairs of their Ethereum accounts. As it is later discussed, security has been the most critical topic during the design and development of the technology.

The code is hosted at GitLab in a private repository visible to i3 Market task-force members. *https://gitlab.com/i3-market/code/wp3/t3.2/cloud-wallet*

5.1 Infrastructure

Cloud Wallet Server can be accessed via web at a server hosted by our University with different software installed.

Docker is a virtualization tool aimed at isolating application software into *containers*. In conjunction with *docker-compose*, a network of containers can be created and coordinated to deploy a complete technology stack.

This project's infrastructure is composed of two containers:

- CloudWalletServer: builds the source code and starts the server.
- PostgreSQL: starts a PostgreSQL database.

Even though at production-grade deployments of PostgreSQL containers should be avoided (as much as any kind of containerized database), it is very helpful at a development stage. In production, open to end-users, the dockerized server should establish connection with an external database. In i3 Market this database is going to be a CockroachDB [7], a cloud-native, distributed database.

5.2 Public API

The server exposes an Application Programming Interface (API) in order to authenticate clients and interact with their vault. Server's publicly exposed HTTP endpoints are represented with entities, *user* and *resource*:

User

- **GET /user/login**
  → Get login fields [”username”, ”password”]
- **POST /user/login** {username, password}
  → Login with request body {username, password}
- **GET /user/signup**
  → Get signup fields [”username”, ”email”, ”password”]
- **POST /user/signup** {username, email, password}
  → Sign up with request body {username, email, password}

Resource: Authenticated with Bearer JWT

These endpoints require an Authorization HTTP header with a value of 'Bearer *validToken*', where *validToken* must be stored at device's localStorage from a previous request to *POST /user/login* or *POST /user/signup*.

- **GET /resource/list**
  → Get user's resource list
- **POST /resource** {address, value, description}
  → Create a new resource
- **PUT /resource/{id}** {Optional(description), Optional(value)}
  → Update an existing resource with {Optional(description), Optional(value)}
- **GET /resource/{id}**
  → Get user's resource with {id}
- **DELETE /resource/{id}**
  → Delete user's resource with {id}

Cloud Wallet Server is documented conforming to the OpenAPI Specification 3 standard as it will be later seen on a dedicated section of the chapter. By using OAS3, both humans and computers can easily understand public APIs of millions of services.

5.3 Secrets Vault

While Cloud Wallet Server is designed to be a general purpose secrets vault, it specifically acts as a vault to users' Ethereum accounts storing a representation of their private keys. The Server holds an encrypted copy of the private key, so that only the legit owner is able to decrypt it. By doing this, the server will not be able to know the value stored in it, it just returns it back to the legit client in order for it to decrypt it.

This measure does not prevent offline attacks in case an attacker would steal and download the server database and had unlimited time to try cracking passwords offline. For these cases, the Server includes a special key stretching derivation extra step for which the usual weak passwords introduced by users is enhanced with *scrypt* KDF in order to discourage offline brute-force attacks by making them overly expensive.

In the i3 Market context, all the signing and connection to the Blockchain is performed at the client application, for that reason, data owners need to have a safe vault for their market access credentials in case the device (smartphone, laptop, etc...) used to operate in the market (which has the access credentials in it) is stolen or lost. It is necessary to store them securely on the cloud to be able to recover them on the future in another device. But, as previously mentioned, the Server does not store the private key by itself, but rather a client-encrypted version of it so that in case of authentication recover, only real users will be able to provide the right password to decrypt the credentials and use them.

5.4 Security Aspects

Best practices for online security have been implemented on Cloud Wallet Server such as HTTPS connections through TLS, JSON Web Token (JWT) authentication mechanism and a derivation process when signing up and logging in or when creating, retrieving or updating a resource.

5.4.1 Key Stretching

Users tend to have weak, low entropy passwords and that may cause a serious problem for the security of the service. An attacker would easily be capable of performing a brute-force attack with a massive amount of passwords until eventually finding the correct one. In order to prevent this kind of attacks among others, the server uses 32 pseudorandomly generated bytes as salt both for user and resources within the server, that combined with the password and *scrypt* key-stretching algorithm, it makes these set of tasks deliberately slower and overly expensive even for state-sponsored attackers.

Cloud Wallet Server uses a new implementation, called *scrypt-pbkdf* made by Juan Hernández, tutor of this thesis and main collaborator of the i3 Market project. In the following table, a benchmarking comparison of *scrypt-pbkdf* Javascript library. It estimates derivation time for several working factors N. Test were preformed on an Intel Core i5-6200U with 8 GB of RAM running Ubuntu 20.04 LTS 64 bits, and with Node.js 14 LTS for server testing, and Chrome 83 Linux 64 bits as browser [13].

