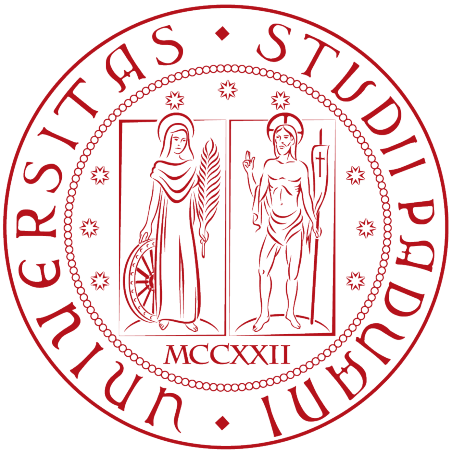
**Università degli Studi di Padova**

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica




# La rivoluzione dell'identità digitale: utilizzo di smart contract nella self-sovereign identity

*Tesi di laurea triennale*

*Relatore*
Prof.ssa Silvia Crafa
*Correlatore*
Dott. Mattia Zago

*Laureando*
Matteo Midena

Anno Accademico 2021-2022

# Abstract

Il problema dell'identità online è qualcosa che è stato discusso e affrontato molto negli ultimi 10 anni. In particolare il processo di evoluzione comprende il passaggio da credenziali fisiche a digitali, in cui si sono risolti i principali problemi legati a un conseguimento burocratico e costoso sia per l'ente di emissione sia per l'utente stesso. Rimane però molto discutibile la tematica della privacy, i nostri dati sono salvati sui server dell'emittente e per usufruire di molti servizi, vengono mostrati a servizi di terze parti che molto spesso hanno un incentivo economico a collezionarli e a salvarli. Lo scopo di questa tesi è la realizzazione di un sistema decentralizzato di identità digitale che permetta all'utente di essere indipendente da un qualsiasi ente centrale. La soluzione basata sul Self-Sovereign Identity è user-centered, l'utente ha il completo controllo e la gestione del consenso delle proprie informazioni. Questo tipo di approccio prevede l'uso di tre componenti principali: credenziali verificabili, decentralized identifier e registri distribuiti. L'uso della tecnologia blockchain come verifiable data registry offre la possibilità di avere un registro trasparente, immutabile, sicuro e decentralizzato. Nello specifico, verrà sviluppata una suite di smart contract che svolga le operazioni di rilascio, verifica e revoca di credenziali verificabili.

# Ringraziamenti

*Innanzitutto, vorrei esprimere la mia gratitudine al Dottor Mattia Zago e alla Professoressa Silvia Crafa, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.*

*Desidero ringraziare e dedicare la tesi ai miei genitori che mi hanno sostenuto economicamente e moralmente, appoggiando ogni mia decisione durante gli anni di studio.*

*Infine vorrei ringraziare i miei amici e in particolare la mia ragazza, per essermi sempre stati vicini e per aver creduto in me, soprattutto nei momenti difficili.*

*Padova, Settembre 2022* Matteo Midena

# Indice

# Elenco delle figure



[1] *SEBA Bank*. URL: https://www.seba.swiss/.

# Capitolo 1

# Introduzione

*In questo capitolo verranno riportate le informazioni riguardanti l'azienda ospitante e brevemente il progetto affrontato.*

## 1.1 L'azienda

Athesys srl è un'azienda italiana collocata a Padova che da decenni si occupa di accompagnare le aziende durante le loro evoluzioni tecnologiche. Si occupa quindi principalmente di consulenza e sono esperti in ambito: Cloud, Database Management, Identity and Access Management e Sviluppo Software.
Lo stage è stato svolto ufficialmente presso Athesys in collaborazione con una loro startup: Monokee. È quest'ultima la realtà che offre soluzioni Identity and Access Management (IAM) per le identità digitali centralizzate e decentralizzate, per il momento l'unica realtà ad abbracciare i due mondi.

**Figura 1.1:** Logo Athesys srl

**Figura 1.2:** Logo Monokee

## 1.2 Il progetto

Come appena descritto, l'azienda si occupa principalmente di consulenze e sviluppo software in ambito identità digitale, ma vista la forte crescita di interesse a livello globale verso la tecnologia blockchain su questo tema, ha deciso di incentrare questo progetto sullo studio, analisi critica e realizzazione di una possibile soluzione che preveda la Self-Sovereign Identity e un'architettura blockchain. La SSI è un modello di identità digitale che, tramite meccanismi crittografici, fornisce all'utente il controllo delle sue informazioni e la capacità di decidere con chi condividerle.
Poichè l'azienda ha già esplorato in parte la realtà decentralizzata, la richiesta nel concreto era di ideare e presentare un Proof-of-Concept (PoC) che preveda l'uso dei kit forniti da **walt.id**[1] (progetto open source che permette di gestire le principali funzioni

[1] *walt.id.* URL: https://walt.id/.

della SSI) per il trattamento delle credenziali verificabili e, se possibile, ricreare parte di quelle funzionalità sulla blockchain attraverso un insieme di **smart contract** sviluppati in Solidity. La tipologia di integrazione concordata con l'azienda è lo sviluppo di una webapp (in questo caso una web DApp) che metta in luce i punti di forza di questo tipo di soluzione e mostri come le due realtà off-chain e on-chain comunicano tra loro. Poichè l'integrazione comporta un progetto abbastanza complesso, sia in termini di difficoltà che di tempo, l'azienda ha ritenuto opportuno spartire il lavoro con un altro stagista. A me è stata affidata la parte relativa alla progettazione e realizzazione degli smart contract, per questo motivo in questo documento verrà discusso prevalemente questo argomento. Mentre l'altro stagista si è occupato di creare un Software Development Kit (SDK) per fornire e gestire le principali funzionalità del kit di walt.id. Lo sviluppo della DApp invece è stato affrontato insieme.
Per raggiungere gli obiettivi richiesti ci si è appoggiati a un front-end sviluppato in ReactJS, utile per effettuare e gestire le chiamate asincrone alla blockchain, un backend in NodeJS per eseguire operazioni di cifratura con chiavi asimmetriche e a una blockchain pubblica Ethereum Virtual Machine (EVM). La webapp realizzata permette quindi di osservare e testare, tramite un'opportuna interfaccia grafica, quelle che sono le principali funzioni in ambito SSI. Tra le tante la creazione e la risoluzione di Decentralized Identifier (DID) off-chain; creazione, verifica e revoca di credenziali verificabili off-chain e in soluzione ibrida.

## 1.3 Pianificazione del lavoro

Lo stage è stato suddiviso principalmente in due periodi, il primo incentrato sullo studio delle tecnologie coinvolte mentre il secondo sulla fase di progettazione e sviluppo. La durata prevista per il periodo di stage è di approssimativamente 300-320 ore, la suddivisione delle ore per ogni compito è stata definita nel piano di lavoro redatto dall'azienda ospitante.
Riporto qui il preventivo delle ore dettagliato:

| **Attività** | **Settimana** | **Ore** |
|---|---|---|
| Formazione sulle tecnologie Self-Sovereign Identity | 1 | 40 |
| Studio di fattibilità ed analisi dei requisiti | 2 | 40 |
| Ciclo 1: smart contract Issuer | 3 | 40 |
| Ciclo 2: smart contract Revoker | 4 | 40 |
| Ciclo 3: smart contract Verifier | 5-6 | 80 |
| Ciclo 4: integrazione smart contract e walt-id sdk | 7-8 | 80 |
| Totale | | 320 |

**Tabella 1.1:** Tabella suddivisione ore lavorative

Ogni ciclo inserito è stato svolto con la metodologia di lavoro AGILE.

# Capitolo 2

# Background

*In questo capitolo verranno esposte quelle che sono le conoscenze tecniche necessarie a comprendere il lavoro svolto durante il periodo di stage.*

## 2.1 Concetto di Identità Digitale

Per comprendere le richieste del progetto e del modello di relazione di identità SSI è necessario avere un'idea chiara di cosa significhi parlare di identità digitale.
L'identità digitale è l'insieme delle risorse digitali associate in maniera univoca ad una persona fisica che la identifica, rappresentandone la volontà, durante le sue attività digitali. L'identità digitale, di norma, viene presentata per accedere a sistemi informatici, sistemi informativi o per la sottoscrizione di documenti digitali. In un'accezione più ampia essa è costituita dall'insieme di informazioni presenti online e relative ad un soggetto.
La rappresentazione dell'identità digitale deve essere tanto più completa quanto è complessa e delicata l'operazione in cui è coinvolta, infatti il grado di affidabilità e le quantità di informazioni richiesti possono variare anche in modo molto significativo. Contemporaneamente devono essere tutelati quelli che sono gli aspetti informativi relativi alla privacy che non competono al tipo di operazione per cui sono richiesti. Un esempio potrebbe essere una situazione in cui è necessario dimostrare nome e cognome, in questo caso le altre informazioni anagrafiche o il numero del documento stesso, non dovrebbero essere comunicate.
Un'identità digitale è articolata in due parti:

* chi è una determinata entità (**identità**)

* le credenziali che ognuno possiede (gli **attributi** di tale entità).

L'identità digitale più semplice consiste semplicemente in un *ID* (o nome utente), che è sufficiente per identificare il portatore.
Poichè si sta parlando sempre di identità ci si deve scontrare con diverse tematiche cardine come: autenticazione, autorizzazione, riservatezza, integrità dei dati, autorevolezza e non ripudiabilità. Senza la garanzia di riuscire a soddisfare questi principi, l'identità digitale non avrebbe nessun valore formale. Proprio sulla base di quest'ultimo punto è possibile valutare l'efficacia o meno di una soluzione.

## 2.2 Problemi di una credenziale fisica e digitale

È opportuno inoltre analizzare quello che è il percorso evolutivo svolto dalle credenziali identificative e i loro problemi.

### 2.2.1 Credenziali fisiche

Per credenziali fisiche si intende un qualsiasi documento in formato fisico, cartaceo o plastificato. Questo tipo di documento molto spesso è ottenibile solo dopo un processo lungo, burocratico e costoso (sia per l'utente che per l'emittente). Possono essere facilmente falsificati, ed esposti a casi di furto di identità; il solo modo di verificarne l'autenticità è quello di contattare direttamente l'ente di emissione, procedura che di rado viene svolta viste le tempistiche richieste. Non sono privati: quando si fornisce un documento per provare qualcosa su noi stessi, il verificatore ha accesso a tutte le informazioni di quella credenziale. Informazioni che nella maggior parte dei casi non sono necessarie. In casi estremi, la sede in cui sono registrate le credenziali potrebbe essere distrutta e non ci sarebbe più nessun modo di confermare l'autenticità di quest'ultime (si pensi a situazioni di guerra o calamità naturali).

### 2.2.2 Credenziali digitali

Le credenziali digitali attualmente più diffuse richiedono di registrarsi con dati differenti di login su ogni nuovo sito e la gestione di un grande numero di password (processo scomodo e rischioso). Anche in questo tipo di credenziali non si ha sempre il controllo di quali dati vengano effettivamente condivisi e con chi. La verifica delle credenziali è vincolata alla disponibilità del relativo emittente (se dovesse venire a mancare non ci sarebbe più la possibilità di verificarle). Un ulteriore punto molto discusso è relativo al fatto che servizi di terze parti solitamente hanno un incentivo economico a collezionare e salvare i nostri dati, con il risultato che la nostra privacy è compromessa. I nostri metadati permettono di collegarci velocemente ed essere tracciabili per avvisi pubblicitari online[1] (nel peggiore dei casi potrebbero anche portare a influenzare un'elezione[2]). Inoltre i nostri dati personali molto spesso sono salvati sui server dell'emittente, le cui banche dati centralizzate diventano spesso bersagli degli attaccanti, con tutti i pericoli legati (breaches, leaks, hacks).

## 2.3 Analisi attuali modelli di relazione di identità digitale

I modelli di relazione di identità digitale, tra l'utente e un servizio o organizzazione, rispecchiano anche quella che è l'evoluzione dell'autenticazione in rete, ognuno di questi cerca di risolvere le problematiche del precedente.

[1] Il Sole 24 Ore. *Scandalizzati da Facebook? I vostri dati sono in vendita da anni.* URL: https://www.ilsole24ore.com/art/scandalizzati-facebook-vostri-dati-sono-vendita-anni-AEocgTXE?refresh_ce=1.

[2] SKY TG24. *Lo scandalo di Cambridge Analytica.* URL: https://tg24.sky.it/mondo/approfondimenti/facebook-gestione-dati-accuse.

### 2.3.1 Siloed

In questo modello ogni organizzazione rilascia una credenziale identificativa a un utente (o gli concede di crearla) per permettergli di accedere ai propri servizi. La fiducia tra la persona e l'organizzazione quindi risiede su un segreto condiviso, tipicamente nella forma di un username e una password, ma possono essere anche segreti di altro tipo come data di nascita, PIN o altri dati strettamente personali. In alcuni casi comprende anche l'uso di un token fisico o biometrico.
Questo scenario di richiesta o creazione deve essere svolto per ogni organizzazione o realtà con la quale un utente vuole interagire. Questo porta sicuramente a una scomoda esperienza per l'utente. Il Siloed è il modello di relazione di identità digitale più vecchio e di gran lunga il più utilizzato oggi.

#### Pro

* modello largamente utilizzato, chiaro e semplice;
* aiuta le organizzazioni a gestire conformità, responsabilità e altri rischi usando la tecnica *"keeping subjects close"* (dall'inglese, mantenere i soggetti vicini), ovvero salvare i dati in casa nei propri server e controllare meglio quello che è il flusso di attori e operazioni sulle loro piattaforme;
* permette di eliminare la necessità di password registrando specifici dispositivi, i quali permettono di autenticarsi con dati biometrici o PIN;
* questo modello consente inoltre di avere delle credenziali uniche per ogni relazione, il che migliora la sicurezza e la privacy, a patto che nomi utente e password non vengano riutilizzati.

#### Contro

* poichè cresce sempre di più la necessità da parte delle organizzazioni di richiedere un'identità digitale per offrire i propri servizi o l'accesso ad alcune delle proprie risorse, è evidente che questo modello non è scalabile e sostenibile in futuro;
* diversi attacchi (come ad esempio quello avvenuto a Equifax[3]) hanno chiaramente mostrato che una falla in un'organizzazione che utilizza questo modello può essere catastrofica, milioni di dati personali sarebbero esposti;
* questo modello, seppur semplice, è quello che offre la peggior esperienza cliente. Obbliga l'utente a dover custodire un numero spropositato di credenziali, che porta molto spesso alla perdita di password, e nel caso peggiore al riuso di quest'ultime;
* non si presta al meglio per l'ambito *Internet of Things*. Solitamente questi sistemi si basano su un metodo chiamato Pre-Shared Key (PSK), che utilizza una chiave segreta condivisa che è stata scambiata precedentemente tra le due entità coinvolte;

[3] *Equifax data breach.* URL: https://www.cnet.com/news/privacy/equifaxs-hack-one-year-later-a-look-back-at-how-it-happened-and-whats-changed/.

