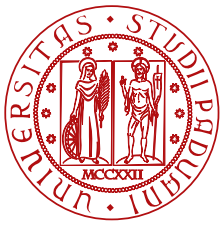

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

Spartizione di un'eredità tramite l'uso di blockchain

Tesi di laurea

Relatore
Prof. Sergio Canazza

Laureando
Raffaele Bussolotto, Matricola 1224718

ANNO ACCADEMICO 2023/2024
Data di laurea 14/03/2024

Per aspera ad astra

Ringrazio i miei genitori, Francesca e Damiano, mia sorella e mio fratello, Angela e Giovanni, e tutta la mia famiglia per aver sempre creduto in me.

Ringrazio tutti i miei amici per avermi sostenuto, riso con me e alimentato le mie ambizioni.

Infine, ringrazio me stesso che, nonostante le cadute, non ho mollato mai.

Sommario

La nuova società dell'informazione offre servizi sempre più utili quanto innovativi dei quali però è importante non ignorare i difetti come la mancanza di regolamenti efficaci per la tutela dei diritti dell'utente. Attualmente, ogni azione sul web lascia una 'traccia' e spesso questa contiene dati personali. Alternativa a questo problema è l'uso di un'infrastruttura informatica per la stipula regolarizzata di *smart contract* che permette di verificare e autenticare *transazioni* senza memorizzare i dati personali delle entità coinvolte. Su questo concetto si basa l'idea di questa tesi: si immagina uno scenario in cui parte o la totalità del patrimonio dei cittadini è valutato in *criptovalute*, offrendo loro un servizio basato sulla tecnologia *blockchain* che permette di effettuare *transazioni* per regolare la divisione di un'eredità fornendo solo DID e indirizzi *wallet* degli eredi, adeguandosi ai meccanismi di *Self Sovereign Identity* e *Zero Knowledge Proof*.

Indice

1 Introduzione **1**

1.1 L'azienda 1

1.2 Obiettivi di tirocinio 2

2 Tecnologie utilizzate **3**

2.1 Blockchain 3

2.1.1 Tipologie 3

2.1.2 Blocco 4

2.1.3 Smart Contract 5

2.1.4 Transazione 6

2.1.5 Firma Asimmetrica 7

2.1.6 Token 7

2.1.7 Criptovalute 7

2.1.8 Mining 7

2.1.9 Vulnerabilità 9

2.2 Self Sovereign Identity 10

2.3 Zero Knowledge Proof 11

3 Progetto di tirocinio **13**

3.1 Analisi dei requisiti 13

3.2 Ambiente di sviluppo 14

3.2.1 Gestione delle librerie 14

3.2.2 Librerie utilizzate 14

3.2.3 Condizioni iniziali 15

3.3 Casi d'uso 15

3.4 Architettura del software 18

3.4.1 Frontend 18

3.4.2 Backend 23

4 Conclusioni **27**
4.1 Spunti per implementazioni future 27
4.2 Possibili applicazioni 27

Elenco delle figure

1.1 Logo Sync Lab 1

2.1 Struttura di un *blocco* 5
2.2 Procedura di *firma* di una *transazione* 6
2.3 Schema di funzionamento dell'*algoritmo di consenso* PoW 8
2.4 Schema di funzionamento della *Self Sovereign Identity* 10
2.5 Schema di funzionamento della *Zero Knowledge Proof* 11

3.1 Schema del *caso d'uso* 1 16
3.2 Schema del *caso d'uso* 2 16
3.3 Schema del *caso d'uso* 3 17
3.4 Schema del *caso d'uso* 4 17
3.5 View della pagina di login al sito 18
3.6 View della pagina di autenticazione al login del sito 19
3.7 View della homepage del sito 20
3.8 View della pagina della lista di polizze di eredità 21
3.9 View della pagina per l'inserimento dei dati di testamento 22
3.10 View della pagina di consegna della Verifiable Presentation 23
3.11 Procedimento di avvio di una polizza 24
3.12 Procedimento di verifica di una Verifiable Presentation 24

Capitolo 1

Introduzione

In questo capitolo viene descritta l'azienda presso cui si tenuto il tirocinio e gli obiettivi dell'esperienza.

1.1 L'azienda

Nata a Napoli nel 2002, Sync Lab (logo in figura 1.1) è una *Innovative Company* che realizza prodotti e soluzioni originali per diversi mercati come Sanità, Industria, Energia, Telecomunicazioni, Finanza, Trasporti e Logistica. Con ormai oltre 20 anni di esperienza nel settore IT, Sync Lab spicca per l'ottima competenza nei campi di GDPR, Big Data, Cloud Computing, IoT, Mobile e Cyber Security affermandosi come uno dei più importanti *System Integrator* italiani. Inoltre, grazie al laboratorio di *Ricerca e Sviluppo*, Sync Lab collabora con le più importanti università italiane con lo scopo di formare un nucleo solido di professionisti che si occupino di ricerca avanzata riguardo tematiche di tipo industriale e tecnologico.

Figura 1.1: Logo Sync Lab

1.2 Obiettivi di tirocinio

Con questo esposto di tesi si descrivono le attività e i concetti appresi durante il periodo di tirocinio che ho intrapreso tra i mesi di Marzo e Giugno 2023 presso la sede di Padova di Sync Lab. Il mio percorso in azienda è stato supervisionato dal tutor aziendale Ing. Fabio Pallaro e prevedeva in particolare:

- lo studio delle *tecnologie blockchain* e i loro concetti di base;
- lo studio degli *smart contract* e del linguaggio Solidity;
- lo studio di *Zero Knowledge Proof* e *Self Sovereign Identity* e le loro applicazioni;
- lo sviluppo di un prototipo di *smart contract* in linguaggio Solidity;
- l'implementazione del *contratto intelligente* tramite un'interfaccia web sviluppata in linguaggio JavaScript.

Durante tutto il periodo di tirocinio ho potuto collaborare con Gabriel Rovesti, laureando triennale in Informatica presso l'Università degli Studi di Padova, e Alessio De Biasi, laureando magistrale in Computer Science presso l'Università Ca' Foscari di Venezia.

Capitolo 2

Tecnologie utilizzate

In questo capitolo vengono descritte le *blockchain*, i loro meccanismi di base e alcune filosofie di applicazione.

2.1 Blockchain

L'idea di *blockchain* fu creata nel 2008 da Satoshi Nakamoto e si basa sulla creazione di una struttura dati che funge da registro immutabile per la compravendita di asset, garantendo così la loro tracciabilità e autenticità. Questa tecnologia è basata sul concetto della decentralizzazione: le catene non sono salvate su un unico server centralizzato (dove si rischia che i dati possano essere manipolati) ma sono peer-to-peer ovvero tutti gli utenti autorizzati ad accedere ad una data rete possiedono una copia della catena corrispondente. In questo modo, per verificare l'autenticità di una catena basterà confrontarla con le sue copie presenti su tutti gli altri nodi della rete.

2.1.1 Tipologie

Esistono diversi tipi di reti *blockchain* ma tra le più diffuse troviamo:

- reti *blockchain* **pubbliche**: sono reti a cui chiunque può accedere. Nonostante i dati registrati su queste reti siano pubblici, questi vengono crittografati limitando il più possibile problematiche di scarsa privacy e sicurezza. Alcuni esempi di reti *blockchain* pubbliche sono Bitcoin ed Etherium;

- reti *blockchain* **private**: sono reti gestite da una singola organizzazione che decide e regola i permessi di accesso tramite protocolli di consenso;

- reti *blockchain* **con autorizzazioni**: queste reti possono essere pubbliche o private e pongono dei limiti in fase di login con lo scopo di filtrare l'accesso ai dati da parte degli utenti e monitorare la loro partecipazione alle *transazioni*. Questo contribuisce ad incrementare la sicurezza della rete diminuendo così il rischio di hacking;

- reti *blockchain* **di consorzio**: sono reti la cui gestione è affidata a più organizzazioni che stabiliscono chi è autorizzato ad effettuare *transazioni* o accedere ai dati. Questo tipo di rete è più diffuso nei contesti aziendali in cui è necessario diffondere report lavorativi (e.g. fatture o stipendi) in maniera rapida, precisa ed efficiente.

2.1.2 Blocco

I dati delle *blockchain* sono raggruppati in *blocchi* (struttura in figura 2.1), dei pacchetti di codice che contengono specifiche informazioni per identificare una o più *transazioni* riguardo un determinato asset. Tra questi dati troviamo:

- l'hash del *blocco*;

- il numero del *blocco*;

- un puntatore hash al *blocco* precedente;

- una o più *transazioni*;

- un timestamp ovvero la data e ora di generazione del *blocco* (che corrisponde al timestamp dell'ultima *transazione*);

- il corrente valore di target;

- il nonce del *blocco*.

Le *transazioni*, oltre a *wallet* e *firme*, possono contenere informazioni aggiuntive che servono come granzia per accertarsi che l'asset scambiato rimanga quanto più immutato possibile nel susseguirsi dei vari passaggi di proprietà. Poiché l'hash di un *blocco* è generato dalla combinazione di tutte le informazioni che contiene, una volta scritte non possono essere modificate senza alterare anche tutti gli hash di tutti i suoi *blocchi* figli nella catena.