N	Browser (Chrome 83)	Node.js 14
$2^{12} = 4096$	$85ms \pm 10.66\%$	$12ms \pm 6.45\%$
$2^{13} = 8192$	$165ms \pm 4.47\%$	$23ms \pm 1.80\%$
$2^{14} = 16384$	$336ms \pm 2.65\%$	$47ms \pm 2.82\%$
$2^{15} = 32768$	$648ms \pm 1.93\%$	$94ms \pm 0.66\%$
$2^{16} = 65536$	$1297ms \pm 0.29\%$	$210ms \pm 1.81\%$
$2^{17} = 131072$	$2641ms \pm 0.36\%$	$422ms \pm 0.81\%$
$2^{18} = 262144$	$5403ms \pm 2.31\%$	$847ms \pm 0.81\%$
$2^{19} = 524288$	$10949ms \pm 0.32\%$	$1704ms \pm 0.70\%$
$2^{20} = 1048576$	$22882ms \pm 0.45\%$	$3487ms \pm 3.42\%$
$2^{21} = 2097152$	-	$7031ms \pm 1.06\%$

Table 2: Estimated derivation time on Browser and Node.js of different working factors N for *scrypt-pbkdf* implementation

The Server establishes three levels of difficulty on derivation processes. By following the table in order to find the right balance between security and usability, users do not feel like the app is stuck when it is performing some of the derivation tasks. These different levels can be set by users according to their needs.

Such levels are:

- **Low 1**: $N = 2^{18}$ Used for login and sign up
- **Mid 2**: $N = 2^{19}$ Currently unused

- **High 3 (Default)**: $N = 2^{20}$ Used for resource management

5.4.2 Authentication

Authentication mechanism relies on password-over-TLS strategy and JSON Web Token for both registering and entering the application, the token itself contains the following data for the client to use.

- **Id**: User ID to let know the client for subsequent interactions with the server.
- **Username**: To display in the application
- **Email**: To display in the application
- **Password Hash**: SHA-256 hash of the derived password to enable password checking on the client for several operations.

5.5 OpenAPI Specification

As described on the OpenAPI Specification itself [4]:

> *The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.*

CloudWalletServer is created to conform to this standard specification on its newest version, OAS3. With OAS3 webpage depicted at Figure 13, a visual, concise and clear picture of the publicly exposed API methods of the server can be used by either humans or machines to learn on how to interact with it. Adopting OAS3 is a perfect decision for this project because it aims to be open-sourced, so well-documented API with OAS3 could allow developers to easily test the API or even automatic generation of software in several programming languages to consume the API methods. The full OAS3 documentation can be found at *https://api.cloudwallet.gold.upc.edu/*

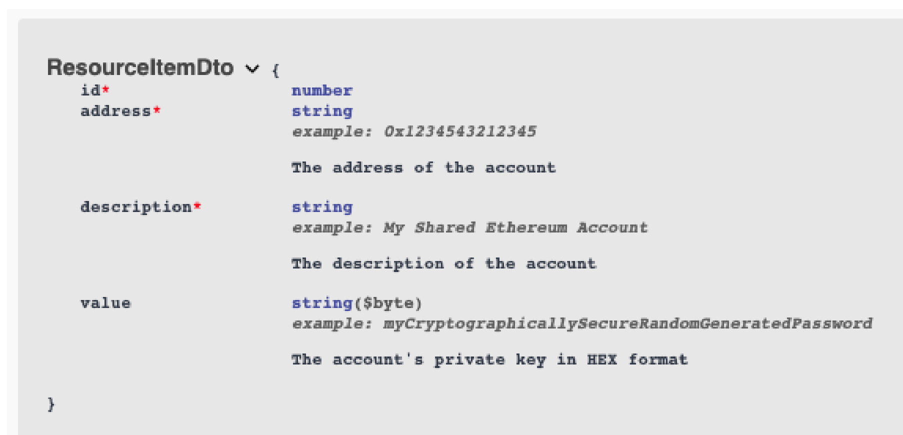

Figure 12: API Schemas excerpt from Cloud Wallet Server documentation website

resource

GET /resource/list Get user's resource list

Parameters

Cancel

No parameters

Execute Clear

Responses

Curl

```
curl -X GET "http://localhost:3000/resource/list" -H  "accept: application/json" -H  "Authoriza-
tion: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MywidXNlcm5hbWUiOiJwYXV1IiwiZW1haWwiOiJw-
YXV1QGNzdy5pbyIsImhhc2giOiIzNDU5YWYxMzIxN2E0ZTA3NTMzMzAyYWY1MjlkMmE0ZWFhMmUzMTIzNmFjZmYwZjE3NzlmYz-
JjM2NjMTc0ZDZmIiwiaWF0IjoxNjA5MTE2NjE5LCJleHAiOjE2MDkyMDMwMTl9.IlBw1Wj-
EIqDgTAblAzoxyvV2ZLNIe2t8Hf1o4RSkm8"
```