* ogni organizzazione che utilizza questo modello deve diventare una specie di esperto in identità e sicurezza, richiesta alquanto difficile da soddisfare e che ha come risultato circa 4 trillioni di dollari per frodi ogni anno[4].

### 2.3.2 Federated o Third-Party IDP

A causa della scarsa esperienza utente del primo modello, si sono introdotti enti di terze parti che rilasciano le credenziali di identità digitali per permettere all'utente di effettuare la login su siti web e servizi. Sono quindi degli Identity Provider (IDP) che rilasciano la credenziale digitale fornendo un'esperienza che è definita **single sign-on**, la quale permette di effettuare la stessa procedura di accesso anche per diversi servizi non per forza relazionati, riducendo quindi il numero di credenziali separate da mantenere. Nel concreto, l'utente farà l'accesso sul IDP, che certificherà per lui l'accesso al servizio usando protocolli come OAuth, SAML o OpenID Connect. La fiducia tra l'utente l'IDP è mantenuta attraverso un modello Siloed, sperabilmente fortificato con fattori addizionali di garanzia (ad esempio richiendo una multi-factor authentication[5]). Gli esempi più conosciuti sono quelli di *"social login"*, come ad esempio "Login with Facebook" e "Login with Google". Queste compagnie diventano quindi dei *middlemen of trust* (o oracoli), opzione che reputo valida per contesti lower-trust (come può essere l'iscrizione ad un forum) ma non per ambienti di high-trust come ad esempio l'accesso a una banca online.

#### Pro

* in contesti lower-trust l'uso di IDP permette agli utenti di accedere a molte applicazioni con una credenziale singola, rendendo la fase di autenticazione più semplice, riducendo gli username e password, e migliorando l'esperienza utente. L'azienda ospitante (Monokee) si occupa di offrire esattamente questo servizio.

#### Contro

* i dati relativi alle identità degli utenti sono centralizzate negli IDP;
* richiede la presenza di un ente di terze parti del quale bisogna avere assoluta fiducia;
* nemmeno questo modello funziona bene in ambito Internet of Things.

## 2.4 In cosa la Self-Sovereign Identity è diversa?

SSI è il terzo modello di relazione di identità, è il più recente e nasce dall'avvento della tecnologia blockchain, di Decentralized Identifiers e Credenziali Verificabili. Rispetto ai modelli precendenti quest'ultimo permette di creare un'identità che l'utente possiede, è sua. Solo lui la detiene nel suo personale digital identity wallet, solo lui può decidere chi può vederla e quali informazioni effettivamente mostrare.
La SSI si basa sulla creazione di un canale peer-to-peer sicuro tra Issuer (emittente), Holder (possessore della credenziale) e Verifier (verificatore della credenziale). Le

[4] Crowe. *The Financial Cost of Fraud.* URL: https://f.datasrvr.com/fr1/521/90994/0031_Financial_Cost_of_Fraud_2021_v5.pdf.

[5] *Multi-Factor Authentication.* URL: https://www.cisa.gov/publication/multi-factor-authentication-mfa.

credenziali SSI sono tamper-proof (a prova di manomissione) attraverso l'uso della crittografia. Le credenziali rilasciate sono private e sotto il nostro controllo, solitamente in un Wallet che fornisce un meccanismo di selezione delle informazioni da divulgare. Sono verificabili ovunque e in qualunque momento anche nel caso in cui l'emittente non esista più (con l'eccezione nel caso in cui si stiano usando private DIDs non scritti sul Ledger).
I dati personali non sono salvati su server centralizzati. Questo significa che gli hacker per rubare delle informazioni dovrebbero attaccare individualmente ogni persona. La SSI cerca di abolire le password, l'unica che è necessario ricordare sarà quella del proprio wallet. Si può affermare quindi che un sistema SSI è portabile (permettere all'utente di verificare la sua identità su più piattaforme e posizioni), privato e sicuro.

### 2.4.1 SSI Wallet e Credenziali Verificabili

Il wallet, che solitamente è un app mobile, è il posto in cui il proprietario salva le sue credenziali digitali rilasciate da altri, come ad esempio passaporto, certificazione del diploma e altre. Alcune di queste credenziali potrebbero essere state firmate digitalmente e questo crittograficamente permette di dimostrare quattro cose a un verificatore:

1. chi (o cosa) è l'emittente;
2. a chi (o cosa) è stata rilasciata;
3. se è stata alterata da quando è stata rilasciata (prova di integrità);
4. se è stata revocata dall'emittente.

La particolarità è che il wallet può contenere credenziali firmate anche dall'utente stesso, infatti quest'ultime possono essere rilasciate e firmate digitalmente da ogni persona, organizzazione o oggetto e verranno poi utilizzate in ogni contesto in cui sono fidate. Questo meccanismo permette di creare delle regole, un'organizzazione infatti può decidere di fidarsi solo di credenziali rilasciate da lei o da un'altra particolare entità.
Lo scambio di queste credenziali avviene attraverso la creazione di un canale diretto criptato tra i due peer coinvolti. La connessione rimane persistente solo sulla volontà di entrambe le entità coinvolte. La fiducia reciproca invece viene stabilita al momento dello scambio delle credenziali e successiva verifica delle firme presenti usando quanto registrato su un ledger distribuito (o su una blockchain). Ogni peer controlla personalmente le informazioni da condividere con altri e può scegliere tra una credenziale intera, parte di questa o una **zero-knowledge proof (ZKP)**[6][7] derivata dalla credenziale stessa.

### 2.4.2 Punti di forza

In questo modello l'utente è letteralmente "sovrano" (per riprendere il nome del modello) del suo SSI wallet e delle credenziali in esso contenute. Nessuno può precludergli l'accesso o eseguire operazioni senza il suo consenso. Con le credenziali verificabili è

[6] *Zero Knowledge Proofs.* URL: https://wiki.hyperledger.org/display/CP/Zero-Knowledge+Proof%3A+Verifying+Blockchain+Transactions+with+Less+Risk.
[7] *Zero-Knowledge Proof: Verifying Blockchain Transactions with Less Risk.* URL: https://www.hyperledger.org/blog/2017/06/06/zero-knowledge-proofs.

possibile controllare se una di queste è stata revocata senza il bisogno di contattare l'emittente. Inoltre un altro grande punto a favore è che il modello SSI è interoperabile e portabile, poichè non dipende da nessuna particolare compagnia o entità associata. Qualsiasi applicazione o agenzia è modulare e rimpiazzabile.
Riassumendo quanto detto in punti distinti:

* **Autenticazione più robusta:** i segreti condivisi possono essere rimpiazzati con le credenziali crittograficamente sicure e firmate digitalmente;

* **Migliore esperienza utente:** l'autenticazione è molto più semplice, si può accedere a molti servizi senza dover comunicare o scambiare segreti;

* **Prevenzione dal phishing:** poichè l'autenticazione è mutuale, in una connessione l'utente sa per certo con chi sta interagendo e lo stesso vale per l'altra entità;

* **Canale di comunicazione privato:** il canale creato tra due peer è privato e sicuro, non ci sono intermediari e può essere utilizzato per comunicazioni di ogni tipo;

* **Migliori relazioni:** l'autenticazione avviene in modo praticamente passivo, quindi non c'è più la necessità di trattare come estranei e non fidati i clienti all'inizio di ogni interazione con i propri sistemi.

* **Stesso modello di resposabilità:** potendo scegliere delle proprie politiche di accettazione delle credenziali ricevute, il sistema SSI può essere utilizzato senza problemi di conformità, similmente a come avviene con il modello Siloed.

### 2.4.3 Punti deboli

Innanzitutto è bene sottolineare che diversi dei punti riportati per il modello SSI, non sono una sua esclusiva. Credenziali verificabili e ZKPs per esempio potrebbero essere teoricamente costruite anche all'interno di soluzioni non SSI, anche se al momento della stesura di questo documento non ho trovato nulla relativamente a soluzioni di questo tipo.
Adottare un sistema SSI comporterebbe la rivisitazione di molti sistemi attuali e interfacce utente, oltre che a un grande lavoro di educazione e formazione dello staff, clienti e organizzazioni stesse.
Infine non bisogna sottovalutare la gestione delle chiavi, che potenzialmente è uno dei talloni d'Achille di tutte le tecnologie basate su blockchain. Se si perdono le proprie chiavi private si perdono i propri dati.

> *"Your Keys, Your Bitcoin. Not Your Keys, Not Your Bitcoin"*[8]

Non esiste quindi un *"Hai dimenticato la password?"*, ma sarà necessario escogitare ugualmente una soluzione analoga per permettere il recupero delle chiavi nel caso di un'adozione di massa.

[8] *Your Keys, Your Bitcoin. Not Your Keys, Not Your Bitcoin.* URL: https://cointelegraph.com/news/antonopoulos-your-keys-your-bitcoin-not-your-keys-not-your-bitcoin.

## 2.5 Modello Self-Sovereign Identity

Per poter comprendere il progetto che verrà presentato è necessario esaminare concretamente da cosa è composto un modello di questo tipo.
Self-Sovereign Identity si basa su tre pilastri principali:

### 2.5.1 Verifiable Credentials

Una credenziale verificabile consiste di diverse informazioni:

* un'informazione necessaria a identificare il soggetto della credenziale, sia essa una foto, un nome o numero identificativo;
* l'informazione relativa all'autorità emittente (ad esempio agenzia nazionale, comune di una città)
* l'informazione relativa al tipo di credenziale (ad esempio passaporto, patente)
* l'informazione legata a specifici attributi o proprietà assunte dall'autorità relativamente al soggetto (ad esempio nazionalità, data di nascita, classe di veicoli che possono essere guidati)
* una prova di come la credenziale sia derivata
* informazioni relative ai vincoli sulle credenziali (scadenze, termini d'uso).

Per esemplificare, una credenziale verificabile contiene le stesse informazioni di una relativa credenziale fisica. La possibilità di usare la tecnologia, ad esempio inserendo una firma digitale, rende le credenziali verificabili più affidabili e non manomissibili rispetto alle loro controparti fisiche.
Importante tenere conto che la verificabilità di una credenziale non implica la veridicità delle affermazioni in essa codificate. Per questo motivo l'emittente può includere dei particolari valori, tra le proprietà della credenziale, per permettere a un verificatore di verificarla ed applicare la sua business logic (scelte adottate dall'organizzazione che ne farà la verifica) per determinare se le asserzioni presenti hanno una veridicità sufficiente per le proprie esigenze. Il valore di questa prova comprende uno o più schemi in grado di fornire sufficienti informazioni a un verificatore per determinare se la prova fornita dall'emittente incontra i requisiti richiesti per essere affidabile. Ognuno di questi schemi è identificato da un tipo.

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://www.w3.org/2018/credentials/examples/v1"
  ],
  "id": "http://example.edu/credentials/3732",
  "type": ["VerifiableCredential", "UniversityDegreeCredential"],
  "issuer": "https://example.edu/issuers/14",
  "issuanceDate": "2010-01-01T19:23:24Z",
  "credentialSubject": {
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "degree": {
      "type": "BachelorDegree",
      "name": "Bachelor of Science and Arts"
    }
  },
  "evidence": [{
```