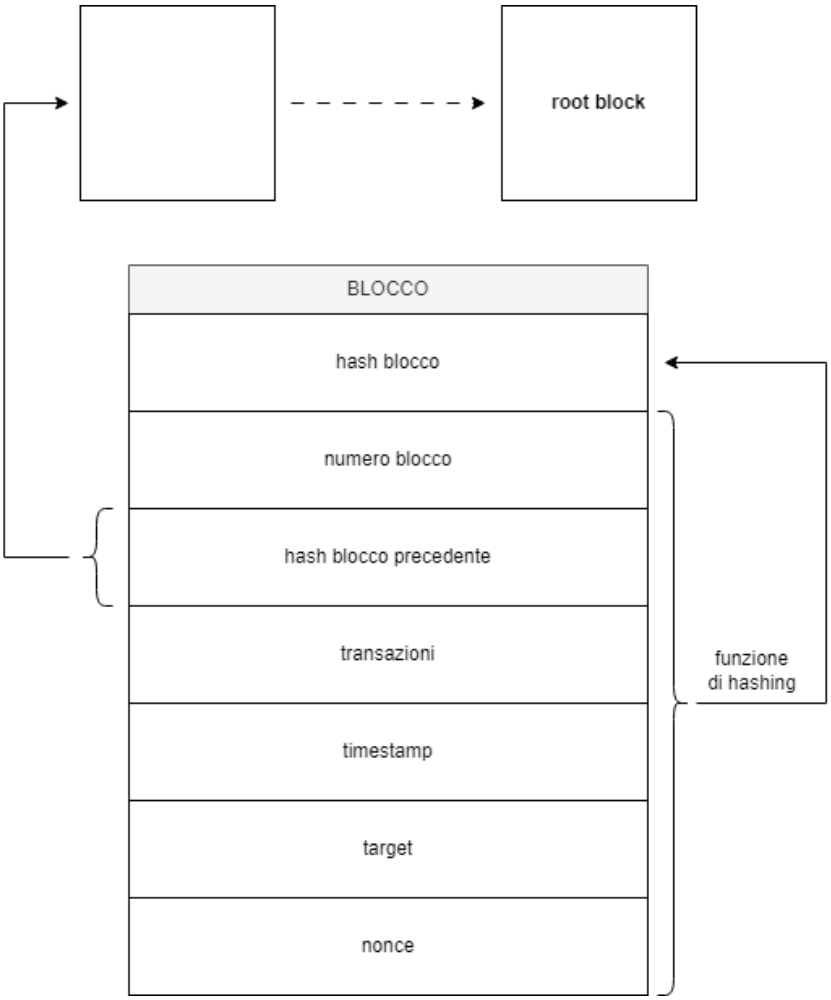

Figura 2.1: Struttura di un *blocco*

2.1.3 Smart Contract

Uno *smart contract* (o *contratto intelligente*) è un programma scritto nel codice della *blockchain* che viene eseguito per automatizzare l'esecuzione di un accordo (e.g. la compravendita di un asset) in modo che tutti i partecipanti possano essere immediatamente certi del risultato senza il coinvolgimento di intermediari come notai o enti terzi. La stesura di uno *smart contract* è a carico di un programmatore e può contenere tutte le clausole necessarie per assicurare ai partecipanti che la *transazione* verrà completata in modo soddisfacente, esplorando tutte le possibili eccezioni con lo scopo di definire anche una struttura per la risoluzione delle controversie. Questa trasparenza è uno dei vantaggi fondamentali degli *smart contract* in quanto, una volta siglato il *blocco*, non sarà necessario chiedersi se le informazioni sono state modificate per vantaggio personale.

2.1.4 Transazione

Come descritto nella sezione precedente, una *transazione* è la parte di un *blocco* che contiene i dati costitutivi dell'effettivo registro condiviso di una *blockchain*. Quando un utente si unisce ad una *blockchain* viene creato il suo *wallet* e le corrispondenti *chiavi crittografiche* così che possa effettuare e validare *transazioni*.

2.1.4.1 Wallet e Chiavi Crittografiche

Un *wallet* è un portafoglio digitale che permette di comunicare con la *blockchain* dove risiedono gli asset (e.g. *criptovalute*) di cui si è proprietari. Per accedere e usufruire del *wallet*, al momento della creazione l'utente è dotato di due chiavi alfanumeriche distinte:

- *chiave* **pubblica**: è un codice identificativo e univoco dell'utente, usato per avviare o ricevere *transazioni*. A ogni *wallet* è associato un indirizzo pubblico che viene calcolato tramite l'hashing della *chiave pubblica*;

- *chiave* **privata**: è un codice segreto necessario per accedere al *wallet*, usato per firmare *transazioni* e usufruire degli asset. Chiunque conosce la *chiave privata* di un *wallet* è come se lo possedesse in quanto potrebbe scambiare tutti gli assets con qualsiasi altro indirizzo; è quindi importante custodire la *chiave privata* con molta cautela e segretezza.

Figura 2.2: Procedura di *firma* di una *transazione*

2.1.5 Firma Asimmetrica

La *firma asimmetrica* (procedura schematizzata in figura 2.2) si basa su una funzione matematica che lega *chiave pubblica e privata*. In questo modo, il mittente può inviare un messaggio criptandolo con la chiave pubblica del destinatario; una volta arrivato a destinazione, il messaggio potrà essere decifrato grazie la *chiave privata* del destinatario. Questo è il ragionamento di scambio che funziona alla base della *blockchain*.

2.1.6 Token

Con *token* si intende un asset digitale registrato su *blockchain* e associato ad un utente che ne detiene il diritto di proprietà, sia esso un bene o un servizio. I *token* si dividono in due categorie:

- *token* **fungibile**: è un *token* il cui valore è equivalente a quello di qualsiasi altra sua copia. Un esempio sono le *criptovalute*: il valore di un Ether è uguale a quello di qualsiasi altro Ether;
- *token* **non fungibile**: più conosciuti con il loro acronimo, NFT, sono *token* unici e indivisibili (e.g. lotti di terreno, arte reale o digitale). La loro non intercambiabilità permette ai dati registrati in *blockchain* di fornire una rappresentazione univoca dell'asset.

2.1.7 Criptovalute

Le *criptovalute* sono monete virtuali (e.g. Ether e Bitcoin) che non dipendono da nessun organo bancario o governativo. Tutte le informazioni sulle transazioni di una data *criptovaluta* sono registrate in maniera decentalizzata sulla *blockchain* di appartenenza (e.g. gli Ether registrati sulla *blockchain* Ethereum).

2.1.8 Mining

Con *mining* (dall'inglese 'to mine' che significa estrarre) si intende il processo per "generare" nuove unità di *criptovalute* elaborando e verificando le *transazioni* registrate sulla *blockchain* della valuta da estrarre. In sé, la validazione di questi *blocchi* avviene attraverso *algoritmi di consenso* risolti dai minatori grazie a reti di elaboratori con elevata potenza di calcolo sfociando così in un inevitabile alto consumo di energia.

2.1.8.1 Algoritmi di Consenso

Con *algoritmi di consenso* si intendono i meccanismi di validazione dei *blocchi* in fase di *mining*. Tra i principali *algoritmi di consenso* troviamo:

- **Proof of Work**: o PoW (procedimento in figura 2.3), consiste nella proposta di un nuovo *blocco* con la struttura descritta della sezione 2.1.2. Tramite l'algoritmo di hashing SHA-256 viene poi generato l'hash del *blocco* che è successivamente confrontato con il corrente valore di target della *blockchain*: se l'hash è maggiore o uguale al target allora si cerca un nuovo nonce (per il calcolo di un nuovo hash che rientri nel valore di target), altrimenti il *blocco* è risolto e aggiunto alla catena. È prevista l'emissione di una quota fissa di *criptovaluta* come reward per l'utente che riesce a validare e aggiungere alla *blockchain* un nuovo *blocco*. Prima di completare una PoW valida sono necessari una grande quantità di tentativi ed errori perciò si tratta di un algoritmo ad alto consumo energetico e bassa probabilità di successo;