Request URL

```
http://localhost:3000/resource/list
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 2,
    "description": "Yeyeye",
    "address": "0x70845C56754AE20b4F5715c7a6BA0348174Ea6Ee"
  }
]
```

Download

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: http://localhost:3001
content-length: 88
content-type: application/json; charset=utf-8
date: Mon,28 Dec 2020 00:51:16 GMT
etag: W/"58-bVfTHpO+sStaRvAEmF3vyi83hOI"
vary: Origin
```

Responses

Code	Description	Links
200	OK.	*No links*

Figure 13: Server OpenAPI Specification 3 Website

6 Client-Server Interactions

In this section, sequence diagrams are presented describing the interaction between Cloud Wallet components and the End User.

6.1 Sign Up

Users need to register to the application in order to access to Cloud Wallet. Initially, the client requests (*GET /signup*) the signup form from the Server to display it to the user. Additionally, registration fields are rendered dynamically according to the needed fields for a valid registration as for the Server, this behaviour dismisses the Client from any responsibilities on the registration logic while putting the Server on full control of it.

After the user fills the form, another request (*POST /signup*) is sent from the Client to the Server with the required parameters in order to create a new user. In case of failure, an error message is sent back to the Client as response. Otherwise, the Server internally logs in that newly-created user and returns back a valid JSON Web Token *cw_token* as response.

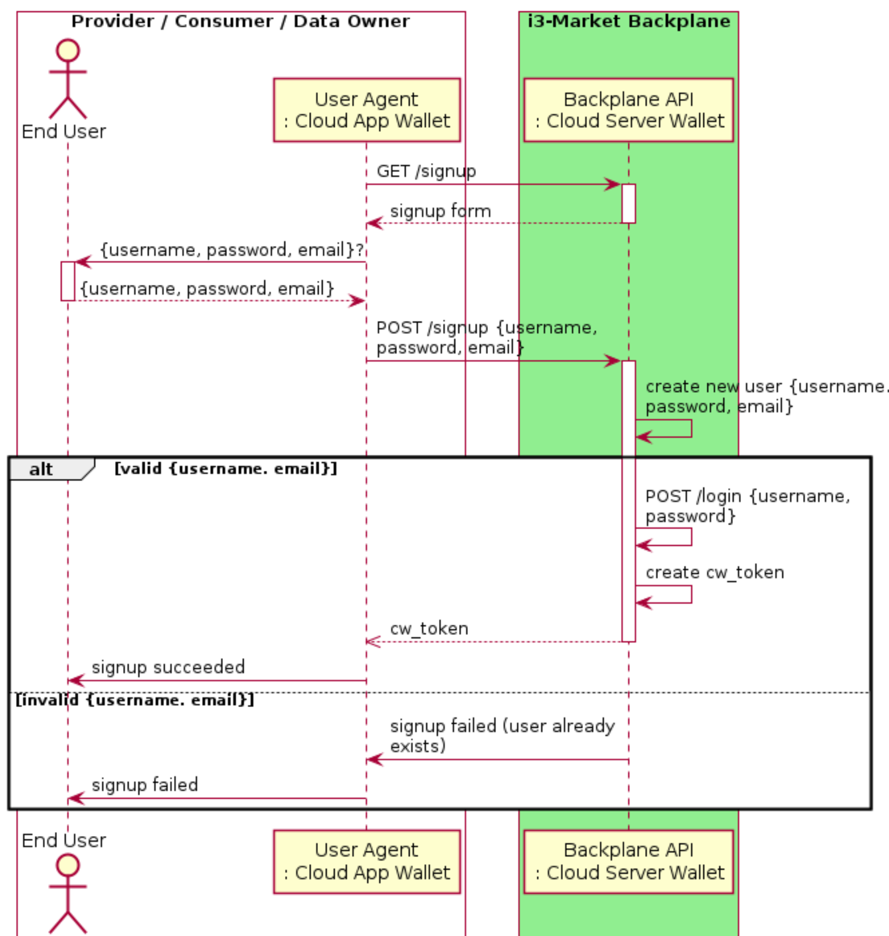

Figure 14: Sign Up diagram

6.2 Login