```
    "id": "https://example.edu/evidence/f2aeec97-fc0d-42bf-8ca7-0548192
        d4231",
    "type": ["DocumentVerification"],
    "verifier": "https://example.edu/issuers/14",
    "evidenceDocument": "DriversLicense",
    "subjectPresence": "Physical",
    "documentPresence": "Physical",
    "licenseNumber": "123AB4567"
  }],
  "proof": {  }
}
```

**Listing 2.1:** Esempio di credenziale verificabile contenente una prova e il relativo schema

### Ecosistema

Quando si parla di credenziali verificabili entrano in gioco vari attori. Ogni attore è accompagnato da un ruolo che varia a seconda del tipo di caso d'uso. I ruoli sono così divisi:

* **Holder:** è l'entità che possiede una o più credenziali verificabili e genera una presentazione verificabile di quest'ultime. Sono holder ad esempio studenti, impiegati e clienti.

* **Issuer:** crea delle asserzioni tra affermazioni (claims) e uno o più soggetti (subject), creando una credenziale verificabile da quest'ultime e trasferendola successivamente a un holder. Fanno parte di queste entità associazioni, organizzazioni non-profit, governi o individui stessi.

* **Subject:** è l'entità su cui si fanno affermazioni. Possono essere esseri umani, animali o oggetti. Nella maggior parte dei casi l'holder della credenziale verificabile coincide con il soggetto, ma in certi casi non è così. Ad esempio un genitore potrebbe essere l'holder delle credenziali verificabili di suo figlio o di un animale domestico, che saranno invece i soggetti. In questi casi si parla di **delegation**.

* **Verifier:** è l'entità che riceve una o più credenziali verificabili per processarle. Possono essere ad esempio impiegati, personale di sicurezza e siti.

* **Verifiable Data Registry:** ruolo che si basa sulla creazione e la verifica di identificatori, chiavi e altri dati (ad esempio schemi, registri, chiavi pubbliche degli issuer), che potrebbero essere richiesti per utilizzare credenziali verificabili.

Uno schema rappresentante il flow delle informazioni tra le entità è riportato in Figura 2.1

### Dati chiave del modello

Dati chiave come claims, credentials e presentazioni vanno a formare le basi di questa specifica.

**Claims** Sono affermazioni relative a un soggetto e sono espresse usando relazioni definite **subject-property-value** (vedi Figura 2.2). Più claims possono essere unite insieme per esprimere un grafo di informazioni su un soggetto.

**Credentials** Una credenziale è un insieme di una o più affermazioni (claims) fatte dalla stessa entità. Possono includere un identificativo e dei metadata che descrivono le proprietà della credenziale, tra cui la firma dell'emittente. Vedi Figura 2.3.

**Presentations** Sono una vera e propria presentazione per esprimere un sottoinsieme di dati di un'entità relativamente al contesto in cui devono essere utilizzate, generalmente in risposta a una proof request. Vedi Figura 2.4.

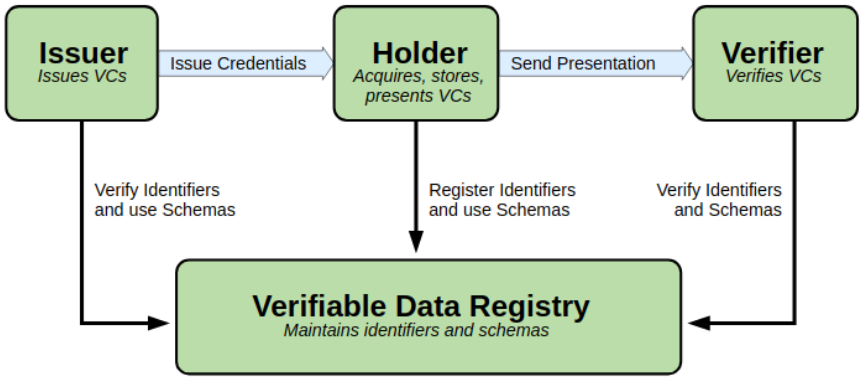
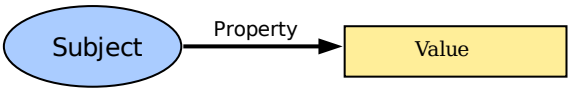



**Figura 2.1:** Immagine che rappresenta i ruoli e il flow che le informazioni seguono nel modello SSI

**Figura 2.2:** Relazione subject-property-value



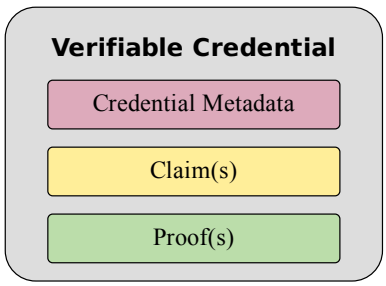




**Figura 2.3:** Componenti di base di una Verifiable Credential

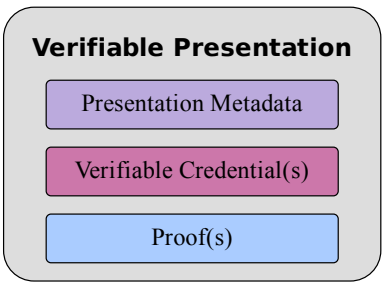



**Figura 2.4:** Componenti di base di una Verifiable Presentation

### 2.5.2 Decentralized Identifiers

Un DID è una semplice stringa composta tra tre parti:

1. la keyword *did* che identifica lo schema;
2. l'identificatore per il *metodo DID*;
3. l'*identificatore specifico* del DID.

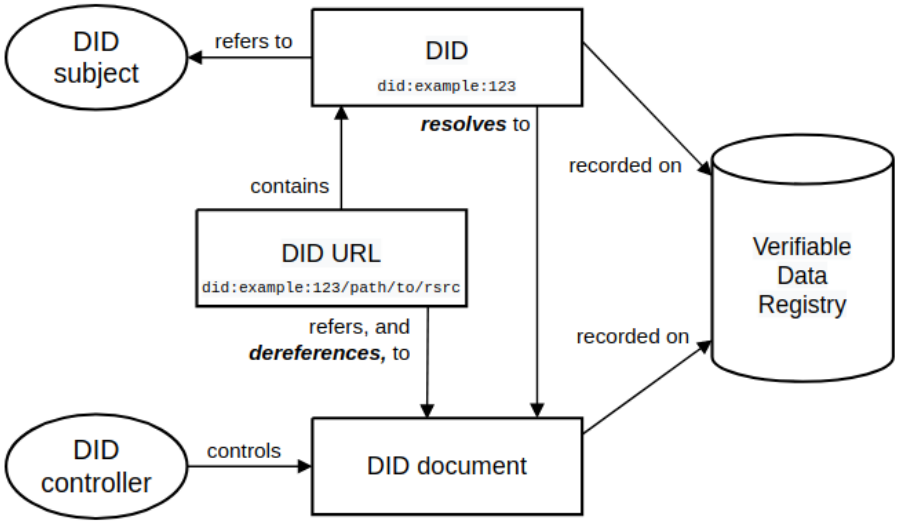
**Figura 2.5:** Semplice esempio di DID


Un DID (come ad esempio quello riportato in Figura 2.5 viene risolto in un Decentralized Identifier document (DID document), quest'ultimo contiene informazioni associate al DID, come le modalità di autenticazione crittografica di un controller DID.
Un'architettura DID può essere composta da diverse componenti che interagiscono tra loro:

* **DID URL:** è conforme allo standard Uniform Resource Identifier (URI), può comprendere path (path-abempty), query, fragment (fornisce un metodo indipendente), parametri (basati sulla sintassi query, diventa parte dell'identificatore).
* **DID controller:** è l'entità che ha la capacità, come definito dal DID method, di fare modifiche a un DID document. Questa capacità solitamente è ottenuta attraverso il controllo di un insieme di chiavi crittografiche (ma non solo in questo modo). Un DID può avere più di un controller e il *DID subject* può essere il controller stesso, o uno di questi.
* **DID document:** è un document accessibile tramite un Verifiable Data Registry (VDR) e contenente le informazioni legate a uno specifico DID, ad esempio il wallet (chiamato in questo caso **repository**) a cui è associato e la chiave pubblica.
* **Verifiable Data Registries:** è un qualsiasi sistema che supporti la registrazione di DID e che ritorni i dati necessari per produrre DID document, indipendentemente dalla specifica tecnologia usata (database o blockchain ad esempio).
* **DID Methods:** sono i meccanismi con cui un particolare tipo di DID e il suo associato DID document sono creati, risolti, aggiornati e disattivati. Vengono definiti in una specifica separata.
* **DID resolver e DID resolution:** un DID resolver è un componente del sistema che preso in input un DID, produce un DID document conforme come output. Questo processo è chiamato DID resolution. I passi per risolvere uno specifico tipo di DID sono definiti all'interno della specifica del DID method.

I Decentralized Identifiers (DID) permettono la creazione di comunicazioni peer-to-peer uniche, private e sicure tra le due parti coinvolte. Senza questo tipo di soluzione, siamo identificati da degli intermediari come Google, Facebook, email provider o operatori

**Figura 2.6:** Schema dell'architettura DID e delle relazioni tra i componenti base




di rete, e come anticipato, ha delle conseguenze in termini di privacy e controllo dei nostri dati. Prendendo un esempio molto semplice come WhatsApp in cui le nostre conversazioni sono criptate, poichè Facebook fa da IDP può ancora vedere e salvare i nostri metadata. Può quindi sapere: con chi ci scriviamo, a quale ora, per quanto tempo, con quali intervalli, da che posizione, quali app si sta usando in quel momento. Usando opportunamente DIDs pubblici o privati si andrebbe a risolvere questo problema. Quelli privati possono essere scambiati tra due parti per creare un canale cifrato e autenticato. È possibile inoltre creare più DIDs separati per altrettante relazioni separate in modo da evitare la correlazione delle informazioni private, senza affidarsi quindi a un'unica autorità intermediaria centrale. Nel caso in cui si usassero solo private DIDs a questo punto gli unici pubblici verrebbero utilizzati strettamente per quelle entità che vogliono essere pubblicamente identificabili (ad esempio uffici, organizzazioni e enti pubblici) che potrebbero registrarli ad esempio sulla blockchain o su un altro tipo di registro pubblico.
Per riportare un esempio reale, supponiamo di voler dimostrare a un bar il fatto di essere maggiorenni. Invece di mostrare la nostra intera carta d'identità, generiamo un QR code dal nostro wallet che lo dimostri. A questo punto il barista lo scansionerà e verificherà l'integrità e l'autenticità del documento, validandone poi la veridicità attraverso il controllo dell'autorità che ha rilasciato la credenziale (Issuer). Quello che succede nel "backend" è che viene verificato il public DID del municipio, lo schema, la definizione della credenziale e viene controllato il registro delle revoche. In una soluzione on-chain, tutto viene registrato nel VDR.

### 2.5.3 Verifiable Data Registry

Non è altro che un registro che serve a tenere traccia dei DIDs in modo che la rete possa interrogarlo per effettuare le operazioni richieste. Solitamente la tecnologia scelta per questo scopo è quella di usare **Trusted Database** o una rete di **ledger distribuiti** (ovvero una blockchain). Nell'ambito del SSI questa tecnologia è in grado di garantire nativamente fiducia tra le parti, autenticità, integrità dei dati e delle attestazioni.

Ulteriori informazioni sul suo funzionamento coincidono principalmente con quelle riguardati la sezione 2.6.
Ormai diversi progetti hanno optato per utilizzare la blockchain come VDR. Ci sono due grandi gruppi e filosofie:

* Hyperledger Indy[9]: con capostipite SOVRIN[10], con implementazioni reali e funzionanti in Columbia Britannica (Canada).
* Hyperledger Besu: con capostipite EBSI[11] e IBSI[12] (derivazione italiana).

Esistono comunque anche progetti progetti privati come ad esempio cheqd[13].

## 2.6 Blockchain

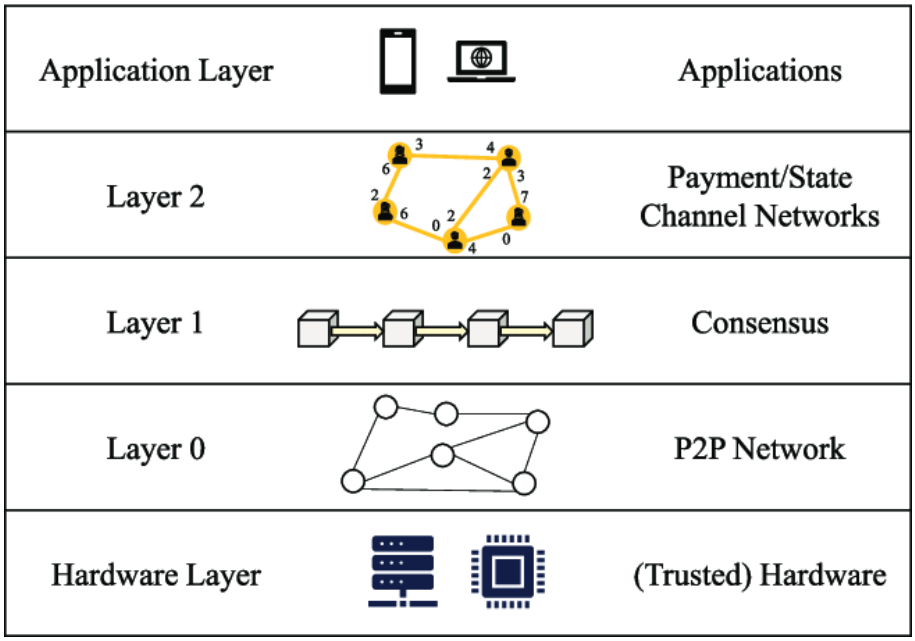
La blockchain, letteralmente significa "catena di blocchi", è una struttura dati condivisa e immutabile. Viene definita come un registro digitale le cui voci sono raggruppate a blocchi che vengono concatenati in ordine cronologico e la cui integrità è garantita dall'uso della crittografia. Si basa sul concetto in cui quando il contenuto viene scritto (tramite un processo normato) questo non è più modificabile né eliminabile pena l'invalidazione dell'intero processo.
Le caratteristiche di base che caratterizzano questa tecnologia sono: digitalizzazione dei dati, decentralizzazione, disintermediazione (non è richiesto nessun intermediario), tracciabilità dei trasferimenti, trasparenza e verificabilità, immutabilità e programmabilità dei trasferimenti. Grazie a tali proprietà, la blockchain si pone come una valida alternativa in termini di sicurezza, affidabilità, trasparenza e costi alle banche dati e ai documenti gestiti in maniera centralizzata da autorità riconosciute e regolamentate (come, tra le altre, banche e pubbliche amministrazioni).

### 2.6.1 Differenze tra blockchain di layer 0, 1, 2

Le blockchain possono essere catalogate in base al tipo di architettura su cui si basano, in particolare possono basarsi su un layer 0, 1 o 2. In base a questo tipo di differenziazione offrono un diverso grado di astrazione e funzionalità. Più si sale di livello, più si va a fare astrazione su quello che è lo "strato fisico".

[9] *Hyperledger Indy*. URL: https://www.hyperledger.org/use/hyperledger-indy.
[10] *Sovrin.org*. URL: https://sovrin.org/.
[11] *What is the European Blockchain Services Infrastructure?* URL: https://ec.europa.eu/digital-building-blocks/wikis/display/EBSIDOC/What+is+EBSI.
[12] *Progetto IBSI*. URL: https://progettoibsi.org/.
[13] *Cheqd.io*. URL: https://cheqd.io/.

**Figura 2.7:** Schema architettura blockchain dall'hardware al livello applicativo




### Layer 0

È il livello più basso, quindi permette di agire direttamente sull'architettura blockchain senza dover mettere mano a layer e protocolli sottostanti. Qui inoltre è possibile creare DApp, validare schemi di dati, "coniare" criptovalute e altro ancora. Tutto l'hardware, server, nodi e qualsiasi device collegato ai nodi fa parte di questo livello. A questo livello agiscono direttamente gli algoritmi di consenso, i principali sono Proof-of-Work (PoW)[14], Proof-of-Stake (PoS)[15].

### Layer 1

Si occupano principalmente di rendere il protocollo base più scalabile. Si basa su due principali soluzioni:

* **Consensus protocol changes**: il primo grande cambiamento dall'algoritmo standard di consenso Proof-of-Work, c'è stato con Ethereum (sottosezione 2.7.1) in cui i progetti hanno iniziato a spostarsi verso il protocollo più veloce e leggero del Proof-of-Stake. Questo è avvenuto su Ethereum con il protocollo Casper[16].

* **Sharding**: si basa sul semplice concetto di dividere le transazioni in "frammenti" più piccoli (traduzione di "shards") che possono essere processati parallelamente dalla rete.

[14] *Proof of Work.* URL: https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/.
[15] *Proof of Stake.* URL: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/.
[16] *Ethereum Casper Explained.* URL: https://academy.binance.com/en/articles/ethereum-casper-explained.

**Pro**

* Aggiunge semplicemente un livello superiore all'architettura, senza apportare grandi modifiche;
* Permette di eseguire un maggior numero di operazioni a parità di tempo;

**Contro**

* **Inefficient Consensus Protocol:** alcuni protocolli di consenso non sono adatti a gestire grandi traffici di transazioni e conseguentemente queste strutture non sono pronte per un'adozione di massa;
* **Excessive Workload:** il carico di lavoro su questo livello può diventare eccessivo e provocare grandi rallentamenti[17] (è il caso di Ethereum e le sue conseguenti elevate gas fee finchè il TheMerge[18] non verrà ultimato).

**Layer 2**

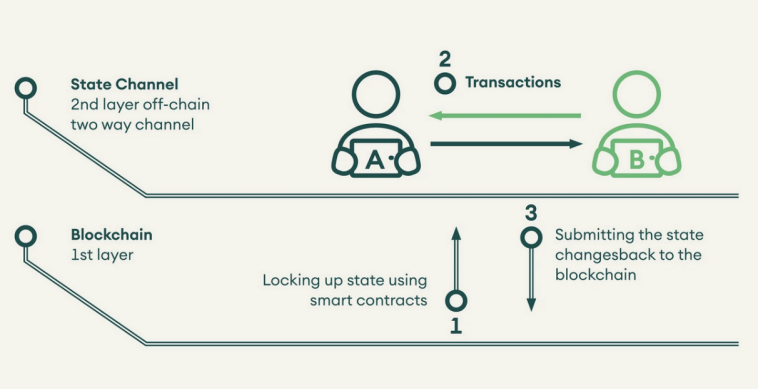
Sono lo strato più alto di astrazione, e attualmente si pongono l'obiettivo di risolvere gran parte dei problemi presenti nel layer 1. Le soluzioni si dividono in due possibilità:

* **State Channels**: ha come componente principale un canale full-duplex tra due partecipanti che permette di svolgere compiti che tipicamente sono on-chain, off-chain. Questo permette di diminuire di molto i tempi di attesa visto che viene a mancare la componente terza, come ad esempio le azioni dei miner. Come funziona:
  - Una porzione di blockchain viene ritagliata (viene usata l'espressione "sealed off", ovvero sigillata) tramite multi-firma o sotto smart contract (sezione 2.9), previo accordo tra le due parti coinvolte;
  - Quest'ultime possono interagire tra loro senza preoccuparsi di firmare le transazioni eseguite
  - Quando la sequenza di scambi termina, solo lo stato finale viene registrato sulla blockchain.

  La metafora più facile è quella di un "conto da saldare", è come se una persona aprisse un conto con un'altra e poi lo saldasse "a rate". Invece di registrare ogni singola rata sulla rete, viene registrato solo il saldo finale.
  Le soluzioni di questo tipo più comuni sono: la **"Bitcoin's Lightning Network"**[19] (permette di eseguire tante microtransazioni) e la **"Ethereum's Raiden Network"**[20] (permette anche di eseguire smart contracts (sezione 2.9) sui canali). Altre informazioni su *State Channels*[21].