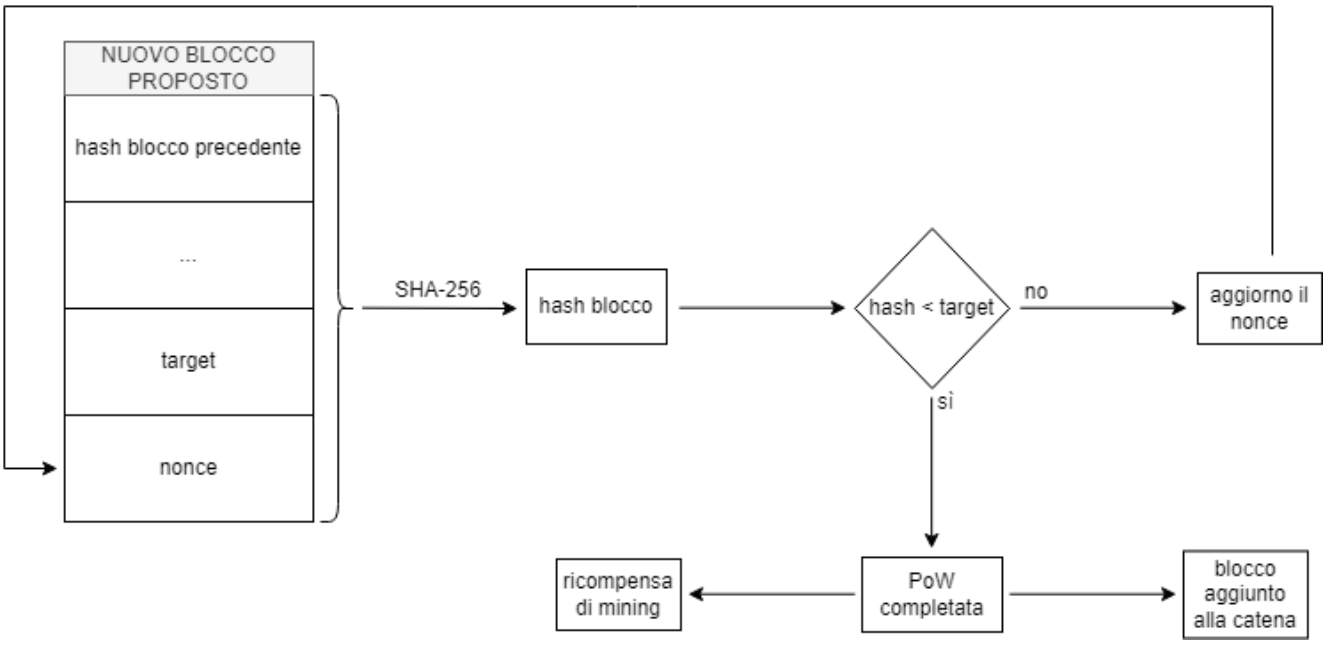

Figura 2.3: Schema di funzionamento dell'*algoritmo di consenso* PoW

- **Proof of Stake**: nella PoS, i validatori devono già possedere della *criptovaluta* e immobilizzarla ('stake') come garanzia di partecipazione al processo di consenso. A differenza del PoW, nel PoS è l'algoritmo stesso a decidere: più alto è lo stake e più alta è la probabilità di venire scelti come validatore del prossimo *blocco*. I validatori ricevono ricompense sottoforma di *criptovaluta* come commissioni di *transazione* ma in caso di intenti malevoli tutto il loro capitale immobilizzato può essere distrutto. In questa maniera, più un utente investirà valuta, più sarà elegibile e sicuro che i *blocchi* saranno validati senza

frodi. Questo *algoritmo di consenso* garantisce quindi una migliore sicurezza ed è più energeticamente efficiente del PoW.

2.1.9 Vulnerabilità

Nella csotruzione di *blockchain* è importante valutare le possibili falle di sicurezza che potrebbero compromettere la privacy dell'utente. Queste *vulnerabilità* sono spesso dovute a insufficiente controllo dei dati (e.g. password o indirizzi) all'interno dei *contratti intelligenti* che gestiscono le *transazioni* di una catena. Tra le possibili diverse tipologie di attacco hacker a una *blockchain* troviamo:

- *attacco* di **re-entrancy**: questo attacco è causato dallo *smart contract* della *blockchain* che effettua una chiamata ad una funzione di un *contratto* esterno alla catena. Così facendo, se lo *smart contract* esterno è malevolo allora la funzione potrà essere chiamata ricorsivamente estraendo a ogni iterazione una data quantità di valuta;

- *attacco* con **poison contract**: un *contratto* "avvelenato" agisce in maniera analoga a quella della re-entrancy ma invece di rubare fondi si occupa di bloccare il servizio offerto dalla *blockchain* causando cicli di istruzioni infiniti (e.g. while True). L'unico scopo di questo tipo di attacco è quindi procurare un disagio all'utenza della *blockchain*;

- *attacco* tramite **integer underflow/overflow**: questo attacco lavora sulle variabili numeriche di un *contratto* che, incautamente, non sono state gestite correttamente (e.g. non esistono controlli in caso di valori anomali). L'incremento (o decremento) smisurato di queste variabili e la memoria limitata degli elaboratori generano situazioni di underflow o overflow falsando completamente il dato autentico. Uno tra i casi più celebri di questo tipo di attacco è avvenuto nel 2018 con lo *smart contract* di Beauty Chain Coin: gli hacker attaccanti hanno ottenuto una crescita non autorizzata della *criptovaluta* portando la quantità di moneta in circolazione al suo limite massimo e facendo così crollare il suo valore da 900 milioni di dollari a quasi zero;

- *attacco* **timestamp**: come descritto nella sezione relativa al *mining*, ogni *blocco* di una *blockchain* di *criptovaluta* è estratto con cadenza regolare. Questo attacco mira ad usare una *vulnerabilità* logica dello *smart contract* per falsificare il timestamp di un *blocco* e generare così valori nonce fasulli che farebbero però guadagnare la quantità di valuta associata al *blocco* minato. La possibilità di contraffarre il timestamp permetterebbe così ad un utente malevolo di indovinare il nonce in maniera ricorsiva.

2.2 Self Sovereign Identity

La *Self Sovereign Identity* è uno standard di identità digitale che lavora sulla tecnologia decentralizzata della *blockchain*. Questa politica di tutela dei dati personali si basa sulla restituzione all'utente delle proprie informazioni abolendo la necessità di appoggiarsi ad enti terzi come Meta o Google. Il funzionamento della *SSI* (schematizzato in figura 2.4) si basa sul meccanismo di *firma asimmetrica* e nel suo processo coinvolge tre entità principali:

- **holder**: è la figura che emette una credenziale relativa al servizio che offre (e.g. DID) che può contenere uno o più 'claim', degli attributi collegati all'identità digitale (e.g. data di nascita). Inoltre, l'holder si occupa di rilasciare delle Verifiable Credential che certificano in modo crittografato la valenza della propria autorità;
- **issuer**: è la figura proprietaria della credenziale generata dall'holder. È compito dell'issuer certificare un dato firmandolo con la propria coppia di chiavi;
- **verifier**: è l'entità che si occupa di controllare e verificare l'autenticità della credenziale fornita dall'issuer risalendo la Chain of Trust associata alla VC generata dall'holder.

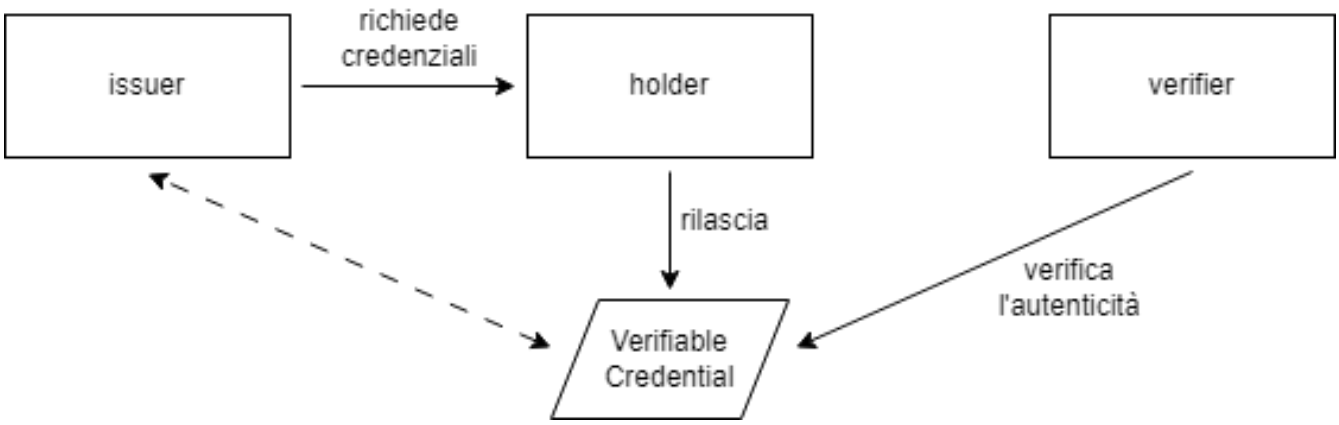

Figura 2.4: Schema di funzionamento della *Self Sovereign Identity*

2.3 Zero Knowledge Proof

Oltre a *SSI*, esistono altri protocolli di standardizzazione che consentono di scambiare dati crittografati garantendo alle entità il controllo indipendente della propria identità. Uno di questi è senz'altro *Zero Knowledge Proof*, un meccanismo di verifica delle informazioni che consente agli utenti di fornire una prova d'accesso senza però rivelare dati sensibili (e.g. verificare la maggiore età senza fornire la data di nascita). Il funzionamento di *ZKP* (schematizzato in figura 2.5) si basa su due entità principali:

- **prover**: è l'entità che conosce l'informazione segreta e che, rispondendo a delle domande specifiche, dovrà dimostrare al verifier di possedere il dato ('proof') oltre la sua correttezza;
- **verifier**: è l'entità che sfida il prover con dei quesiti specifici sull'informazione e che una volta ottenuta una 'proof' ne verifica la veridicità. Il verifier può decidere di porre più domande sullo stesso argomento per assicurarsi che il prover non stia rispondendo correttamente in modo fortuito.