Similar to registration, login logic and fields are controlled by the Server, with an initial request (*GET /login*) the client gets the required forms for user input. After, the request (*POST /login*) is sent with the required data.

When server verifies user input, either a valid JSON Web Token or an error message is sent back as response.

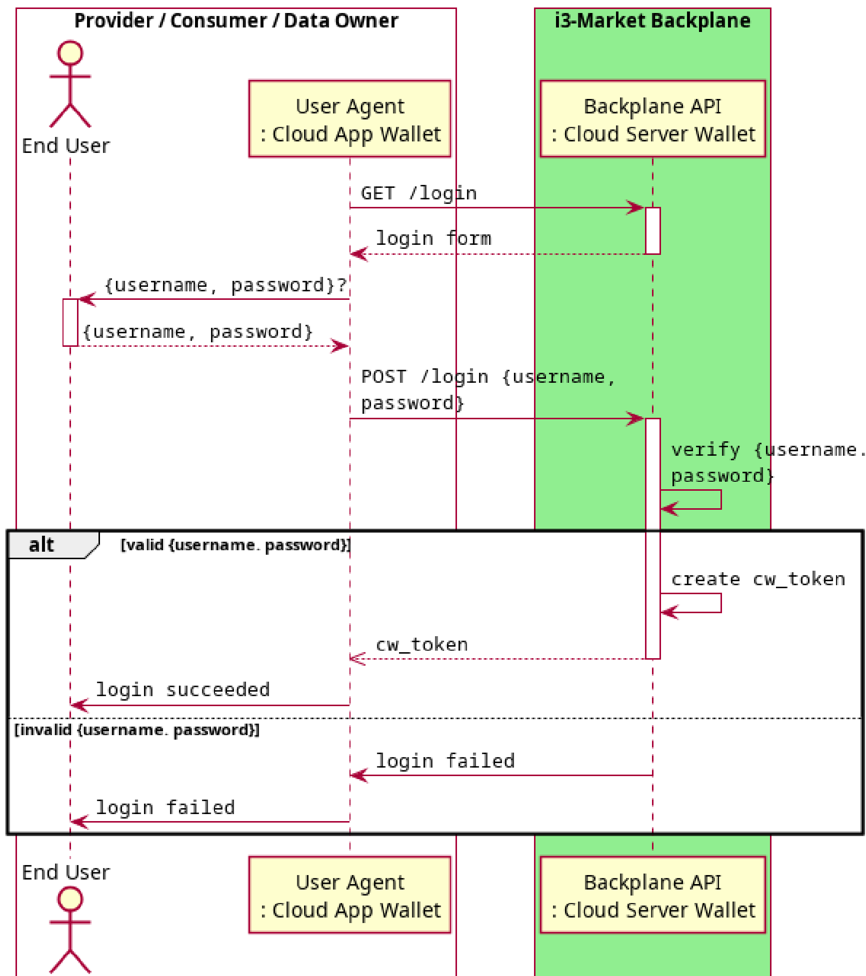

Figure 15: Login diagram

6.3 Create Resource

When an authenticated user wants to create a new wallet account (i.e. public/private keypair) the blockchain interaction is held at the client. After creating the keypair and encrypting its private key, it is sent to the server in a JWT authenticated request (*POST /resource*) along a description of that account.

When the server receives the information to store a new account, it will derive the encrypted private key, store the new resource record on database and confirm the creation to the client. In case of failure, it will notify the reason to the client with an error message.

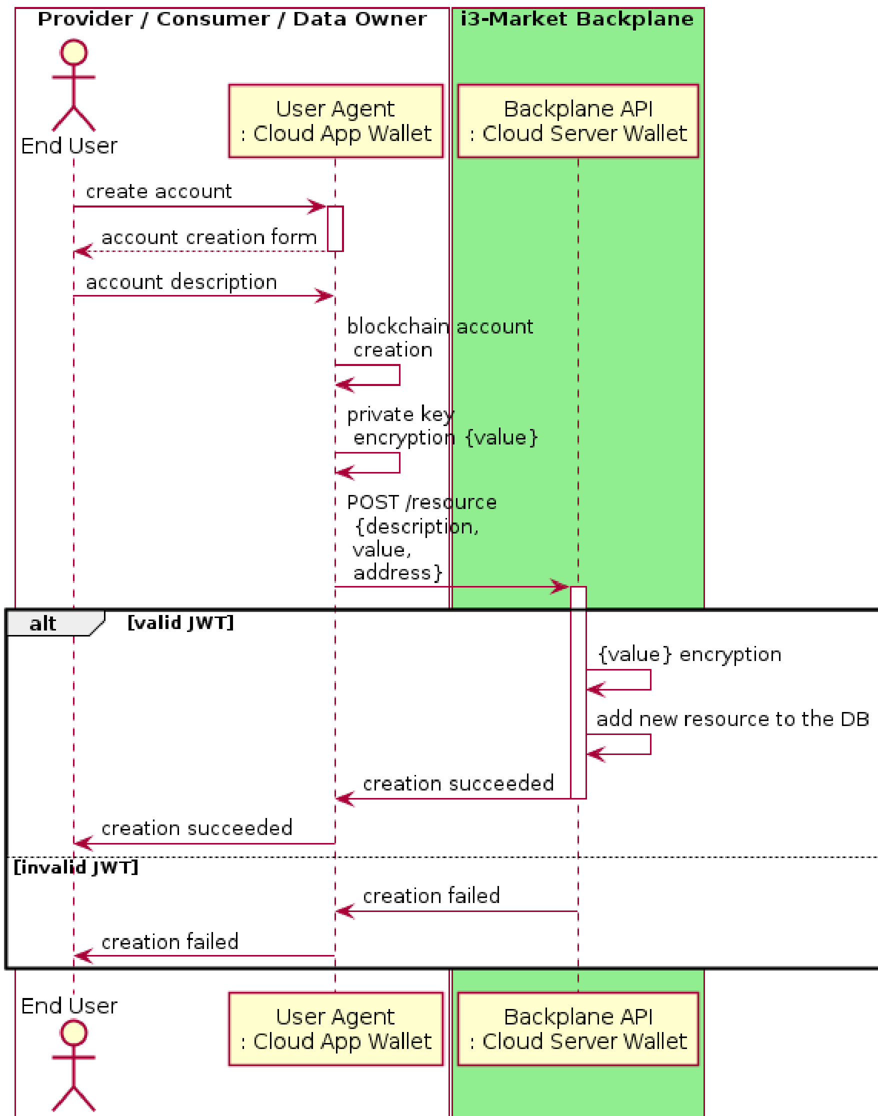

Figure 16: Create Resource diagram

6.4 Import Resource

Similar to resource creation, when a user wants to import an existing account (i.e. public/private keypair), an import form will be displayed for the user to input the description and private key of the account. After encrypting it and obtaining the public key on the client, the same process for request (*POST /resource*) seen on the previous section is done.

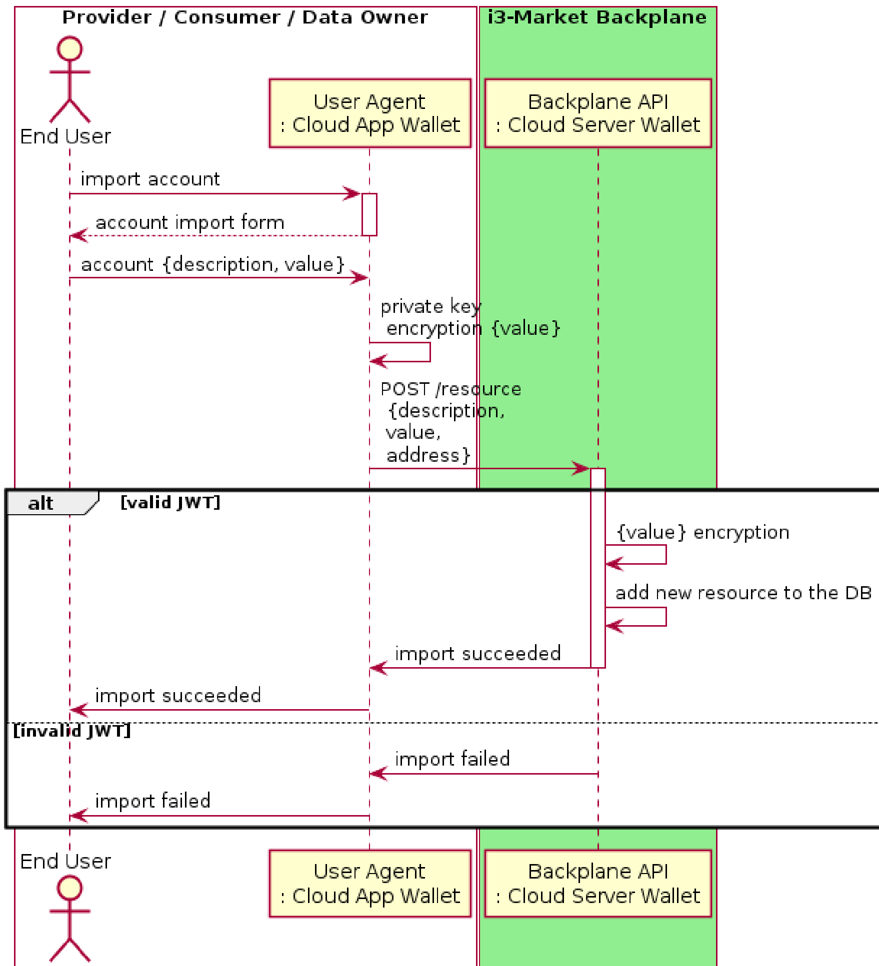

Figure 17: Import Resource diagram

6.5 Sign Message

Cloud Wallet also includes signing capabilities for raw messages and Ethereum transactions related to i3 Market. There are two similar procedures for this task depending on the pre-selection of the account used to sign the message or transaction.