[17] *Understanding Blockchain Layers.* URL: https://cointelegraph.com/blockchain-for-beginners/a-beginners-guide-to-understanding-the-layers-of-blockchain-technology.
[18] *The Merge aggiornamenti.* URL: https://ethereum.org/en/upgrades/merge/.
[19] *Lightning Network.* URL: https://lightning.network/.
[20] *Raiden Network.* URL: https://raiden.network/.
[21] *State Channels.* URL: https://ethereum.org/en/developers/docs/scaling/state-channels/.

**Figura 2.8:** State channel e passi necessari a fare una transazione. Sorgente: Seba Bank[22]




* **Nested blockchains**: si basa sulla soluzione chiamata Plasma[23], caratterizzata come segue:
  - La rete principale (main chain) stabilisce le regole di base dell'intero sistema e non prende parte alle operazioni svolte a meno che non debba risolvere dispute;
  - Si basa su un sistema in cui la main chain si appoggia a multipli livelli di blockchain. Questi livelli sono connessi tra loro in modo da formare una struttura del tipo padre-figlio. La chain padre delega il lavoro alla chain figlia in grado di eseguire quella particolare istruzione e che ritornerà poi il risultato indietro al padre.
  - Questo sistema permette di ridurre il carico della root chain e se costruita e usata correttamente è in grado di aumentare la scalabilità della rete esponenzialmente.

**Pro**

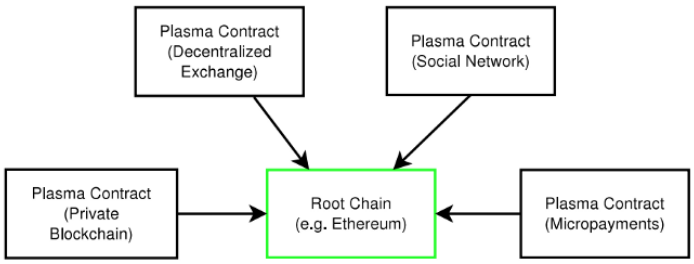
* Evita problemi di disordine per quanto riguarda i compiti di una blockchain;
* Evita di pagare inutili fees su operazioni poco rilevanti o microtransazioni (pagamenti low-value, ovvero sotto il valore del dollaro);

**Contro**

* Minore controllo sui dati a basso livello;

[23] *Plasma*. URL: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/plasma/.

**Figura 2.9:** Esempio di architettura Nested Blockchain




### 2.6.2 Confronto blockchain permissionless e permissioned

Un'ulteriore importante distinzione tra le blockchain si ha con la gestione degli accessi a quest'ultima.

#### Permissioned

Sono blockchain chiuse o che hanno un layer che si occupa di gestire gli accessi. Questo stesso layer si occupa anche di gestire le azioni che i partecipanti della rete sono autorizzati a svolgere.
Il concetto chiave è che quindi un utente può accedere alla rete, leggere e scrivere informazioni in blokchain solo se gli viene dato l'accesso e se ne ha i permessi richiesti. Questo tipo di rete quindi supporta anche un certo grado di personalizzazione, ad esempio la verifica dell'identità di nuovi utenti può essere fatta dalle persone presenti in rete, invece che esclusivamente dal proprietario della rete. Ed è possibile ovviamente personalizzare le attività possibili in rete.
Proprio per questo motivo le permissioned blockchain possono essere centralizzate o parzialmente decentralizzate.
Un esempio di permissioned blockchain è Ripple[24].

**Consensus model** L'algoritmo di consenso usato in queste reti solitamente è differente da quello utilizzato per la controparte permissionless, i principali sono:

- **Practical Byzantine Fault Tolerance (PBFT) consensus:** è un algoritmo voting-based. In questo modello, la sicurezza della rete è garantita finchè la percentuale minima richiesta di nodi si comporta in modo onesto e funziona correttamente. Un esempio potrebbe essere Hyperledger Fabric (sezione 2.7.2).

- **Federated consensus:** si basa su un numero limitato di firmatari fidati per ogni nodo della blockchain. Utilizzano un unico generatore di blocchi che riceve le transazioni e le filtra di conseguenza. Un esempio è la blockchain Corda[25].

- **Round-robin consensus:** in questo modello i nodi sono selezionati in modo pseudo-randomico per creare blocchi, ogni nodo quindi deve aspettare un certo numero di cicli prima di essere scelto nuovamente per aggiungere un nuovo

[24] *What is Ripple?* URL: https://www.forbes.com/advisor/investing/cryptocurrency/what-is-ripple-xrp/.
[25] *Corda.* URL: https://www.corda.net/.

blocco. Sembra che questo modello non sia stato ancora scelto per nessuna implementazione rilevante.

#### Pro

* alto livello di privacy e sicurezza;
* flessibile: può essere parzialmente decentralizzata o completamente centralizzata in base ai casi d'uso;
* altamente personalizzabile: come descritto precedentemente, può soddisfare configurazioni e integrazioni basate sui bisogni dell'organizzazione;
* la sua grandezza limitata rende queste reti scalabili e performanti

#### Contro

* grande mancanza di trasparenza, questo si traduce in possibili rischi di corruzione (il controllo è nelle mani di un gruppo privato);
* la sicurezza della rete dipende direttamente dell'integrità dei suoi membri, un'alterazione dei dati per un loro tornaconto è sempre possibile;
* sono regolamentate e censurate, non c'è una libertà completa.

#### Permissionless

È il modello più diffuso tra le principali criptovalute (come Bitcoin e Ethereum). Come si evince dal suo nome, queste reti permettono l'accesso a chiunque, sono quindi decentralizzate e pubbliche. La parola "permissionless" implica la mancanza di vincoli o censura sulle possibili azioni.
Tecnicamente, come da protocollo, ognuno può usare la rete e farci quello che vuole. Le permissionless blockchain si avvicinano al concetto originale di blockchain sviluppato da Satoshi Nakamoto[26].
Questa accessibilità al pubblico si paga in termini di trade-off (dall'inglese compromesso) sulla velocità, spesso queste reti infatti risultano più lente della controparte permissioned, che ha molti meno membri.
Le transazioni, che contengono le informazioni salvate in blockchain, sono validate direttamente dal pubblico. Il principale meccanismo di consenso in uso è il PoW e PoS.

#### Pro

* alta trasparenza;
* decentralizzata, le informazioni non sono in repository centralizzate e questo rende i dati sicuri, accessibili e affidabili per tutti (sono per questo considerati praticamente non hackerabili);
* sicurezza, un attaccante dovrebbe attaccare con successo almeno il 51% della rete per soverchiare il meccanismo di consenso.

[26] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* URL: https://bitcoin.org/bitcoin.pdf.

**Contro**

* grande richiesta di energia e potenza di calcolo per raggiungere in consenso;
* per sua natura le informazioni sono pubbliche, quindi presenta problemi di privacy;
* l'anonimicità della rete non offre possibilità di negare l'accesso a utenti malevoli o malintenzionati.

## 2.7 Attuali blockchain rilevanti

In questa sezione vengono discusse e riportate brevemente le principali blockchain, utili a una completa comprensione del progetto.
Bitcoin è il primo protocollo blockchain creato nel 2008 da Satoshi Nakamoto, che lo ha descritto originariamente come una soluzione alla necessità di un "sistema di pagamento elettronico basato sulla prova crittografia anzichè sulla fiducia". Dal 2009 è stato rilasciato come progetto open source e attualmente è la blockchain e la criptovaluta più conosciuta e dominante.
Poichè il protocollo bitcoin si basa sul rustico algoritmo di consenso Proof-of-Work e non prevede nativamente l'abilitazione allo sviluppo di smart contract (sezione 2.9) non verrà approfondito ulteriormente. Ulteriori informazioni su *Bitcoin.org*[27].

### 2.7.1 Ethereum

Ethereum è una blockchain **permissionless**, ossia aperta a chiunque voglia interagire con essa: il compromesso è l'introduzione delle **fee**, ossia una "tassa" da pagare ogni volta che si vuole modificare lo stato della rete (quindi scrivere una transazione), per mitigare i problemi che introduce il fattore permissionless (ad esempio lo spam di transazioni). In caso di sola lettura non occorre pagare alcuna fee.
Chiunque quindi può visionare cosa accade nella blockchain, e soprattutto può utilizzarla. Per farlo, basterà generare un **wallet** (ossia un indirizzo nella blockchain), trasferirci alcuni fondi dall'esterno per poter pagare le fee, e iniziare ad interagire con le applicazioni decentralizzate che sono già sviluppate, oppure semplicemente trasferire fondi ad altri wallet.
Per poter effettivamente utilizzare delle applicazioni su Ethereum vengono utilizzati gli **Smart Contracts** (sezione 2.9). Altre informazioni su *Ethereum.org*[28].

### 2.7.2 Hyperledger

Hyperledger Foundation è un'organizzazione **senza scopo di lucro** che riunisce tutte le risorse e le infrastrutture necessarie per garantire ecosistemi fiorenti e stabili attorno a progetti blockchain di software open source.
Il personale della Hyperledger Foundation fa parte di un team più ampio della **Linux Foundation** che vanta anni di esperienza nella fornitura di servizi di gestione di programmi per progetti open source.
Tra questi progetti verranno riportati solo quelli basati sull'uso di smart contract

[27] *Bitcoin.org*. URL: https://bitcoin.org/en/.
[28] *Ethereum.org*. URL: https://ethereum.org/en/.

(sezione 2.9), essendo i più rilevanti al fine di questo documento. Altre informazioni su *Hyperledger.org*[29].

### Besu

Hyperledger Besu è un client Ethereum progettato per essere adatto alle imprese per i casi di utilizzo di **reti pubbliche e private permissioned**, che richiedono un'elaborazione delle transazioni sicura e ad alte prestazioni.
La blockchain Besu è EVM compatibile: questo significa che dal punto di vista degli sviluppatori e degli utenti, l'interazione con essa sarà molto simile a quella con Ethereum. Pone però particolare attenzione sulle funzionalità di **privacy e permissioning**, infatti solamente chi è autorizzato (ossia chi possiede un nodo connesso) può interagire con il sistema, e questa è la differenza sostanziale con Ethereum.
La privacy viene considerata sia a livello esterno alla rete, sia a livello interno: tramite le **transazioni private** non tutti i nodi possono accedere a particolari informazioni, e i nodi che vogliono usufruire delle transazioni private devono avere un nodo **Tessera** associato, che si occuperà della parte crittografica. Altre informazioni su *Hyperledger Besu*[30].

**Permissioning** Una permissioned network è una rete che gestisce i permessi su un insieme di nodi e account, permettendo a solo quelli autorizzati di accedere alla rete. In particolare quello che si può fare relativamente ai permessi sugli account è:

* richiedere dettagli sull'identità,
* sospendere gli account,
* escludere contratti invalidi in una denylist,
* limitare le azioni che un account può compiere.

La specifica dei permessi può essere specificata in due modi:

**Localmente:** si lavora a livello del nodo, ogni nodo della rete ha un file di configurazione dei permessi. Questi tipi di permessi hanno effetto sul nodo in questione ma non sul resto della rete. Questo significa che non è necessario un coordinamento con il resto della rete e si può agire direttamente per la protezione del nodo.

**Onchain:** si attua attraverso uno smart contract (sezione 2.9) sulla rete. Specificare i permessi onchain implica che tutti i nodi possono leggere e aggiornare la configurazione dei permessi. Questo tipo di gestione richiede coordinazione per aggiornare le regole e la rete potrebbe non attuarle immediatamente in quanto, ad esempio, lo smart contract (sezione 2.9) potrebbe forzare un minimo numero di voti prima di attuare la modifica.

### Tessera

Innanzitutto è opportuno riprendere quello che il concetto di privacy, ovvero l'abilità di mantenere le transazioni private tra i partecipanti coinvolti. Questo significa che gli altri partecipanti non devono poter accedere al contenuto delle transazioni o alla lista dei partecipanti. Besu usa un private transaction manager, Tessera, un progetto open source sotto Apache 2.0[31] scritto in Java, che ha il compito principale di gestire la

[29] *Hyperledger.org*. URL: https://www.hyperledger.org/.
[30] *Hyperledger Besu*. URL: https://www.hyperledger.org/use/besu.
[31] *Apache License 2.0*. URL: https://www.apache.org/licenses/LICENSE-2.0.

privacy sui client Ethereum che hanno privacy-enabled (dall'inglese "privacy attivata", come Hyperledger Besu appunto). Le principali funzioni di Tessera sono:

* generare e mantenere le coppie chiavi private e pubbliche
* gestire e scoprire tutti i nodi nella rete
* fornisce un'interfaccia Application Program Interface (API) per la comunicazione tra i nodi Tessera e un API per la comunicazione con i client Ethereum.

Ogni nodo Besu che riceve e invia transazioni private necessita un associato nodo Tessera. Quindi le transazioni passano dal nodo Besu all'associato nodo Tessera. Quest'ultimo cripta la transazione privata e la distribuisce point-to-point (punto a punto) ai nodi Tessera partecipanti alla transazione.
Di default, ogni partecipante a una private network utilizza il suo nodo Besu e Tessera, e l'uso di un sistema multi-tenancy permette a più di un partecipante di usare lo stesso nodo Besu e Tessera.

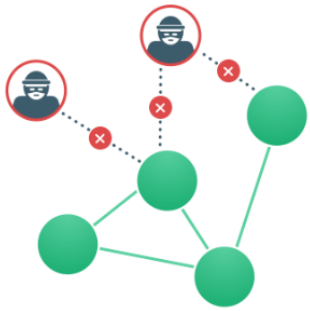


**Figura 2.10:** Chi non ha accesso alla rete non può interagirci in nessun modo

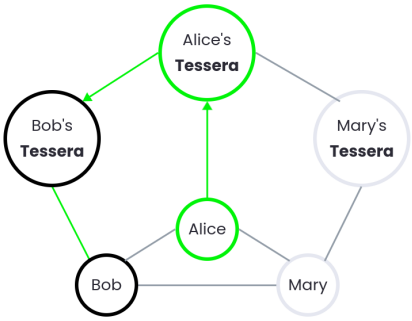



**Figura 2.11:** Alice manda una transazione privata a Bob, e per Mary non sarà visibile

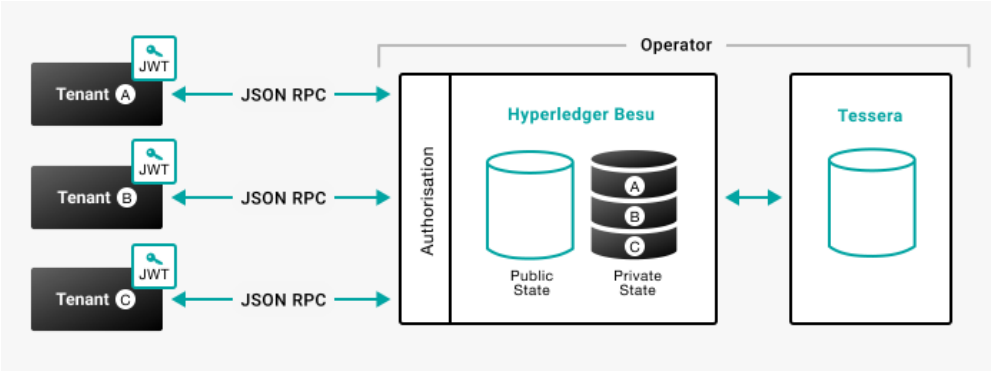



**Figura 2.12:** L'amministratore della coppia di nodi Besu e Tessera può dare l'accesso ad altri tenant, ossia utenti.

**Fabric**

Hyperledger Fabric è una Distributed Ledger Technologies (DLT). Prevede controlli avanzati sulla privacy così che solo i dati che si desidera condividere vengano condivisi **tra i partecipanti** alla rete noti come "autorizzati".
Offre un'architettura **modulare** che rende disponibili componenti (tra i quali, meccanismo di consenso, servizi per l'adesione e la gestione dei membri della blockchain) che, con la logica del plug and play, possono essere attivati nell'ambito di una blockchain.
Si può affermare che è molto **simile a Besu** (anch'essa rete permissioned e privacy-oriented), con la grossa differenza che **non è EVM** compatibile, quindi gli smart contract (sezione 2.9) verranno scritti in linguaggi come Java e Go, invece di Solidity.
Altre informazioni su *Hyperledger Fabric*[32].