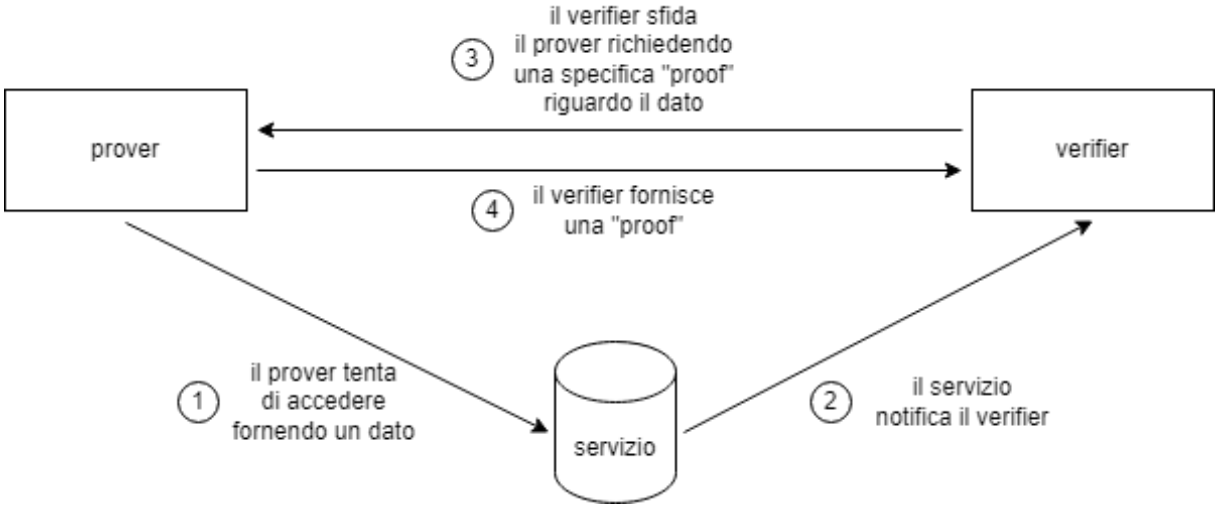

Figura 2.5: Schema di funzionamento della *Zero Knowledge Proof*

Capitolo 3

Progetto di tirocinio

Durante il mio periodo di tirocinio presso Sync Lab ho lavorato ad ”Armtance”, un progetto che prevede un servizio online di polizze atte alla spartizione di eredità quotate in *criptovalute* e la gestione delle varie *transazioni* tramite l'utilizzo di *contratti intelligenti*. ”Armtance” è sviluppato principalmente in linguaggi JavaScript e Solidity e La *blockchain* su cui funziona, lavora affidandosi a politiche di *Self Sovereign Identity* anche grazie ad uno *smart contract* sviluppato interamente da Alessio De Biasi.

3.1 Analisi dei requisiti

Di seguito sono elencati i principali obiettivi di tirocinio nonché i requisiti fondamentali del progetto:

- sviluppo di un'interfaccia utente per l'inserimento dei dati degli eredi;
- sviluppo di un'interfaccia utente per l'inserimento di una Verifiable Presentation (contenente una o più Verifiable Credential) che certifichi la dipartita del testatore;
- creazione di uno *smart contract* in linguaggio Solidity per la gestione delle *transazioni*;
- introduzione di meccanismi di *Self Sovereign Identity*;
- valutazione e studio delle possibili ulteriori *vulnerabilità* legate al progetto.

Il progetto prevedeva inizialmente solo lo sviluppo di un'interfaccia web per l'inserimento di dati di smistamento e la loro verifica perciò, solo dopo la fine del tirocinio, come complemento alla mia interfaccia ho deciso di implementare anche una schermata di login che opera con politiche di *Zero Knowledge Proof* e con meccanismi simili a quelli utilizzati da Gabriel Rovesti nel suo elaborato. Come requisiti non obbligatori ma successivamente aggiunti al progetto troviamo quindi:

- sviluppo di un'interfaccia utente per il login di testatori ed eredi;
- introduzione di meccanismi di *Zero Knowledge Proof*.

3.2 Ambiente di sviluppo

In questa sezione vengono spiegate gli strumenti e le librerie utilizzate nello sviluppo (frontend e backend) del progetto.

3.2.1 Gestione delle librerie

3.2.1.1 Node.js

Node.js è un ambiente di runtime open source basato su JavaScript che consente agli sviluppatori di creare applicazioni web e servizi di rete altamente scalabili e efficienti. Inoltre, *Node.js* offre un servizio di packet management ('npm') che permette di gestire le dipendenze di progetto, installando, aggiornando e tenendo traccia del versionamento delle librerie in modo rapido e controllato.

3.2.2 Librerie utilizzate

3.2.2.1 React

React è una libreria JavaScript open source per la creazione di interfacce utente sviluppata da Facebook. L'obiettivo principale di *React* è semplificare la creazione di interfacce utente complesse, consentendo agli sviluppatori di costruire componenti riutilizzabili che gestiscono il proprio stato e si integrano facilmente all'interno di un'applicazione web o mobile.

3.2.2.2 Web3.js

Web3.js è una libreria JavaScript che consente lo sviluppo di DApps (applicazioni decentralizzate) capaci di integrarsi e interagire con la *blockhain* Ethereum. Questa libreria permette una ottima gestione delle *chiavi crittografiche* e funge da supporto per i metodi chiamati o eventi generati dai *contratti intelligenti*.

3.2.2.3 adb-library

adb-library è una libreria di supporto sviluppata interamente da Alessio De Biasi in linguaggio Typescript. Questa libreria permette, tramite un'istanza di *Web3.js*, di interagire con il *contratto* di *Text Sovereign Identity* in modo da:

- generare DID unici per i nuovi utenti della rete;
- ”risolvere” i DID ovvero, partendo da una Verifiable Presentation, risalire la catena di fiducia per verificare l’autorità degli utenti holder e issuer.

3.2.3 Condizioni iniziali

Il progetto prevede l’esistenza di una rete in cui:

- ogni utente cliente del servizio (testatore, ereditario o utente generico) è provvisto di un DID;
- esiste una *blockchain* (indipendente da quella che gestisce le *transazioni*) che contiene i dati di tutti gli holder autorizzati a rilasciare Verifiable Credential, nel nostro caso riguardo certificati di morte. Tutti questi holder costituiscono la Chain of Trust che il verifier dovrà risalire per verificare l’autenticità delle VC;
- sono attivi tutti gli *smart contract* necessari per gestire *transazioni* e *Self Sovereign Identity*.

Per simulare queste situazioni, eseguo in locale un’istanza di Ganache che genera un network di *wallet* che uso per:

- generare i DID di alcuni utenti di test;
- creare una catena di fiducia di test valida dedicando agli holder alcuni dei DID generati;
- depositare sulla rete gli *smart contract* dedicando loro due indirizzi del network Ganache.

3.3 Casi d’uso

Come descritto nella sezione 3.2.3, lo scenario del progetto prevede che gli utilizzatori del servizio siano già in possesso di un DID valido (lascio la fase di registrazione di un nuovo utente come spunto per un’implementazione futura). Ogni utente registrato può essere sia testatore (colui che stipula una polizza per la spartizione dell’eredità) che erede e allo stesso modo un qualsiasi utente registrato sulla rete (non necessariamente un’ereditiero) può fornire una Verifiable Credential. Dunque, considerando le queste ultime considerazioni e le condizioni iniziali precedentemente elencate identifico i seguenti *casi d’uso*:

1. utente autorizzato testatore accede al servizio per stipulare una polizza di spartizione eredità, inserendo i dati relativi ai singoli eredi. Un utente testatore può avere una sola polizza valida attiva (schema in figura 3.1);

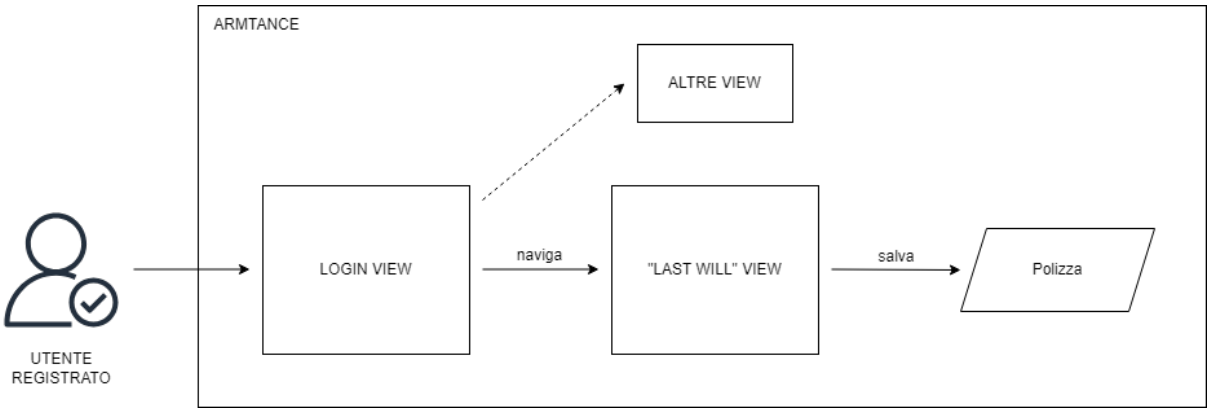

Figura 3.1: Schema del *caso d'uso* 1

2. utente autorizzato testatore accede al servizio per modificare i dati dei singoli eredi relativi ad una sua polizza già stipulata e attiva (schema in figura 3.2);