- Sign with a specified Key ID on Figure 18
- Sign with an unspecified Key ID on Figure 19

When an authenticated user needs to sign an Ethereum transaction, the i3 Market SDK will redirect the user to the Cloud Wallet and send a request to the client with the account to be used, the callback URL to be redirected after signing and the transaction parameters. The transaction parameters specified from the SDK include:

- Destination address.
- Value/Amount to transfer.
- Blockchain network, as i3 Market can operate over several blockchains.
- Gas Price, it is optional and if not present, it will be calculated by Cloud Wallet.
- Gas Limit, it is optional and if not present, the lowest limit value will be used as in simple transactions.

The client sends a JWT authenticated request (*GET /resource/list*) in order to get a list of the public keys of all the stored accounts for that particular user. If the signing key was not specified in the beginning by the i3 Market SDK, the user will need to select an existing account to use for signing in the application.

When a valid key for that user is selected, another JWT authorized request (*GET /resource/id*) is sent to server in order to get the full resource. Then, the full transaction information is displayed to the user and explicit consent along with the password is required from him in order to decrypt the key used to sign. After decrypting it and signing the transaction, it returns the signature in the specified callback URL. The application also displays a link to *Etherscan* blockchain explorer, so that the user can verify the transaction block on this well-known tool.

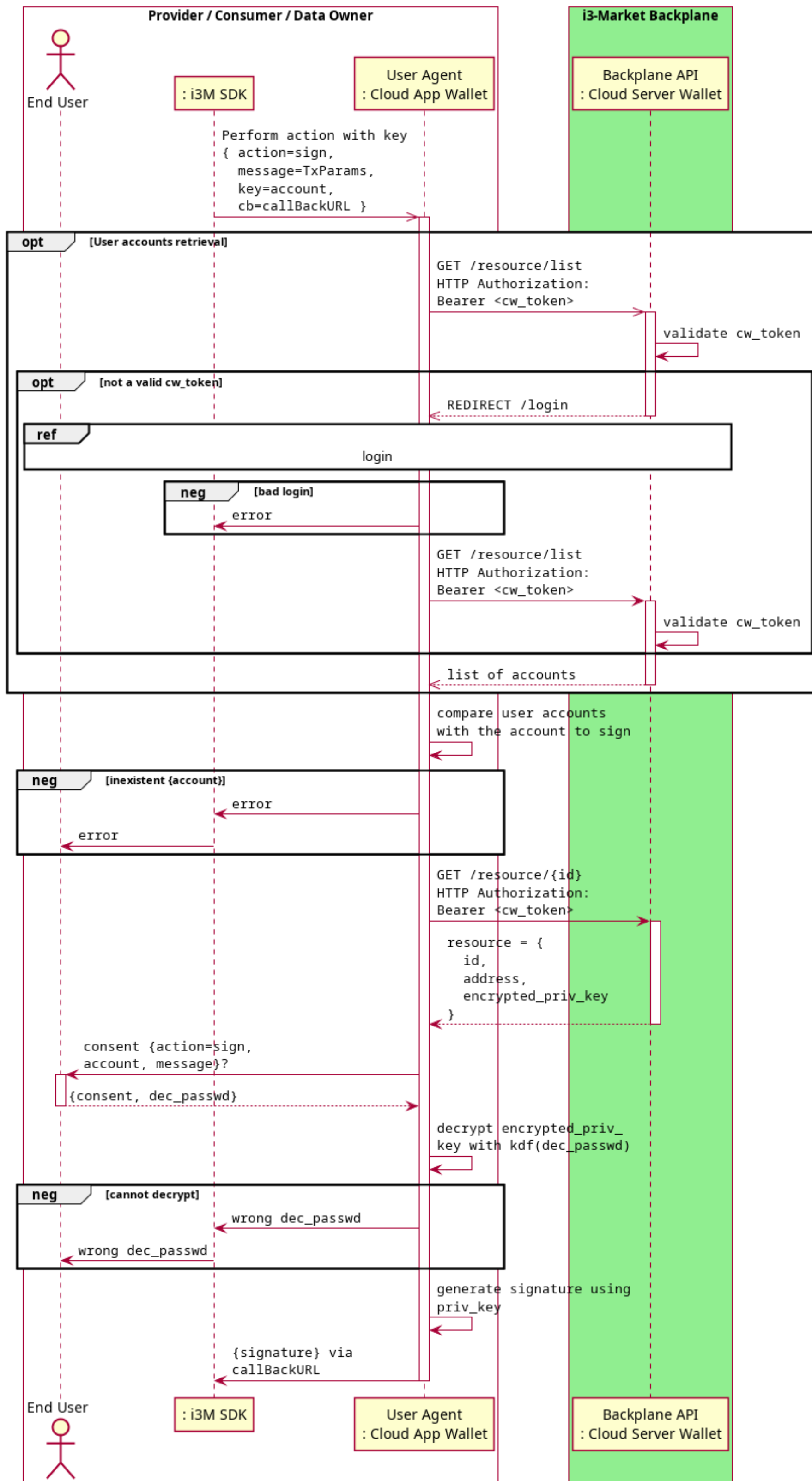

Figure 18: Sign Message with Key ID

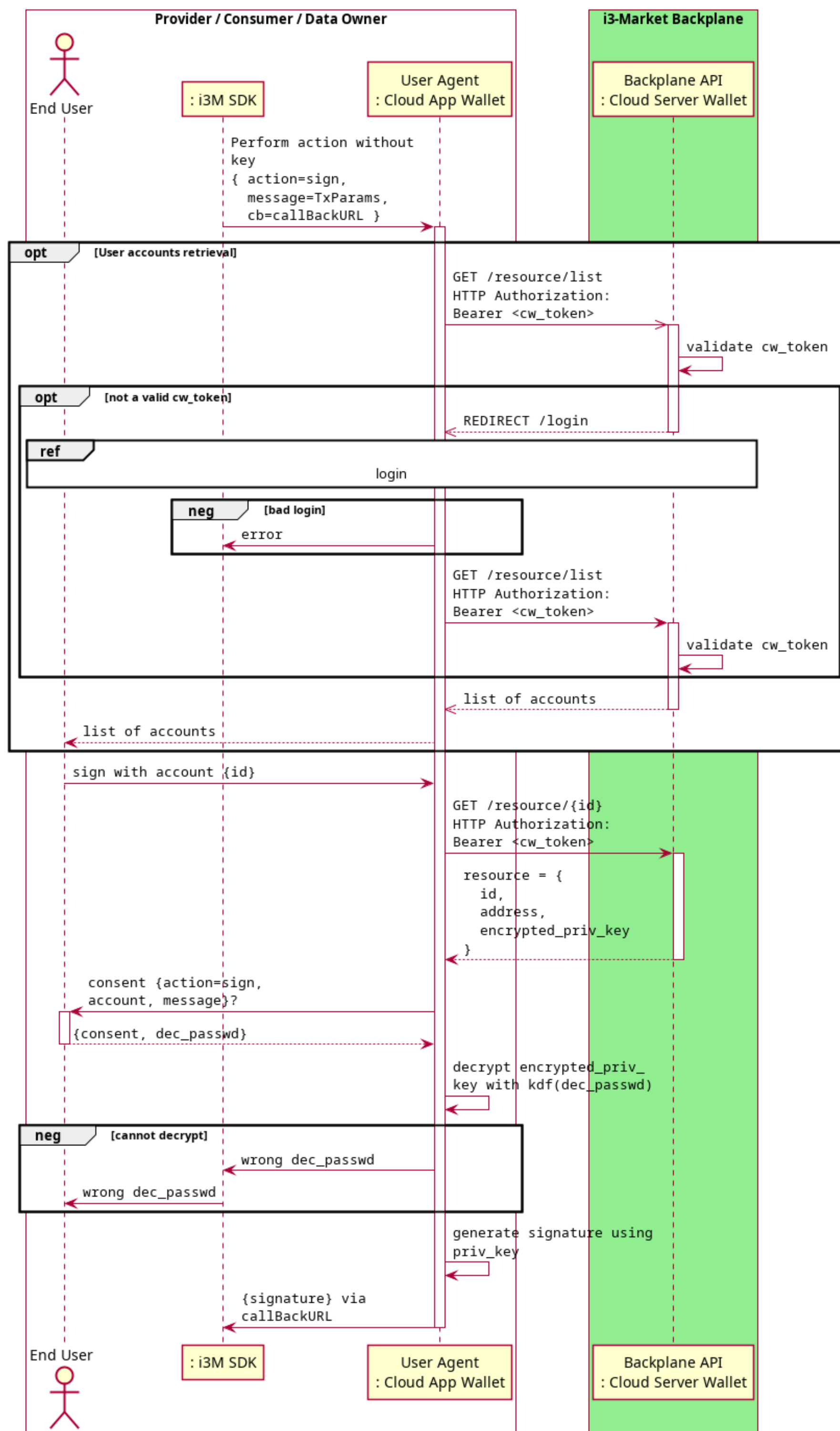

Figure 19: Sign Message without Key ID

7 Conclusions

Since online privacy is becoming of main interest to a growing number of population, it has become crucial to find alternatives to the current centralized paradigm for data ownership in which user data is watched over by organizations. Moreover, weak or similar passwords across different online services enable easier password cracking attacks, a password leakage from one of these services could lead to an impact on the rest of them with similar or identical credentials, potentially compromising the confidentiality, integrity and availability of users data.

i3 Market is a European project aware of these problematic. It aims to provide a trustworthy and secure data sharing mechanism between organizations of different sizes and a solution towards present and future federation of diverse European data markets.

Cloud Wallet enables i3-Market users to own and manage their blockchain accounts in a secure online vault with some basic cryptocurrency wallet functionalities. It is composed of 3 elements: Server, Client and App. While the i3 Market SDK will interact directly with the client, which establishes connections with the Server and the blockchain, the App works as a visual wrapper for users to use the Cloud Wallet.