## 2.8 SSI adottata da EBSI

European Blockchain Services Infrastructure (EBSI) è un'infrastruttura di servizi che permette di gestire le **credenziali** di identità e di istruzione dei cittadini europei. Questi casi d'uso mirano a facilitare la mobilità di studenti, giovani professionisti e imprenditori, nonché a **garantire e verificare l'autenticità** delle informazioni digitali in diversi settori.
La sua struttura è piuttosto complessa, composta da due layer fondamentali: il **layer inferiore** è cositutito dalla **blockchain** di EBSI, che custodisce tutte le credenziali e gli smart contract (sezione 2.9) con i quali si può interagire con esse, mentre il **layer superiore** implementa un'architettura a **microservizi** ed API, che si interfacciano con la blockchain.
In pratica, per dialogare con la blockchain di EBSI (se non si dispone di un nodo) occorre interfacciarsi tramite l'architettura a microservizi e API messe a disposizione.
Le blockchain utilizzate da EBSI sono Hyperledger Fabric e Hyperledger Besu.

In Figura 2.13 è stato riportato un esempio relativo al flow dell'iterazione di una normale operazione con EBSI. Come si può vedere sia gli utenti che le organizzazioni si interfacciano, utilizzando un wallet, con le API, questo significa che nessuno ha davvero idea di cosa facciano effettivamente gli **smart contract** (sezione 2.9) di EBSI poichè **non sono pubblici** (anche se dalle ultime disposizioni sembra che presto saranno resi open source[33]) e l'unica cosa su cui si può fare riferimento è la relativa documentazione fornita. Questo è un esempio di quello che significa utilizzare una blockchain permissioned, il **controllo** è completamente **detenuto da EBSI** stessa.
Probabilmente si può considerare come una forma di *"security by oscurity"*
Saranno le API a questo punto che interagiranno direttamente con gli smart contrat, in cui sono presenti tutte le informazioni relative ai *trusted registries*, DID (sottosezione 2.5.2) e altro.

[32] *Hyperledger Fabric*. URL: https://www.hyperledger.org/use/fabric.
[33] *New steps in the development of the European Blockchain Services Infrastructure (EBSI)*. URL: https://digital-strategy.ec.europa.eu/en/news/new-steps-development-european-blockchain-services-infrastructure-ebsi.

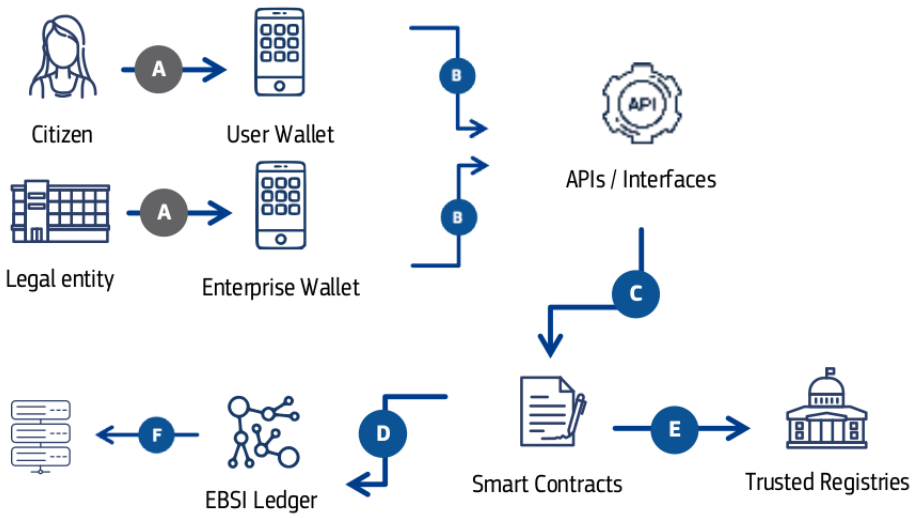



**Figura 2.13:** Il flow dell'interazione con EBSI

## 2.9 Smart contract

Sono essenzialmente dei programmi salvati sulla blockchain che convertono i tradizionali contratti nella loro controparte digitale. Hanno quindi il vero e proprio obiettivo di digitalizzare i termini di un accordo in codice che viene eseguito quando i termini del contratto vengono rispettati. Viene identificato da un indirizzo on-chain.
L'uso della parola contratto comunque può essere un po' fuorviante, poichè fuori da contesti normativi in alcuni casi non è possibile parlare di contratti (in senso strettamente giuridico). Si possono identificare invece come un insieme di **funzioni "if/then"** che rispettano determinati protocolli.
Le applicazioni che si basano su smart contract vengono denominate solitamente DApp.

### 2.9.1 Come funzionano

Attualmente, Ethereum è la piattaforma di smart contract più diffusa. Tuttavia molte altre blockchain per criptovalute (tra cui Tezos[34], Avalance[35], Fantom[36] e Algorand[37]) sono in grado di eseguire uno smart contract. Chiunque può creare e distribuire uno smart contract su una blockchain. Il codice è trasparente e verificabile pubblicamente. Ogni partecipante può vedere esattamente quale logica segue uno smart contract quando riceve determinati input.
Gli smart contract sono scritti in numerosi linguaggi di programmazione (tra cui Solidity[38], Web Assembly[39], Rust[40], Vyper[41]). Per quanto riguarda la rete Ethereum,

[34] *Tezos*. URL: https://tezos.com/.
[35] *Avalanche*. URL: https://www.avax.network/.
[36] *Fantom*. URL: https://fantom.foundation/.
[37] *Algorand*. URL: https://www.algorand.com/.
[38] *Solidity*. URL: https://docs.soliditylang.org/.
[39] *WebAssembly*. URL: https://webassembly.org/.
[40] *Rust*. URL: https://www.rust-lang.org/.
[41] *Vyper*. URL: https://vyper.readthedocs.io/en/stable/.

il linguaggio utilizzato è Solidity e il codice di ogni smart contract è memorizzato sulla blockchain. Qualsiasi partecipante è in grado di esaminarlo, accertandosi del suo stato attuale, al fine di verificarne le funzionalità. Ogni computer sulla rete (denominato anche "nodo") memorizza una copia di tutti gli smart contract esistenti e del loro stato attuale nella blockchain, oltre ai dati delle transazioni.
Quando uno smart contract riceve fondi da un utente, il suo codice viene eseguito da tutti i nodi della rete, al fine di raggiungere il consenso in merito all'esito e al flusso di valore risultante. Questo meccanismo consente l'esecuzione sicura degli smart contract, senza che sia necessaria un'autorità centrale.
Per eseguire uno smart contract sulla rete Ethereum, è necessario versare una commissione denominata "gas" (come anticipato nella sottosezione 2.7.1). Anche se esistono alcune eccezioni[42], in generale, una volta distribuiti sulla blockchain, gli smart contract non posso più venire modificati, neanche dal loro autore. Ciò garantisce che il processo sia a prova di censura e che non venga interrotto.

### 2.9.2 Limitazioni

Una prima limitazione è data dal fatto che gli Smart Contract da soli non possono ottenere informazioni sugli eventi del mondo off-chain perché **non** possono inviare **richieste HTTP**, questo problema è noto anche come **"The Oracle Problem"**[43].
Sono stati progettati così e basarsi su informazioni esterne potrebbe pregiudicare il consenso, fondamentale per la sicurezza e la decentralizzazione. Esistono comunque modi per aggirare questa condizione, attraverso l'uso degli oracoli[44], i principali sono quelli sviluppati da Chainlink[45].
Un altro limite degli smart contract è la dimensione massima del contratto. Uno smart contract può avere una **dimensione massima di 24KB**[46][47], problema che è stato risolto con l'introduzione del Diamond Pattern (schema a Diamante)[48].

### 2.9.3 Quali sono i rischi legati agli Smart Contract

Blockchain e smart contract non sono immuni a cyber incidenti, se ne sono verificati diversi negli ultimi anni. I fattori che solitamente portano a grandi perdite di denaro sono: furto di fondi, caduta del prezzo del token associato a una DApp, perdita di investitori e altro. Questo nel peggiore dei casi può portare al totale fallimento della propria attività.
Proprio alcune particolarità degli smart contract possono generare vulnerabilità e facilitare la loro violazione. Ad esempio il codice di uno smart contract è **pubblico**, mentre nelle classiche applicazioni web, parti di quest'ultima sono nascoste (come ad esempio il backend). Il bytecode di un contratto compilato è caricato in blockchain e può essere facilmente effettuato un reverse engineered del codice. Un attaccante quindi ha tutto ciò che gli serve per osservere il flow delle operazioni e cercare vulnerabilità. Questo significa che avrà senso eseguire solo test di sicurezza di tipo **White Box** e non

[42] OpenZeppelin. *Writing Upgradeable Smart Contracts*. URL: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable.
[43] *What is the Blockchain Oracle Problem?* URL: https://blog.chain.link/what-is-the-blockchain-oracle-problem/.
[44] *Oracles*. URL: https://ethereum.org/en/developers/docs/oracles/.
[45] *Chainlink*. URL: https://chain.link/.
[46] *WHY IS THERE A LIMIT?*. URL: https://ethereum.org/en/developers/tutorials/downsizing-contracts-to-fight-the-contract-size-limit/.
[47] *EIP-170: Contract Size Limit*. URL: https://eips.ethereum.org/EIPS/eip-170.
[48] *EIP-2535: Diamond Standard*. URL: https://eips.ethereum.org/EIPS/eip-2535.

**Black Box**. La prima tipologia prevede che l'attaccante (ovvero il tester del sistema) sia informato dell'infrastruttura e dei servizi esistenti. In questo modo vengono ridotti anche di molto i tempi di esecuzione del test, visto che non sarà necessario eseguire una prima fase di Information Gathering (dall'inglese "raccolta di informazioni"). Inoltre, questo tipo di controllo è molto più preciso, poichè l'attaccante si concentra meglio sul target specificato e si ottiene una copertura più completa di test.
La modalità Black Box invece è molto più verosimile a un attacco vero e proprio apportato da un attaccante esterno. In questo caso, infatti, il tester non avrà informazioni né sull'infrastruttura né sul target da analizzare. Ovviamente questo sistema richiede più tempo. Il report che verrà generato è da considerarsi più verosimile ad un attacco reale, ma le criticità individuate potrebbero non essere tutte, visto che l'attaccante non conosce l'esatta composizione dell'infrastruttura software e hardware.
Un altro aspetto di cui tenere conto è l'**immutabilità**, una volta che un contratto è stato caricato in blockchain non si torna indietro, è una cosa di design della blockchain. Non c'è quindi una seconda possibilità per modificarlo e avere un codice sicuro. È comunque sempre possibile implementare una funzione "self kill" che renda il contratto vecchio inutilizzabile e trasferisca i fondi sul contratto nuovo, ma il caricamento di un nuovo contratto potrebbe avere dei costi comunque non trascurabili.
Alcuni esempi di attacchi recenti dovuti a bug negli smart contract e in questo tipo di architetture includono tra le tante: Ronin[49] (perdita di circa 625 milioni di dollari), Wormhole[50] (perdita di circa 326 milioni di dollari), Poly Network[51] (perdita di circa 602 milioni di dollari). Lo storico di tutti gli attacchi più rilevanti sono elencati sul sito di DEFIYIELD[52].

## 2.10 NFT

Un NFT ha lo scopo principale di dimostrare la proprietà di un contenuto digitale di una persona, in particolare del wallet ad essa associato.
Gli NFT solitamente si compongono di due parti, una parte viene salvata on-chain mentre l'altra off-chain. Il principale motivo è dovuto al fatto che salvare tutto sulla blockchain può risultare molto costoso, quello che viene registrato on-chain quindi corrisponde esclusivamente alla parte relativa alla dimostrazione di identità. off-chain, invece solitamente si vanno a salvare i metadata degli NFT. Poichè gli NFT non sono facilmente aggiornabili dopo la loro creazione è bene pensare a come i relativi dati sono salvati, indicizzati e resi persistenti nel tempo.

### 2.10.1 Content Addressing

Analizzando quello che il modello attuale utilizzato per la risoluzione degli indirizzi sul web, quello che succede è che per accedere a *sito-esempio.it/sottodominio/fonte* il nostro sistema operativo andrà a interrogare un sistema di chiavi-valore condiviso, e suddiviso in diversi domini: non è altro che il Domain Name System (DNS). Il DNS ritornerà un indirizzo IP che la nostra scheda di rete userà per inviare una richiesta HTTP sulla rete e ricevere un payload di risposta. Questo sistema funziona comunque discretamente bene, ma per questo tipo di soluzione presenta un grande problema: il **tempo**.

[49] *Ronin*. URL: https://bridge.roninchain.com/.
[50] *Wormhole*. URL: https://wormhole.com/.
[51] *PolyNetwork*. URL: https://www.poly.network/.
[52] *DeFi Yield Attacks Database*. URL: https://defiyield.app/rekt-database.

Ogni componente di un indirizzo web è mutabile, ovvero possono cambiare nel tempo. Ad esempio, se ci dovessimo dimenticare di pagare l'affitto di un dominio, questo potrebbe scadere e lo comprerebbe il miglior offerente. Un altro esempio potrebbe essere quello di decidere di cambiare dominio e non provvedere a dei redirect, portando il nostro link a ritornare un errore 404 (fenomeno del *Link Rot*[53] a cui ormai siamo abituati).
Il contesto dell'attuale web, dove tutto è mutabile e dinamico, non è quindi adatto a ospitare o indirizzare contenuti digitali come gli NFT che necessitano invece di persistenza e altre garanzie.
Un altro aspetto che forza una soluzione ibrida è quella relativa al costo della memoria. In blockchain è molto più costosa, si stima che nel maggio del 2021 il costo necessario per salvare un megabyte di dati direttamente su Ethereum fosse approssimativamente 21.5 Ether, che al tempo corrispondeva a circa $56000 USD.

#### Necessità di un link "robusto"

Per collegare in modo sicuro un NFT al relativo riferimento off-chain è necessario avere link che siano resistenti al tempo. Il link ideale dovrebbe sempre riferire lo stesso contenuto originale e non dovrebbe essere legato a server o domini mono-proprietari.
La scelta migliore quindi è utilizzare un content-addressed system, che funzionano come i classici sistemi che utilizzano riferimenti chiave-valore, ma con una grande differenza: non è più necessario scegliere le chiavi. Quest'ultime infatti sono derivate direttamente dai valori che sono salvati usando una funzione deterministica che genera sempre la stessa chiave per lo stesso contenuto.

#### Dove si usa il content addressing?

Il modo più semplice è usare Interplanetary File System (IPFS)[54], che è un protocollo decentralizzato e una rete peer-to-peer per salvare e condividere dati in un file system distribuito.
Quando i dati sono salvati usando IPFS, gli utenti possono recuperare i dati salvati da uno qualsiasi dei nodi della rete che ne ha una copia, fornendo anche trasferimenti più veloci e riducendo i tempi di caricamento.

### 2.10.2 Content persistence

Riprendendo il discorso legato al sistema utilizzato attualmente per il web, quest'ultimo è noto come **local addressing** e si riferisce al processo di recuperare un'informazione online da una specifica posizione sul web (ad esempio tramite la risoluzione di un Uniform Resource Locator (URL)). Si può dichiarare inoltre, che questo tipo di indirizzamento sia centralizzato, poichè chi controlla un determinato indirizzo ne controlla anche il contenuto. Questo significa che gli URL che usano local-addressing sono exploitabili.
La soluzione a questo problema si ha sempre con il content addressing sopra citato. Questo tipo di indirizzamento si basa su un'impronta digitale unica (chiamata anche hash[55]) dei dati caricati. Non importa quindi dove i dati siano salvati, se si conosce la firma, si può accedere al contenuto del dato. Questo risolve il problema della centralità

[53] *Link rot*. URL: https://www.techopedia.com/definition/20414/link-rot.