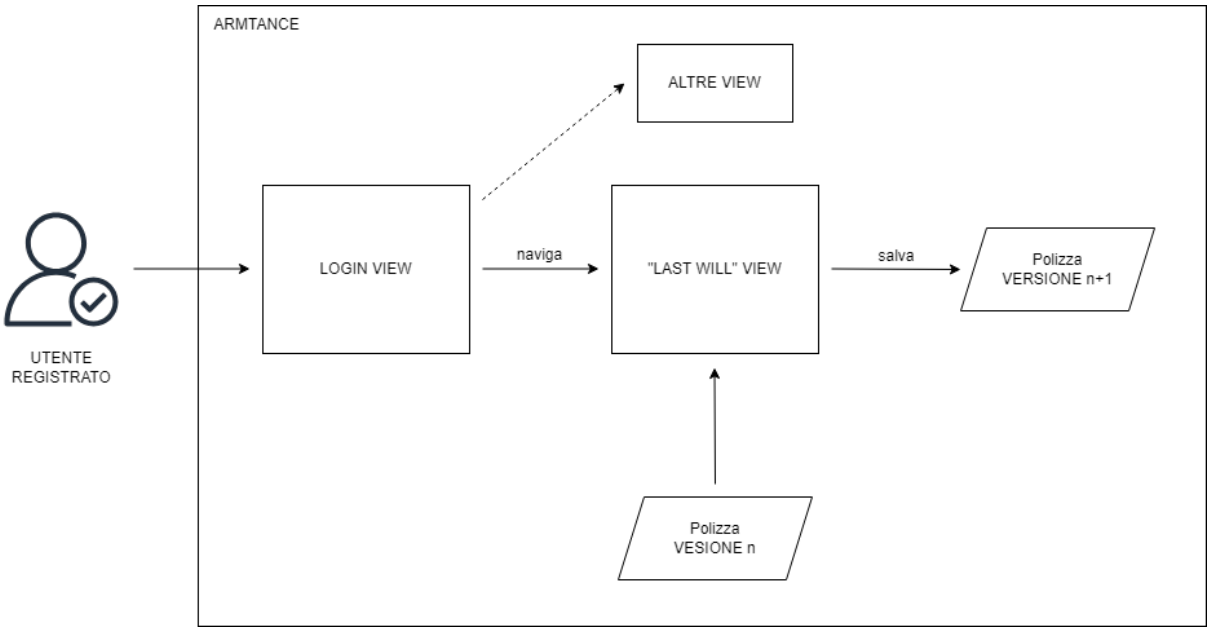

Figura 3.2: Schema del *caso d'uso* 2

3. utente autorizzato ereditiere accede al servizio per visualizzare a quali polizze il suo DID è stato associato. Un utente ereditiere può essere inserito come beneficiario di una o più polizze da parte di utenti testatori differenti (schema in figura 3.3);

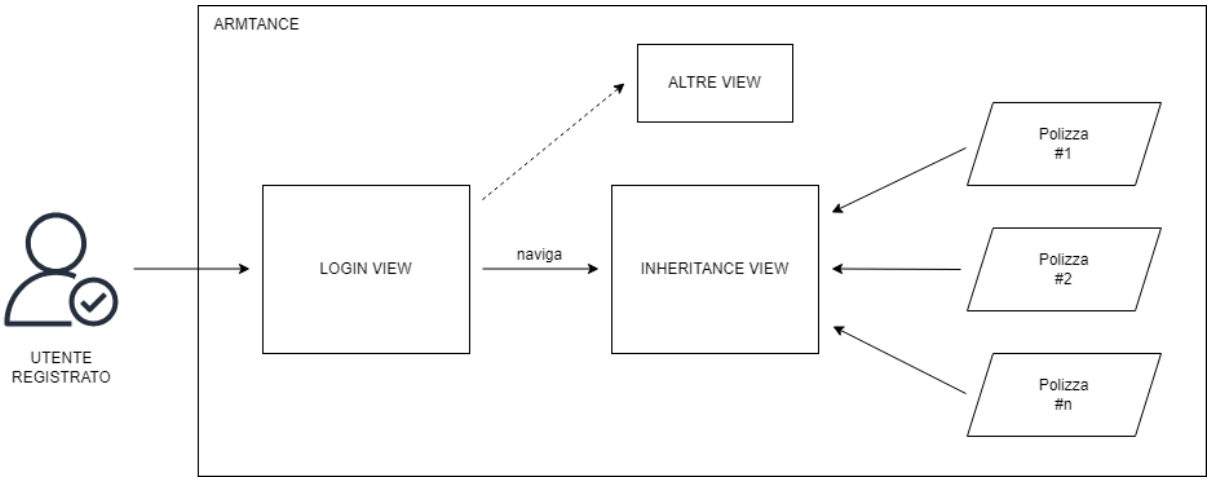

Figura 3.3: Schema del *caso d'uso* 3

4. utente autorizzato generico accede al servizio per fornire una Verifiable Presentation contenente una Verifiable Credential riguardante il certificato di morte di un defunto testatore (schema in figura 3.4).

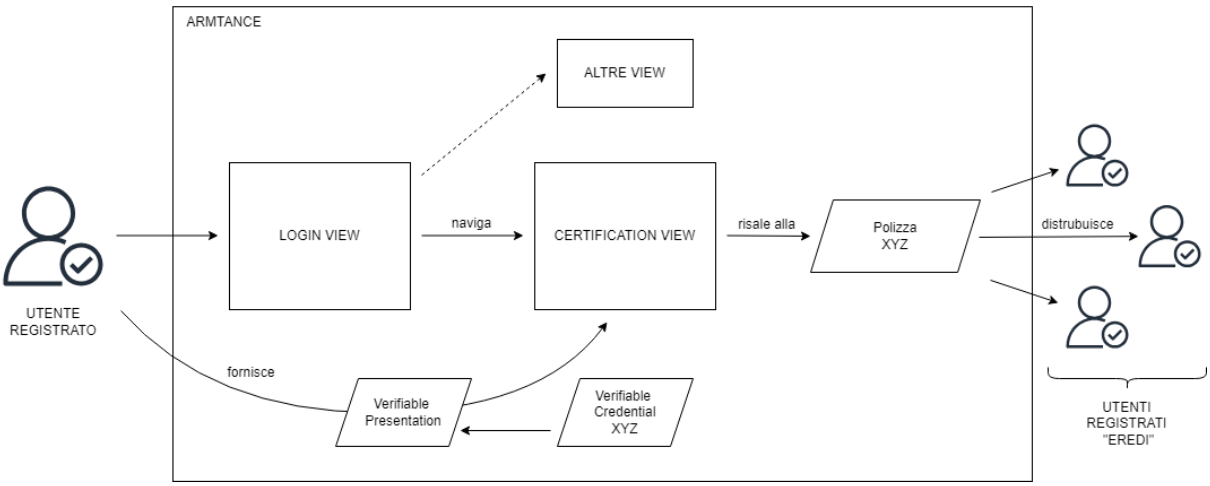

Figura 3.4: Schema del *caso d'uso* 4

3.4 Architettura del software

In questa sezione descrivo come le diverse view interagiscono tra di loro e come il backend comunica con gli *smart contract*.

3.4.1 Frontend

3.4.1.1 Login View

Figura 3.5: View della pagina di login al sito

”*Login View*” (in figura 3.5) è la pagina di Armtance che consente di effettuare il login al proprio account. L’unico dato richiesto per l’accesso è il DID dell’utente; per assicurarsi che non si tratta di un tentativo di accesso malevolo, successivamente compare la ”Authenication View” (in figura 3.6), una schermata che richiede di crittografare con la *chiave privata* associata al DID inserito una serie di parole scelte randomicamente dal sistema. Se la sequenza firmata sarà verificata correttamente allora l’utente potrà accedere al servizio con il DID fornito. Questo

procedimento di verifica (simile a quello utilizzato dal servizio Metamask) consente un login che si omolga ai meccanismi di *Zero Knowledge Proof*.