In terms of security, it provides client-server authentication through TLS and JSON Web Token. Internally, it also uses Key Derivation Functions to deliberately slow password checks and discourage attackers to brute-force in. It is fine-tuned so that the user experience remains intact; while one single valid password check will see no difference, an attacker trying to brute-force it with custom hardware (ASIC, FPGA) will have to invest a vast amount of resources to potentially succeed.

Cloud Wallet is at an early stage, great improvements are yet to be implemented and deployed. Security is among the topmost priorities, that is why additional security measures need to be developed to protect online data transactions on the blockchain.

- OPAQUE would be a great improvement in order to guarantee a secure client-server connection not depending on PKI as well as to prevent pre-computation attacks.
- Multiple Factor Authentication (MFA) is another security feature which needs to be implemented in the system. With MFA, users will need additional steps besides password in order to authenticate (i.e Input code sent on SMS)

These two measures would add an extra layer of security to the presented solution and make cyberattacks harder to perform by hardening users' online accounts and securing their data assets so that they can trade safely on the i3 Market ecosystem.

Bibliography

[1] John Perry Barlow. *A Declaration of the Independence of Cyberspace*. 1996. URL: https://www.eff.org/cyberspace-independence.

[2] Manuel Castells. *La era de la información. Economía, sociedad y cultura. Vol. 1*. 1996. URL: http://www.economia.unam.mx/lecturas/inae3/castellsm.pdf.

[3] *H2020 - i3 Market Project Description*. 2020. URL: https://cordis.europa.eu/project/id/871754.

[4] OpenAPI Initiative. *OpenAPI Specification v3.0.0*. 2017. URL: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md.