[54] *What is IPFS?*. URL: https://docs.ipfs.tech/concepts/what-is-ipfs/.
[55] *IPFS Hash*. URL: https://docs.ipfs.io/concepts/hashing/#important-hash-characteristics.

dei dati, usando ad esempio IPFS, poichè il dato può essere recuperato da un qualsiasi nodo partecipante alla rete che ne ha una copia (o composto prendendo bits da più di uno).
Rimane comunque il problema legato alla persistenza del contenuto, è chiaro che la miglior soluzione sia quella di avere un sistema decentralizzato, che comprenda un numero di nodi separati che comunichino e che garantiscano, tramite la crittografia, la protezione delle informazioni dalle azioni (o inazioni) di un sistema centralizzato.
Usando quindi una piattaforma come IPFS. i dati avranno associato un Content Identifier (CID)[56] che è derivato direttamente dai dati stessi e si riferisce direttamente a questi sulla rete. Questo significa che nessuno può alterare il contenuto dei dati senza rompere il link.

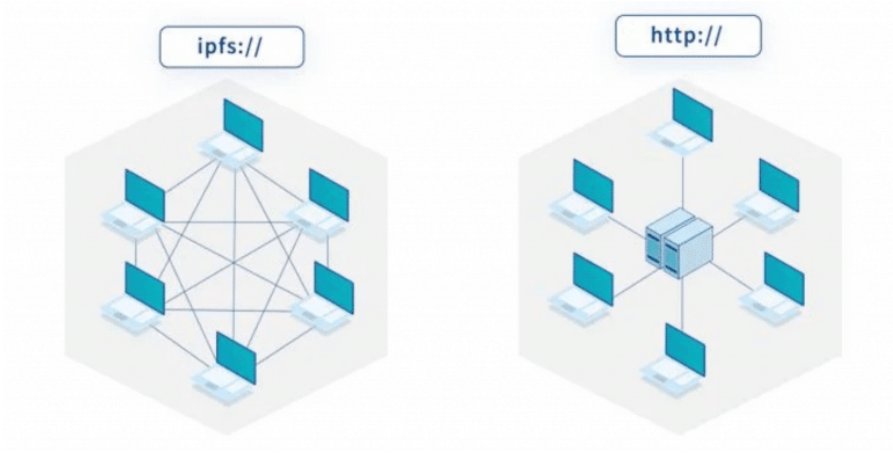



**Figura 2.14:** Schema differenza IPFS e attuale sistema web

[56] *What is IPFS?*.

# Capitolo 3

# Descrizione stage

*In questo capitolo verrà descritto e presentato quanto fatto per la realizzazione del progetto richiesto dall'azienda per questo stage*

## 3.1 Requisiti e obiettivi

Per comprendere lo studio e le scelte fatte è opportuno delineare quelli che sono i requisiti e gli obiettivi concordati. Il piano di lavoro redatto dall'azienda riporta un primo elenco dei requisiti, suddivisi tra obbligatori, desiderabili e opzionali.

**Requisiti obbligatori:**

* implementazione in un ledger EVM per l'emissione di credenziali verificabili governate da uno smart contract (in inglese **issuer**)
* realizzazione di una soluzione basata su smart contract per la **revoca** di credenziali verificabili (in inglese **revoker**).
* realizzazione di una soluzione basata su smart contract per la validazione di credenziali verificabili (in inglese **verifier**).

Nel testo i vari smart contract potrebbero essere utilizzati sia con il loro nome italiano che con quello inglese, poichè sono intercambiabili.

**Requisiti desiderabili:**

* integrazione degli smart contract in uno use case aziendale.

**Requisiti opzionali:**

* sviluppo documentazione architetturale, flussi dati e flussi di controllo per gli smart contract implementati e l'integrazione di questi nello stack aziendale.

## 3.2 Analisi del problema e soluzione

In questa sezione verranno riportati gli studi e le scelte che hanno delineato lo sviluppo della soluzione.

### 3.2.1 Soluzione proposta

La soluzione proposta, che viene riportata, fa riferimento principalmente alla parte di progetto relativa agli smart contract.
Analizzando il problema nel suo complesso emerge come l'uso della blockchain permetta di risolvere molto facilmente alcune necessità, mentre in altre sembra introdurre diversi problemi difficili da risolvere. Lo studio quindi è stato affrontato suddividendolo nei tre macro-requisiti da soddisfare:

1. emissione di credenziali verificabili;
2. verifica di credenziali verificabili;
3. revoca di credenziali verificabili.

#### Emissione: necessità di una soluzione ibrida

La procedura per creare credenziali verificabili non è banale, attualmente sono presenti diversi standard da seguire e a cui fare riferimento[1]. Senza scendere nei dettagli di questo procedimento, il problema è che ci si scontra con la necessità di generare e trattare dei dati sensibili, i quali per motivi di privacy non possono essere scritti pubblicamente on-chain (sia che questa sia pubblica o privata). Supponiamo di scrivere lo smart contract che si occupi ad esempio di rilasciare delle credenziali che rappresentano una carta d'identità. Per poterlo fare l'utente dovrebbe invocare un suo metodo passando tutti i dati anagrafici richiesti, che verrebbero registrati in modo permanente sulla blockchain e resi accessibili a chiunque. Questo tipo di soluzione non è quindi ammissibile.
Inoltre non è nemmeno possibile sfruttare una cifratura a chiave asimmetrica per cifrare le informazioni e renderle comprensibili solo ai diretti interessati, poichè lo smart contract non è in grado di celare la sua chiave privata (come riportato anche nella sezione 2.9, il codice di un contratto è pubblico).
Queste considerazioni rendono obbligatoria un'azione che deve essere compiuta fuori dalla blockchain. Per questo motivo l'approccio a cui si è pensato è quello di sfruttare gli NFTs.
Un NFT come riportato in sezione 2.10, dimostra la proprietà di un contenuto digitale di un indirizzo (utente) e in questo caso verrebbe utilizzato come access token. L'idea quindi è quella di sfruttare le garanzie dell'NFT per poter emettere la credenziale verificabile off-chain direttamente sul wallet dell'utente.
Una volta utilizzato, il token deve essere consumato per non poter essere nuovamente spendibile, quindi è necessario che l'utente fornisca l'approvazione affinchè il contratto issuer possa procedere all'operazione di burn[2], poichè solo il proprietario del token ha i permessi di trasferirlo (il processo di burn è semplicemente il trasferimento del token sull'indirizzo nullo `0x0000000000000000000000000000000000000000`).

#### Verifica: necessità di verificare l'identità

Il requisito di implementare una soluzione on-chain che permetta di effettuare la verifica delle credenziali può essere diviso in due parti:

1. gestione dei verificatori;

[1] *W3C*. URL: https://www.w3.org/TR/vc-data-model/.
[2] OpenZeppelin. *ERC721 _burn*. URL: https://docs.openzeppelin.com/contracts/4.x/api/token/erc721#ERC721-_burn-uint256-.

2. gestione dei record di verifica.

**Gestione dei verificatori:** Il verificatore è la figura che ha il compito di registrare una verifica on-chain attraverso un verification record. Un verificatore, ricoprendo un ruolo abbastanza rilevante, si è scelto di renderlo inseribile solo dal proprietario del contratto, che dovrà registrare l'indirizzo del soggetto e una lista di dati associati al verificatore. Su questa scelta è possibile fare alcune riflessioni.
La blockchain, per costruzione, permette a chiunque di interfacciarsi con la rete in modo anonimo, nascondendosi dietro lo pseudonimo dell'indirizzo (ad esempio `0xda99451f8cf10f842c9c6b758712f856480adaa3`). Questo rende impossibile capire chi si cela effettivamente dietro l'indirizzo di un verificatore. È evidente che in questo caso è presente un forte punto di centralità e fiducia, chi interagisce con il contratto dovrebbe fidarsi di quanto dichiarato dal proprietario del contratto, situazione che si trova in contrasto con la soluzione "user-centered" che il modello SSI propone.
La soluzione scelta è quella di sfruttare le proprietà dei DID e l'inserimento di una **proof** nel record di verifica. Questo dato permette a chiunque di verificare personalmente che quel verificatore è esattamente chi dice di essere, che nella pratica si traduce nel dimostrare che quell'indirizzo possiede effettivamente le chiavi associate al DID che lo identifica. La proof quindi non è altro che un messaggio cifrato dei dati contenuti nel record di verifica, cifrati con la chiave privata del DID posseduta dal verificatore. In questo modo chiunque può recupare il DID presente on-chain, eseguire il processo di risoluzione e ottenere il DID document contenente la sua chiave pubblica. A questo punto, sfruttando la proprietà delle chiavi asimmetriche, sarà sufficiente decifrare la proof con la chiave appena trovata. Se quanto ottenuto è un testo in chiaro con i dati concordi a quanto registrato on-chain nel verification record, il verificatore è autentico.

**Gestione dei verification record:** Un verification record è la prova registrata on-chain che uno specifico soggetto ha subito un processo di verifica con esito positivo. Questo è un caso in cui la blockchain è di grande aiuto, poichè funge da registro decentralizzato delle verifiche. Questo offre innumerevoli vantaggi: i dati sono accessibili a chiunque, questo significa che a un soggetto sarà sufficiente esibire il proprio indirizzo per permettere a qualsiasi servizio di verificare la validità della sua credenziale. I dati inoltre godono di tutte le proprità della blockchain tra cui immutabilità, trasparenza e non ripudiabilità.

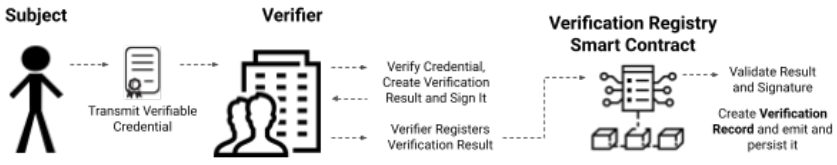



**Figura 3.1:** Schema del processo di verifica descritto

### Revoca

Eseguire la revoca di un verification record, a questo punto è abbastanza semplice, poichè anche cambiare lo stato di un suo flag è sufficiente. È un'operazione banale ma che grazie alla tecnologia blockchain offre diverse garanzie. Quando viene invocato il metodo per eseguire la revoca, dopo che il flag "`revoked`" viene settato a *true*, viene

lanciato un evento `VerificationRevoked(uuid)` il quale testimonia che il record con quel particolare `uuid` è stato revocato dal suo verificatore. Quest'informazione non può essere in alcun modo contraffatta, anzi dà la sicurezza che ad averlo revocato è stato proprio l'indirizzo del suo verificatore.

## 3.3 Implementazione

In questa sezione verranno riportate alcune scelte implementative fatte e discusso il caso d'uso utilizzato come studio del problema, in modo da poter esporre l'interazione tra le entità e chiarire ulteriormente la spiegazione teorica esposta nella sezione 3.2 (un'analisi formale dei principali casi d'uso affrontati è stata riportata in Appendice A).

### 3.3.1 Identificazione di un Verifier on-chain

Per registrare un verificatore on-chain il proprietario del contratto deve passare al metodo l'indirizzo del soggetto e un oggetto contenente i dettagli del verificatore.
I dettagli comprendono i seguenti campi:

* **name**: un nome alfanumerico (ad esempio *"Comune Padova"*);
* **DID**: il DID associato al verificatore (ad esempio *"did:example:12345"*);
* **url**: l'indirizzo riferito a una pagina web di presentazione (ad esempio *"https://padovanet.it/comune"*);
* **signer**: un indirizzo on-chain che dovrà essere utilizzato per firmare i dati al momento dell'inserimento di un verification record (ad esempio "`0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8`").

Su quest'ultimo campo è opportuno precisare quindi che l'indirizzo di un verificatore non deve obbligatoriamente coincidere con quello del firmatario dei dati. Questa scelta è stata fatta per permettere a più verificatori di registrare record che ad esempio provengono da un'autorità diversa (e quindi relativo a un altro indirizzo on-chain).

### 3.3.2 Struttura di un Verification Record

Come riportato in sezione 3.2, i verification record testimoniamo che un utente (indirizzo) ha subito un processo di verifica off-chain con esito positivo. Per questo motivo la verifica per essere registrata deve superare diversi controlli. Innanzitutto per creare un verification record l'utente verificatore deve passare allo smart contract un **verification result** e una **typed signature**. Un verification result è composto da:

* **soggetto:** indirizzo associato all'utente che si vuole verificare;
* **data di scadenza:** data dopo la quale il record non è più considerabile valido;
* **JSON result:** risultato in formato JSON (convertito opportunamente in stringa) che riporta l'esito della verifica off-chain;
* **type index:** è un identificativo del tipo di record. È stato inserito per poter estendere e personalizzare in futuro le regole di verifica (ad esempio se il tipo indicasse il verification record di una patente, il processo di verifica potrebbe includere la condizione di possedere un record valido per una credenziale d'identità);

* **proof:** una stringa che rappresenta l'intero messaggio dati criptato con la chiave che ha generato il did del verificatore.

Quest'ultimo campo permette appunto di risolvere il problema della proprietà del DID associato a un determinato verificatore. La soluzione è stata spiegata nella sezione 3.2.1 e non richiede ulteriori approfondimenti.
Il secondo parametro richiesto, la *typed signature*, si riferisce a un particolare tipo di firma la quale attesta che i dati forniti nel verification result provengano da uno specifico indirizzo, in questo caso assicurano che i dati siano firmati dall'indirizzo registrato come *signer* al momento dell'inserimento del verificatore.
Considerando i campi appena descritti, il risultante verification record scritto in blockchain avrà quindi i seguenti attributi:

* **Universally Unique Identifier (UUID):** codice univoco generato dal contratto sulla base dei seguenti valori: indirizzo del verificatore, indirizzo del soggetto, unix timestamp[3] della data di inserimento e scadenza, proof, type index e numero di verification record attualmente presenti.
* **indirizzo del verificatore:** indirizzo on-chain del verificatore che ha inserito il record;
* **indirizzo del soggetto:** indirizzo on-chain del soggetto della verifica;
* **data di inserimento:** data di inserimento del record in formato unix timestamp;
* **data di scadenza:** data di scadenza della verifica in formato unix timestamp;
* **flag record revocato:** valore che indica se quel record è stato revocato;
* **proof:** messaggio cifrato ricevuto dal verification result;
* **off-chain result:** JSON che rappresenta l'esito della verifica off-chain;
* **type index:** identificativo alfanumerico ricevuto dal verification result;

### 3.3.3 Metadata di un NFT access token

Come descritto nella sezione 2.10, solitamente gli NFT sono accompagnati da dei metadata. Anche in questo progetto si è optato per utilizzarli con l'idea di salvare al loro interno eventuali dati necessari in futuro. In questo momento la soluzione proposta costruisce dei metadata con i seguenti attributi:

* **name**: nome identificativo del tipo di NFT access token;
* **description**: descrizione testuale dello scopo relativo al token;
* **image:** indirizzo IPFS dell'immagine associata da mostrare sulla DApp;
* **details:** stringa codificata in base64 di un JSON contenente eventuali altri campi necessari.

Ad esempio, uno dei token di test creati per il caso d'uso riportato nella sottosezione 3.3.4 contiene i seguenti dati:

[3] *Unix Timestamp.* URL: https://www.unixtimestamp.com/.