Figura 3.6: View della pagina di autenticazione al login del sito

3.4.1.2 Homepage View

”*Homepage View*” (in figura 3.6) si riferisce alla prima schermata che si incontra dopo l’accesso al portale. La pagina contiene principalmente tre pulsanti:

1. CONTROLLA LE POLIZZE DI CUI SEI BENEFICIARIO: questo pulsante conduce alla ”*Inheritance View*”, pagina che si occupa di mostrare le diverse polizze di cui l’utente possessore del DID risulta essere beneficiario;

2. CREA UNA POLIZZA: questo pulsante conduce alla ”*Last Will View*”, pagina che si occupa di stipulare il testamento;

3. INSERISCI UNA VERIFIABLE PRESENTATION: questo pulsante conduce alla ”*Certification View*”, pagina che si occupa di raccogliere dall’utenza una Verifiable Presentation con lo scopo di avviare un evento di ”spartizione” di un’eredità.

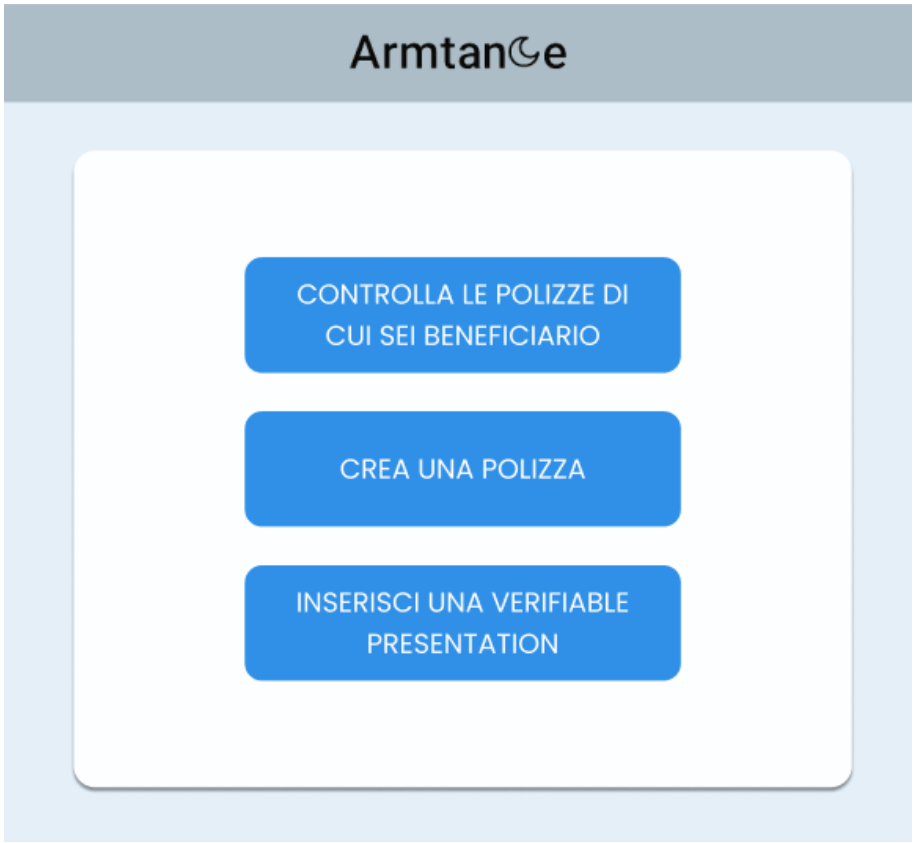

Figura 3.7: View della homepage del sito

3.4.1.3 Inheritance View

Cliccando sul primo pulsante della ”*Homepage View*” si accede alla ”*Inheritance View*” (in figura 3.7). Questa schermata deve il suo nome al fatto che, per scelta progettuale, un DID può essere inserito in una o più polizze. In questo modo, ogni utente ha la possibilità di

- controllare a quali polizze il suo DID è stato associato;
- controllare il DID di ogni testatore;

ma non può vedere l’ammontare dell’eredità finale di una polizza. Quest’ultima scelta è di carattere progettuale e serve a tutelare l’informazione riguardante il patrimonio del testatore, adeguando così il progetto ad alcuni meccanismi della *Self Sovereign Identity*.

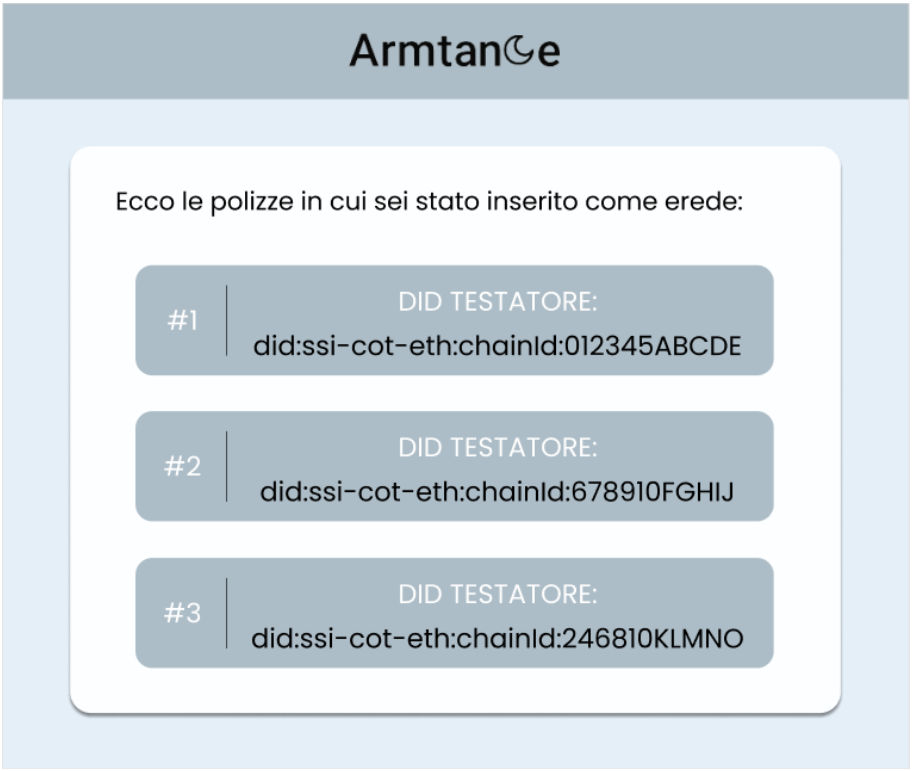

Figura 3.8: View della pagina della lista di polizze di eredità

3.4.1.4 Last Will View

"*Last Will View*" (in figura 3.9) è la schermata in cui un utente può "stipulare" una polizza riguardante il proprio testamento. In questa pagina il testatore ha la possibilità di:

- inserire un numero di eredi a piacimento cliccando sul pulsante "+";
- inserire per ogni 'entry' un DID associato all'account di un erede;
- inserire per ogni 'entry' una percentuale di patrimonio da trasferire al *wallet* del DID associato.

Il valore percentuale finale è calcolato sull'ammontare del patrimonio presente sul *wallet* del defunto al momento della sua dipartita. La scelta di utilizzare un campo "percentuale" è di natura progettuale in quanto se fosse di tipo numerico naturale si potrebbero creare delle ingenti depositi di capitale residuo su account associati a persone decedute.

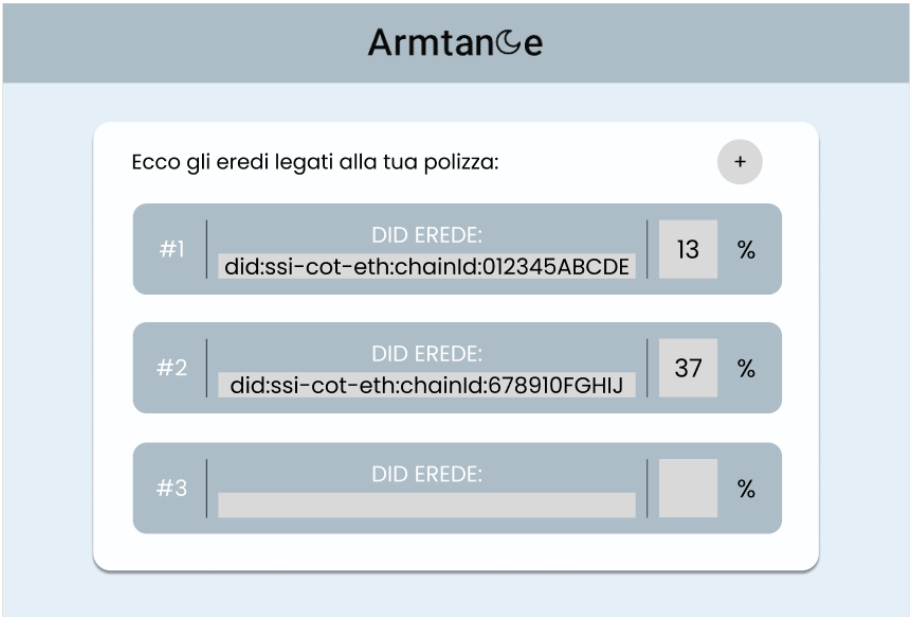

Figura 3.9: View della pagina per l'inserimento dei dati di testamento

3.4.1.5 Certification View

"*Certification View*" (in figura 3.10), è la pagina in un utente può inserire una Verifiable Presentation in formato JSON con lo scopo di 'triggerare' un'evento di spartizione di un'eredità. Anche questa situazione ha dei vincoli progettuali: una Verifiable Presentation valida può essere presentata da qualsiasi utente Armtance correttamente registrato sulla rete (sia esso compreso o non

nella polizza del testatore) tranne che dall'account associato al DID del defunto dal momento che, in caso di decesso del proprietario, un profilo dovrebbe essere considerato inaccessibile.