[5] H. Krawczyk. *The OPAQUE Asymmetric PAKE Protocol*. 2020. URL: https://tools.ietf.org/html/draft-krawczyk-cfrg-opaque-06.

[6] RSA Laboratories. *PKCS #5: Password-Based Cryptography Specification*. 2000. URL: https://tools.ietf.org/html/rfc2898.

[7] Cockroach Labs. *CockroachDB Architecture*. 2020. URL: https://www.cockroachlabs.com/docs/v20.2/architecture/overview.

[8] J. Moubarak, E. Filiol, and M. Chamoun. "On blockchain security and relevant attacks". In: *2018 IEEE Middle East and North Africa Communications Conference (MENACOMM)*. 2018, pp. 1–6. DOI: 10.1109/MENACOMM.2018.8371010.

[9] Alexander Mühle et al. "A survey on essential components of a self-sovereign identity". In: *Computer Science Review* 30 (2018), pp. 80–86. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2018.10.002. URL: http://www.sciencedirect.com/science/article/pii/S1574013718301217.

[10] Colin Percival. *scrypt: A new key derivation function*. 2009. URL: https://www.tarsnap.com/scrypt/scrypt-slides.pdf.

[11] Colin Percival. *Stronger Key Derivation via Sequential Memory-Hard Functions*. 2009. URL: https://www.tarsnap.com/scrypt/scrypt.pdf.

[12] Michael O. Rabin. *How to exchange secrets with Oblivious Transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University*. 1981. URL: https://eprint.iacr.org/2005/187.pdf.

[13] Juan Hernández Serrano. *'scrypt-pbkdf' source code Repository*. 2020. URL: https://github.com/juanelas/scrypt-pbkdf.

[14] Hugo Krawczyk Stanislaw Jarecki and Jiayu Xu. *OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks*. 2018. URL: https://eprint.iacr.org/2018/163.pdf.