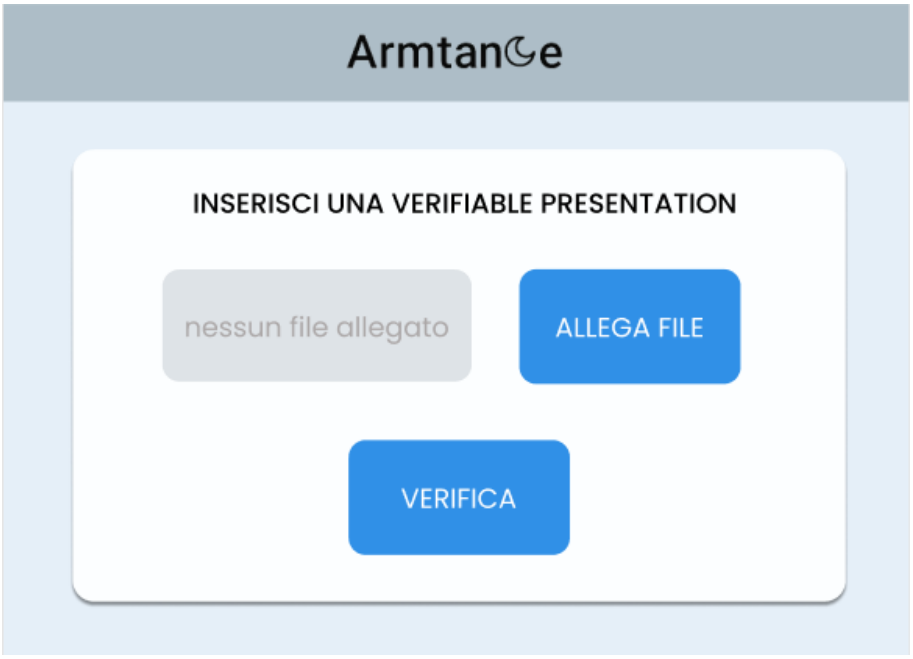

Figura 3.10: View della pagina di consegna della Verifiable Presentation

3.4.2 Backend

3.4.2.1 Interazioni con 'Inheritance.sol'

'Inheritance.sol', lo *smart contract* ideato da me, si occupa di gestire (permettendo di creare e modificare) le polizze d'eredità. Come raffigurato in figura 3.11, la prima volta che un utente decide di stipulare il proprio testamento accedendo alla "Last Will View", il frontend inoltra una chiamata al *contratto intelligente* generando così un'istanza della polizza che terminerà solamente nel momento in cui la Verifiable Presentation presentata sarà ritenuta valida; solo allora i trasferimenti di denaro (*transazioni*) verranno autorizzati e il *blocco* verrà considerato chiuso. Il *contatto* si basa su due 'strutture' principali:

- 'struct' Issuer: utilizzata per conservare i dati del testatore come il suo DID e il suo indirizzo *wallet*;
- 'struct' Heir: utilizzata per conservare i dati di tutti gli eredi; poiché Solidity non permette la creazione di array di oggetti "custom", gli eredi sono mappati in 'heirMap'.

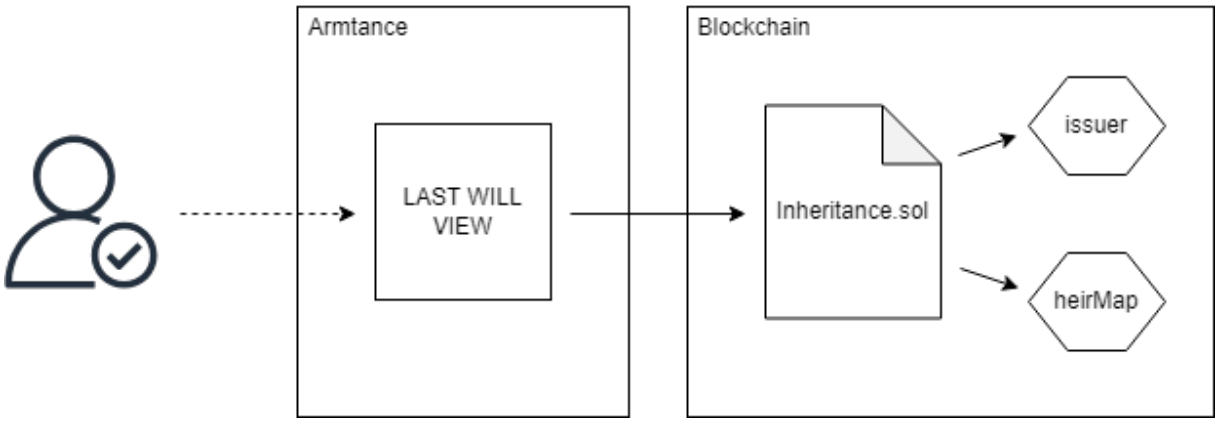

Figura 3.11: Procedimento di avvio di una polizza

La scelta di mantenere l'istanza del *contratto* "aperta" è di natura progettuale e, pur presentando delle vulnerabilità, consente di modificare in qualsiasi momento i dati di spartizione di un'eredità (DID e percentuali).

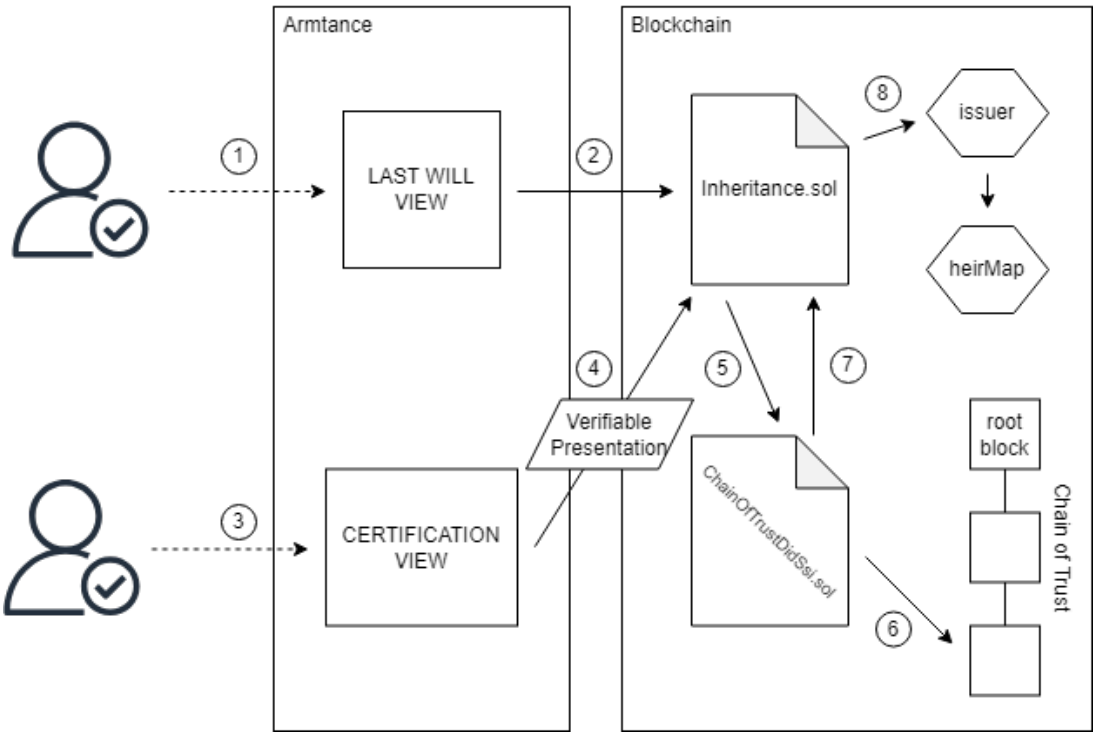

Figura 3.12: Procedimento di verifica di una Verifiable Presentation

3.4.2.2 Interazioni con 'ChainOfTrustDidSsi.sol'

'ChainOfTrustDidSsi.sol' è lo *smart contract* sviluppato da Alessio De Biasi e serve a garantire la *Self Sovereign Identity* del progetto. 'ChainOfTrustDidSsi.sol' è richiamato da 'Inheritan-

ce.sol‘ quando un utente presenta una Verifiable Presentation e serve a risalirne la Chain of Trust per verificarne la validità e procedere così allo smistamento dell’eredità (procedimento schematizzato in figura 3.12).

3.4.2.3 Vulnerabilità

Tra le vulnerabilità principali presenti al momento troviamo:

- non sono presenti meccanismi di verifica riguardo la presenza e veridicità dei wallet;
- non sono presenti controlli di sicurezza in fase di inserimento dei DID ovvero un utente potrebbe inserire un DID che non si riferisce all’account voluto;
- nel caso di perdita o furto della *chiave privata* di un account, non sono presenti meccanismi di recupero o modifica di emergenza delle polizze.

Capitolo 4

Conclusioni

4.1 Spunti per implementazioni future

Con lo scopo di migliorare il progetto risolvendo quante più vulnerabilità possibili, lascio i seguenti spunti per delle possibili implementazioni future:

- permettere a ogni DID di possedere uno o più *wallet* ognuno dei quali può essere riconosciuto tramite una 'key'. Ogni DID assumerebbe dunque la forma 'did:ssi-cot-eth:chainId:012345ABCDE#key'; infatti, in caso di impossessamento illecito di *chiave privata*, si ridurrebbe drasticamente il rischio di frode in quanto basterebbe che il testatore cambi nella sua polizza solo la 'key' corrispondente al *wallet* dell'erede;
- implementare la funzionalità di registrazione da parte degli utenti alla rete *blockchain*;
- migliorare l'interfaccia utente rendendola più intuibile per tutte le possibili fasce di clientela;
- studiare un modo più efficiente per gestire gli eredi e la spartizione dell'eredità tramite 'Inheritance.sol';
- migliorare al gestione delle chiamate tra i *contratti* 'Inheritance.sol' e 'ChainOfTrustDidSsi.sol'.

4.2 Possibili applicazioni

Questa tesi e questo progetto servono da spunto per incentivare la nascita di una nuova generazione di programmatori *blockchain* capaci di sfruttare la potenza di questa nuova tecnologia e avviare un nuovo processo di innovazione per la tutela dei diritti dei cittadini oltre che portare all'eliminazione della lentezza burocratica tipica del nostro Bel Paese. È importante far

notare che questa tecnologia è immutabile e non contraffabile perciò nessuna entità, sia esso un ente governativo o un'azienda privata, dovrebbe temere la sua potenza, anzi, questo dovrebbe essere motivo per incentivare il suo utilizzo. A tale scopo, di seguito elenco alcune possibili applicazioni della *tecnologia blockchain*:

- tracciabilità del cibo. In sé questa idea è applicabile a qualunque tipo di cibo (verdura, carne, pesce, etc.) e serve a tutelare la salute del cittadino mettendo a disposizione di chiunque grandi quantità di dati autenticati riguardo zone e metodologie di allevamento, modalità di trasporto e proprietà dei prodotti;

- rilascio documenti ufficiali come carte d'identità, patenti e passaporti, riducendo notevolmente i tempi burocratici che attualmente ne frenano la velocità. Il funzionamento di questa proposta è simile a quello utilizzato in Armtance per la verifica dei certificati di morte.

Glossario

asset termine che si riferisce a beni di scambio che possono essere tangibili (e.g. oro, case, auto o terreni) o intangibili (e.g. brevetti, copyright o branding). Poiché sono registrati in *blockchain*, una volta salvati sono immutabili e possono essere trasferiti in modo sicuro tra un utente e l'altro . 3–7

Chain of Trust è una *blockchain* i cui *blocchi* identificano gli holder di un servizio. Per verificare se la Verifiable Credential rilasciata da un holder è autentica e valida basterà risalire la catena di cui l'holder fa parte fino a constatare l'autorità del 'root block' . 10, 15, 25

DID o Decentralized Identifier, è un protocollo che permette di creare codici crittografati che consentono di identificare entità in modo univoco, siano essi persone, aziende o dispositivi. Questo schema è sviluppato secondo lo standard W3C Verifiable Credentials ed è composto da un prefisso che identifica la rete *blockchain* per la quale il DID è stato creato, seguito da un identificatore univoco generato dall'utente . i, 10, 15, 16, 18, 20–25, 27

Ganache è un componente del framework Truffle, ambiente di sviluppo basato sul EVM (Ethereum Virtual Machine). Ganache è uno strumento usato per eseguire reti *blockchain* locali e agevolare lo sviluppo di applicazioni decentralizzate sul newtwork Ethereum. Tra i più grandi vantaggi offerti da Ganache troviamo principalmente la semplicità con cui permette di depositare ('deploy') *smart contract* sulle reti locali garantendo così maggiore sicurezza nelal loro fase di sviluppo e test . 15

hash prodotto da una funzione di hashing, l'hash è una stringa strettamente correlata ai dati di ingresso. Infatti, per modificare l'hash di un *blocco* basterà alterare anche solo un carattere del suo contenuto . 4, 8

JavaScript è un linguaggio di programmazione orientato agli eventi che può essere utilizzato sia nella programazione frontend che backend. Javascript, quindi, garantisce la possibilità di progettare applicazioni web dinamiche e interattive integrate alla capacità di comunicazione tramite script lato client e/o API su lato server . 2, 13, 14

minatori i minatori sono un elemento fondamentale per la sicurezza delle *criptovalute* in quanto sono le entità che si occupano di validare i *blocchi* estratti. Il ”mestiere” del crypto-minatore è una vera e propria attività imprenditoriale che richiede risorse dispendiose come network di elaboratori ad alta potenza di calcolo e quantità di energia non indifferenti . 7

nonce il nonce è un valore numerico che viene cercato dai minatori e attribuito ad un blocco per modificarne l'hash affinché esso non superi il valore di target . 4, 8, 9

Solidity è un linguaggio di programmazione orientato agli oggetti basato su un progetto open source sponsorizzato da ”Ethereum Foundation”. Solidity è stato progettato specificamente per la scrittura e il 'deploy' di *smart contract* sulla *blockchain* Ethereum . 2, 13, 23

target il valore di target è il valore entro cui deve stare l'hash di un blocco per poter essere validato come ”estratto”. Questo valore è aggiornato dalla *blockchain* ogni due settimane circa ed è calcolato in base al numero di minatori attualmente attivi in una rete. In sé, quindi, serve a mantenere costante la quantità di *blocchi* minati: più il targe è basso e più la ricompensa per l'estrazione del *blocco* sarà alta . 4, 8

Verifiable Credential o VC, sono documenti digitali che seguono lo standard W3C e contengono informazioni riguardo un determinato ambito (e.g. DID) di una data entità (e.g. persone, aziende o dispositivi). Una VC è un documento crittografato, firmato digitalmente e rilasciato da un'autorità che ne certifica l'autenticità delle informazioni . 10, 13, 17

Verifiable Presentation o VP, sono documenti digitali che seguono lo standard W3C. Una VP è un documento crittografato e firmato digitalmente che contiene una collezione di metadati riguardanti la presentazione, una o più Verifiable Credential e relative 'proof' che certificano l'autenticità delle singole credenziali . 13, 15, 17, 20, 22, 25

Bibliografia

[1] Edward Felten Andrew Miller Steven Goldfeder Arvind Narayanan Joseph Bonneau. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016. isbn: 9780691171692.

[2] Haibin Zhang Enis Golaszewski Alan T. Sherman Farid Javani. «On the Origins and Variations of Blockchain Technologies». In: (2018). url: https://arxiv.org/ftp/arxiv/papers/1810/1810.06130.pdf.

[3] Danilo Gligoroski Sindre A. Pedersen Arild Faxvaag Anton Hasselgren Katina Kralevska. «Blockchain in healthcare and health sciences». In: (2020). url: https://www.sciencedirect.com/science/article/pii/S138650561930526X.

[4] Morteza Analoui Jamile Khalili Shahrouz. «An anonymous authentication scheme with conditional privacy-reserving for Vehicular Ad hoc Networks based on zero-knowledge proof and Blockchain». In: (2024). url: https://www.sciencedirect.com/science/article/pii/S138650561930526X.

[5] Sennur Ulukus Mustafa Doger. «Transaction Capacity, Security and Latency in Blockchains». In: (2024). url: https://arxiv.org/pdf/2402.10138.pdf.

[6] *Blockchain*. url: https://www.ibm.com/topics/blockchain.

[7] *Chiavi asimmetriche*. url: https://www.criptoinvestire.com/come-funziona-la-crittografia-nelle-blockchain.html.

[8] *Decentralized Identifier*. url: https://www.w3.org/TR/did-core/.

[9] *Mining*. url: https://www.bitcoinmining.com/.

[10] *Self Sovereign Identity*. url: https://www.blockchain4innovation.it/tecnologie/self-sovereign-identity-norme-applicazioni-benefici-e-sviluppi-futuri/.

[11] *Smart Contract*. url: https://www.ibm.com/topics/smart-contracts.

[12] *Tokenizzazione*. url: https : / / www . blockchain4innovation . it / arte / blockchain-e-arte-tra-crypto-arte-e-tokenizzazione/.

[13] *Transizioni*. url: https://affidaty.io/blog/it/2019/07/inside-blockchain-la-transazione-come-e-da-cosa-e-composta/.

[14] *Verifiable Credentials*. url: https://www.w3.org/TR/vc-data-model/.

[15] *Vulnerabilità*. url: https://www.thei.it/blockchain-security-quali-sono-le-vulnerabilita/.

[16] *Zero Knowledge Proof*. url: https://en.wikipedia.org/wiki/Zero-knowledge_proof.