```
{
    "name": "Diploma Access Token",
    "description": "Student can exchange this token to obtain diploma",
    "image": "ipfs://QmNkus5BdawNLFvHrE5wc4dW5wpvDjCxESj3oVKWBdwqbZ",
    "details": "ZGlkOnN0dWRlbnQ6MTIzNDU2O2RpZDpjb3JzbzoxMjM0NTY"
}
```

Decodificando `ZGlkOnN0dWRlbnQ6MTIzNDU2O2RpZDpjb3JzbzoxMjM0NTY` si ottiene la stringa: `did:student:123456;did:corso:123456`.

### 3.3.4 Caso d'uso: richiesta diploma di laurea

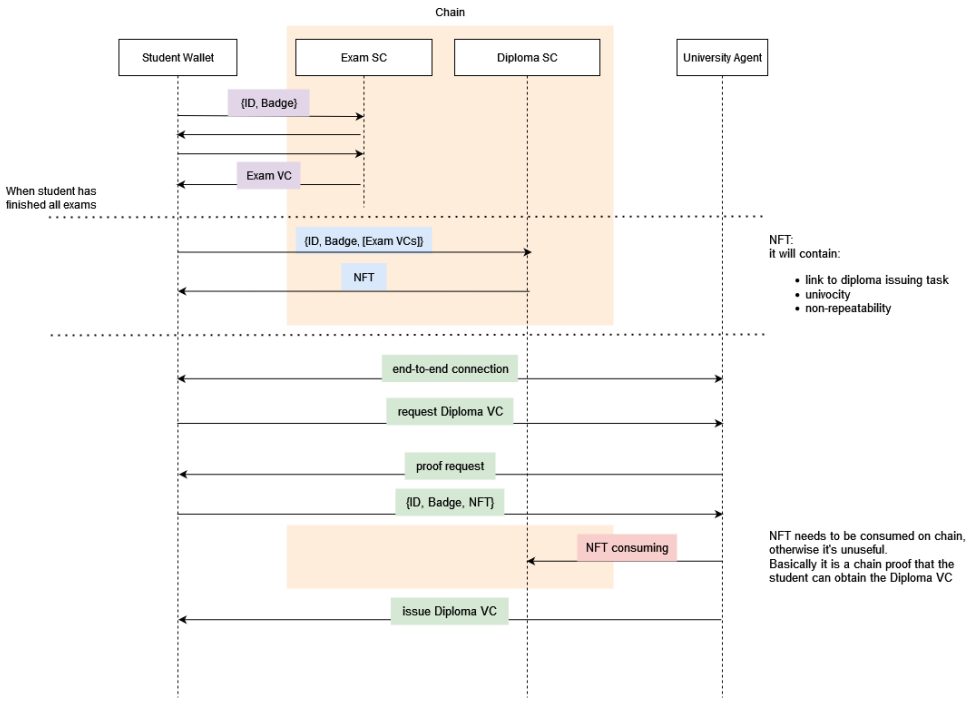



**Figura 3.2:** Schema caso d'uso richiesta diploma

Il caso d'uso riportato è stato utilizzato come esempio di studio del problema e delle richieste da soddisfare. Il diagramma a Figura 3.2 può essere diviso in tre parti sulla base delle due linee tratteggiate.
La prima parte rappresenta uno scambio dati tra il wallet dello studente e un fittizio smart contract, che si occupa dell'emissione delle credenziali relative agli esami superati. Questa prima parte è stata inserita per favorire la comprensione del contesto e non è stata affrontata direttamente (anche se il suo funzionamento è molto simile a quanto avviene per l'emissione del diploma).
La seconda parte descrive il rilascio dell'NFT access token allo studente, che come si può vedere, richiede una serie di verification record validi per le credenziali verificabili richieste. Si può notare inoltre che anche in questa fare è nuovamente l'utente ad innescare il processo di richiesta della credenziale.
A questo punto lo studente possiede quanto necessario per ottenere la credenziale

verificabile del diploma. Viene generata una connessione end-to-end tra il wallet dello studente e l'agent dell'università: il primo effettua la richiesta del diploma mentre il secondo risponderà con la richiesta di una prova di autenticazione. Lo studente a questo punto può dimostrare il possesso del token NFT semplicemente comunicando il suo indirizzo on-chain e eseguendo il processo di *approve*[4], per permettere all'università di consumarlo e rilasciare la credenziale.
I parametri `ID, Badge, Exam VCs, NFT` sono stati inseriti anch'essi per favorire la comprensione del caso d'uso e permettere di mostrare un esempio di credenziali che è necessario verificare.
Come anticipato questo caso d'uso è stato studiato e preso come riferimento per la progettazione, ma per motivi di tempo e per permettere di gestire un numero maggiore di casi d'uso, per il prodotto finale è sufficiente possedere un unico verification record valido per funzionare, e non le diverse credenziali riportate. Questo è un esempio in cui il campo **type index** (spiegato nella sottosezione 3.3.2) può esser utilizzato per modificare e adattare la verifica al caso d'uso specifico. Il prodotto risulta quindi essere un ottimo punto di partenza per soddisfare diverse situazioni.

[4] OpenZeppelin. *ERC721 approve*. URL: https://docs.openzeppelin.com/contracts/4.x/api/token/erc721#IERC721-approve-address-uint256-.

# Capitolo 4

# Progettazione e Realizzazione

*In questo capitolo verrà riportata e discussa la progettazione del prodotto ed eventuali scelte architetturali che hanno guidato la realizzazione fino all'ottenimento del prodotto richiesto.*

## 4.1 Analisi architetturale

Il prodotto realizzato è una DApp che ha lo scopo di testare e mostrare le principali funzioni del modello SSI, attraverso l'integrazione degli **smart contract** da me sviluppati e del SDK di **walt.id**[1] scritto dal mio collega. È composto da un frontend che permette all'utente di interagire con l'applicazione e che integra la connessione al browser wallet Metamask (sezione 4.1.1). L'interfaccia sfrutta delle chiamate Hypertext Transfer Protocol (HTTP) a delle API esposte da un backend e da un subgraph proprietari, le quali permettono di espandere le funzionalità standard di interrogazione della blockchain e degli smart contract rendendo l'esperienza di utilizzo molto più dinamica.
Il backend fornisce la possibilità di svolgere, tramite l'utilizzo di un'opportuna libreria, operazioni di cifratura con chiavi asimmetriche. Il subgraph invece consente di effettuare query sulla blockchain in modo più efficiente ed efficace.
Di seguito vengono riportate le scelte tecnologiche fatte e le relative motivazioni, suddivise per le quattro componenti principali: frontend, backend, subgraph e smart contract.

### 4.1.1 Frontend

Il front-end è l'interfaccia con cui l'utente andrà ad interagire.

#### ReactJs e Typescript

Si è scelto di utilizzare ReactJs come libreria javascript per poter gestire al meglio le chiamate asincrone alla blockchain, e fornire un'esperienza più dinamica all'utilizzatore. Questa libreria molto conosciuta e che non ha bisogno di presentazioni, offre diversi hook che permettono una gestione molto semplice dei valori da aggiornare sulle viste e delle strutture dati.

[1] *walt.id*.

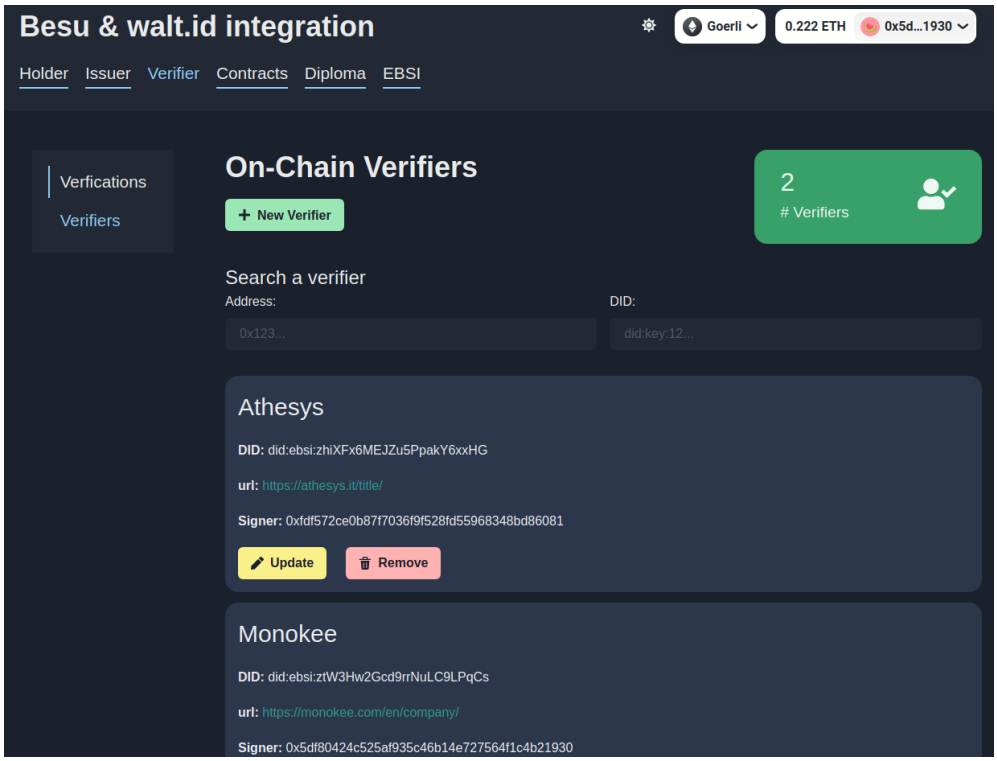



**Figura 4.1:** Screenshot pagina verificatori della DApp

Alla libreria è stato affiancato Typescript in modo da poter prestare più attenzione alla gestione dei tipi. Ulteriori informazioni su *reactjs.org*[2].

### ChakraJS: React components library

Per velocizzare lo sviluppo dell'interfaccia utente è stata utilizzata una libreria di componenti React. Quest'ultima è stata particolarmente utile per avere un codice più omogeneo sulle componenti standard come bottoni, spinner e popup.
La scelta di questa libreria è dovuta principalmente alla sua semplicità di utilizzo. Ulteriori informazioni su *chakra-ui.com*[3].

### Wagmi: React Hooks for Ethereum

Wagmi è una libreria di React hooks per l'interazione con le blockchain Ethereum (EVM). Gli hook messi a disposizione permettono di gestire tutte le principali operazioni di una DApp, tra le tante ad esempio, la connessione al browser wallet, gestione dell'indirizzo utente e delle chiamate in lettura e scrittura a uno smart contract.
La scelta di questa libreria è dovuta principalmente al fatto che gestire molte di queste operazioni da zero non è banale, ogni chiamata a un contratto dovrebbe essere configurata e gestita con i parametri necessari, e inoltre essendo ogni chiamata asincrona, sarebbe necessario sviluppare un meccanismo per gestire al meglio le Promise[4]. Ulteriori

[2] *ReactJS.* URL: https://reactjs.org/.
[3] *Chakra UI.* URL: https://chakra-ui.com/.
[4] *Promise.* URL: https://www.w3schools.com/js/js_promise.asp.

informazioni su *wagmi.sh*[5].

#### Pinata: NFT media management service

Pinata è un **NFT media management service** che fornisce agli utenti la possibilità di caricare, gestire e condividere file su IPFS. La grande comodità offerta da questo servizio è che dà la possibilità di caricare contenuti sul file system distribuito senza dover configurare e gestire un proprio nodo IPFS, per questo motivo è un servizio che si presenta come una soluzione molto valida per caricare e conservare i metadata degli NFT access token. Ulteriori informazioni su *pinata.cloud*[6].

#### Metamask: crypto wallet and gateway to blockchain apps

Metamask è un wallet di criptovalute usato per interagire con le blockchain Ethereum. Permette di accedervi direttamente attraverso un'estensione del browser o tramite applicazione mobile, e di interagire con le DApp. Nella pratica le principali funzioni di Metamask sono: conservare e gestire le chiavi dell'account di un utente, effettuare transazioni di criptovalute e token, e garantire una connessione sicura alle DApp. Ulteriori informazioni su *metamask.io*[7].

### 4.1.2 Backend

L'utilizzo di un semplice backend NodeJs si è reso necessario per eseguire le operazioni di cifratura riportate in sezione 3.2.1. Questo tipo di operazioni non sono eseguibili a front-end poichè le diverse librerie a supporto di queste funzioni operano solo a questo livello. Per il progetto si è optato per la libreria **JOSE**[8] (JSON Object Signing and Encryption), una delle poche che tra i diversi standard di cifratura gestisce anche curve ellittiche **secp256k1**, usate in diverse blockchain, tra cui Bitcoin e Ethereum, per la crittografia della coppia di chiavi.

### 4.1.3 Subgraph

Una DApp solitamente è composta almeno da due componenti: un front-end su cui viene eseguita l'applicazione e uno o più contratti al quale fa riferimento. Quando avviene una transazione, vengono emessi degli eventi, tramite i quali l'applicazione può osservare gli aggiornamenti effettuati e rifletterli poi sull'interfaccia utente. Le DApp però hanno un numero molto limitato di query da eseguire per recuperare dati (è possibile utilizzare i metodi del contratto o del nodo Ethereum).
Questo tipo di situazione non permette quindi di creare applicazioni con una completa esperienza utente, come avviene invece per un attuale sito web. Diverse DApp hanno adottato soluzioni personalizzate per il proprio caso d'uso, tra cui sistemi di tracciamento degli eventi, delle transazioni e di altri dati, con lo scopo di salvarli in un tradizionale database centralizzato. Questa stategia è però in contrasto con la filosofia del mondo web3[9] che si basa sulla minimizzazione della fiducia.

[5] *Wagmi.* URL: https://wagmi.sh/.
[6] *Pinata.* URL: https://pinata.cloud/.
[7] *MetaMask.* URL: https://metamask.io/.
[8] *JSON Web Almost Everything.* URL: https://www.npmjs.com/package/jose.
[9] *What is web3 and.* URL: https://ethereum.org/it/web3/.

### TheGraph Protocol

La soluzione più in linea con questo pensiero e attualmente molto utilizzata è quella sviluppata dal team di TheGraph, che ha costruito un protocollo decentralizzato che sfrutta dei nodi per processare gli eventi Ethereum e salvare i dati in modo indicizzato cosicchè possano essere interrogati attraverso un'interfaccia API.
Nel progetto è stato sviluppato un subgraph che riporta i dettagli degli smart contract, gli eventi a cui prestare attenzione e come mappare i dati presenti in quest'ultimi sulle entità che TheGraph dovrà indicizzare. Ulteriori informazioni su *thegraph.com*[10].

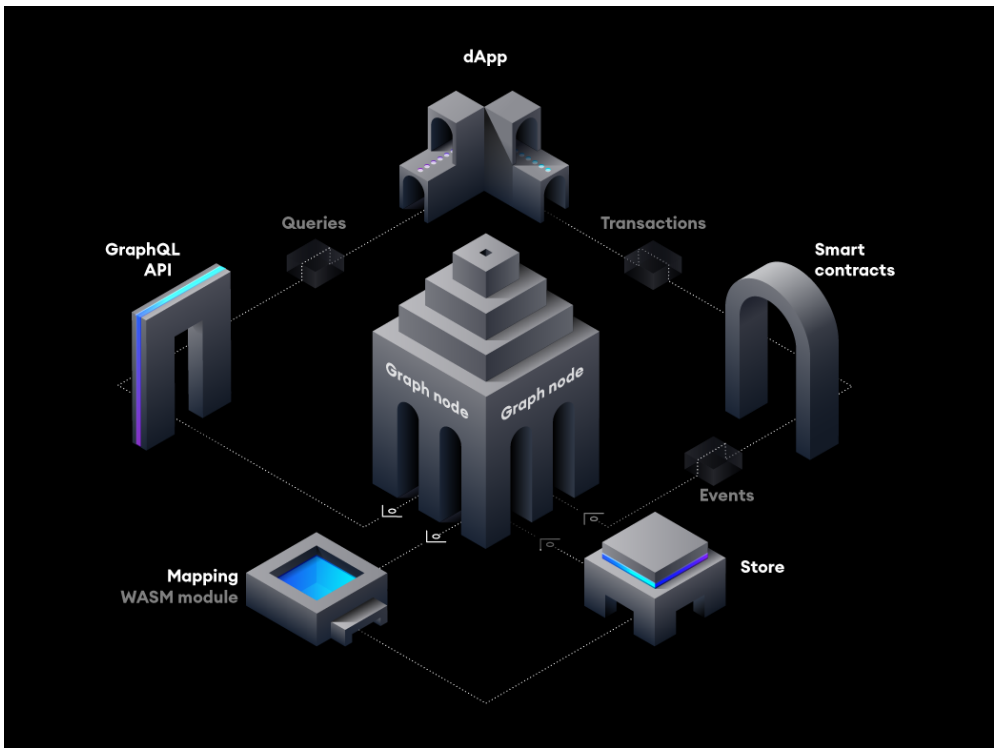



**Figura 4.2:** Schema funzionamento TheGraph

## 4.1.4 OpenZeppelin contracts

Per lo sviluppo degli smart contract è stato deciso di adottare le interfacce e i contratti offerti da OpenZeppelin. In questa sezione verranno riportati quelli a cui si fa riferimento e le proprietà che offrono.

### ERC721

Questo standard fa riferimento a tutte le funzionalità legate allo standard ERC721[11], ovvero quello relativo ai Non-Fungible Token. Insieme allo standard "puro", OpenZeppelin fornisce anche altre interfacce per gestire caratteristiche associabili a questi token. Per il progetto sono stati utilizzati i contratti:

[10] *The Graph*. URL: https://thegraph.com/.
[11] *EIP-721*. URL: https://eips.ethereum.org/EIPS/eip-721.

* **ERC721URIStorage:**[12] permette di definire NFT con metadati recuperabili tramite un URI, una sequenza alfanumerica che identifica una risorsa da un'altra. Nel progetto è stato utilizzato per permettere di registrare l'indirizzo IPFS associato. Ad esempio:
`ipfs://QmW7aofc5AHLHQQ4V3kwojYPW316j7ybgGgx3Var2DJTvo`
* **ERC721Burnable:**[13] serve a definire NFT che possono essere bruciati e quindi distrutti, caratteristica obbligatoria degli NFT access token utilizzati nel progetto.

### Pausable

Modulo[14] che permette ai contratti figli di implementare dei meccanismi di "emergency stop" che possono essere innescati da un account autorizzato. Questo modulo è usato attraverso l'ereditarietà. Fornisce dei metodi per vincolare le funzioni scelte sulla base dello stato in cui si trova il contratto.[15]

### Ownable

Fornisce al contratto un basico meccanismo di controllo degli accessi, in questo caso ci sarà un account **owner** il quale avrà un accesso esclusivo a specifiche funzioni. Per default questo modello imposta l'owner sull'account che fa il deploy del contratto, questo privilegio potrà poi essere modificato con la funzione `transferOwnership`.
Questo modulo si applica attraverso l'ereditarietà. Rende disponibile il modifier `onlyOwner`, che può essere applicato alle funzioni per restringere l'accesso solo all'owner.

### AccessControl

Rispetto al modulo Ownable, questo permette di implementare un meccanismo di controllo degli accessi basato su ruoli[16]. I ruoli sono definiti tramite un identificativo in `bytes32` (tipo primitivo di Solidity). Solitamente sono dichiarati all'interno dei contratti con delle costanti, nel prodotto sono state utilizzati i seguenti:

```
bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");
bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
```

I ruoli possono essere concessi o revocati dinamicamente tramite le due funzioni messe a disposizione dal modulo: `grantRole` e `revokeRole`. Queste funzioni hanno il vincolo di poter essere chiamate solo dal ruolo `DEFAULT_ADMIN_ROLE` che solitamente viene assegnato all'account che fa il deploy del contratto.

### ECDSA

Permette di eseguire operazioni Elliptic Curve Digital Signature Algorithm (ECDSA)[17]. Questa libreria fornisce delle funzioni che nel progetto sono state utilizzate per verificare

[12] OpenZeppelin. *ERC721URIStorage*. URL: https://docs.openzeppelin.com/contracts/4.x/api/token/erc721#ERC721URIStorage.
[13] OpenZeppelin. *ERC721Burnable*. URL: https://docs.openzeppelin.com/contracts/4.x/api/token/erc721#ERC721Burnable.
[14] OpenZeppelin. *Ownable*. URL: https://docs.openzeppelin.com/contracts/4.x/api/access#Ownable.
[15] OpenZeppelin. *Pausable*. URL: https://docs.openzeppelin.com/contracts/4.x/api/security#Pausable.
[16] OpenZeppelin. *AccessControl*. URL: https://docs.openzeppelin.com/contracts/4.x/api/access#AccessControl.
[17] OpenZeppelin. *ECDSA*. URL: https://docs.openzeppelin.com/contracts/2.x/api/cryptography#ECDSA.

che un messaggio sia stato firmato dal proprietario della chiave privata di un dato indirizzo.
Questo contratto insieme allo standard EIP712 permette di generare la typed signature descritta nella sottosezione 3.3.2.

#### EIP712

É uno standard usato per eseguire operazioni di hashing e firma di date strutture dati. Utilizza una combinazione di `abi.encode` e `keccak256`.
Questo contratto implementa lo EIP712[18] domain separator che viene utilizzato principalmente come step finale per ottenere il message digest che verrà poi firmato con l'algoritmo ECDSA (la funzione messa a disposizione è `_hashTypedDataV4`).

## 4.2 Smart Contract

Poichè gli smart contract rappresentano la parte principale del lavoro svolto, si è ritenuto opportuno redigere una sezione dedicata. Di seguito verranno riportate le best practice e gli standard di sviluppo sui quali si è condotto uno studio e che hanno influenzato la progettazione degli smart contract.

### 4.2.1 Provvedimenti adottati

Poichè lo scopo finale degli smart contract sviluppati è quello di dimostrare la fattibilità dei requisiti richiesti attraverso un esempio funzionante, non hanno nessuna pretesa di essere completi e pronti ad un ambiente di produzione.
Le precauzioni studiate e riportate nelle sezioni successive sono state prese in considerazione, ma solo in parte adottate. Vista la complessità e l'attenzione richiesta per la creazione di un proxy pattern (sottosezione 4.2.4) si è optato per non svilupparlo e dare invece spazio allo sviluppo del PoC.
Il compromesso scelto è quello di aver inserito comunque dei meccanismi di sicurezza di tipo circuit breaker (sezione 4.2.3) in modo da poter fermare il ciclo di rilascio delle credenziali nel caso di presenza di errori. Fanno riferimento a quanto appena descritto i casi d'uso UC18 (sezione A.2.4) e UC19 (sezione A.2.4) riportati in Appendice A.
Inoltre durante tutta la fase di sviluppo del codice si sono seguite le best practice riportate nella sottosezione 4.2.2.
Ulteriori informazioni sulla parte relativa alla codifica sono state inserite nella sezione 4.3.

### 4.2.2 Raccomandazioni generali sullo sviluppo in ambito blockchain

Una delle prime accortezze mentre si sviluppano smart contract è quella di prestare molta attenzione alle chiamate esterne (in inglese **external calls**), in particolare effettuare chiamate a contratti untrusted (non fidati) può introdurre diversi rischi o errori. Una chiamata potrebbe eseguire codice malevolo in quel contratto o in un qualsiasi contratto ad esso collegato. Quindi ogni chiamata va trattata come un possibile rischio per la sicurezza. Ci sono tuttavia delle raccomandazioni per minimizzare il pericolo nel caso in cui non sia proprio possibile rimuovere le chiamate esterne.

[18] OpenZeppelin. *EIP712*. URL: https://docs.openzeppelin.com/contracts/4.x/api/utils#EIP712.

**Tracciamento dei contratti untrusted** Nel progetto è stato creato un registro contenente le informazioni relative ai contratti che si andrà ad utilizzare, in cui vengono indicate le principali proprietà e lo stato di fiducia.
Inoltre, quando si ha a che fare con contratti esterni, è buona norma indicare sempre con nomi di variabili, metodi o interfacce se si sta interagendo con qualcosa di potenzialmente non sicuro.

**Evitare cambiamenti di stato dopo chiamate esterne** Assumere sempre che si possa aver eseguito codice malevolo, quindi evitare cambiamenti di stato dopo la chiamata. Questo tipo di pattern è noto anche come **checks-effects-interactions pattern**[19].

**Non usare transfer() o send()** `.transfer()` e `.send()` inoltrano esattamente 2,300 gas al ricevente. Lo scopo è quello di prevenire reentrancy vulnerabilities[20], ma dopo l'Istanbul hard fork[21] del 2019, con Ethereum Improvement Proposals (EIP) 1884[22], c'è stato un incremento del costo delle SLOAD operation[23] che causano la chiamata della fallback function che costa più di 2300 gas. Quindi al loro posto è raccomandato usare `.call()`.
Questa funzione non fa nulla per mitigare il problema dei reentrancy attacks, quindi è necessario comunque prendere delle precauzioni (usando ad esempio il pattern citato sopra).

**Evitare di combinare trasferimenti multipli in una singola transazione** Combinare più trasferimenti di criptovalute in una singola transazione può portare facilmente all'interruzione di una funzione e al conseguente revert delle operazioni. Per evitare questo problema è sempre consigliato separare i trasferimenti.

**Non usare deletegatecall() con codice untrusted** La funzione `delegatecall()` è usata per chiamare funzioni di altri smart contract come se appartenessero al contratto chiamante. Questa chiamata può essere insicura nel caso in cui venga eseguita verso un contratto non fidato. Ad esempio se si riusciesse a far invocare al chiamante una funzione che contiene `selfdestruct(0)` andrebbe a distruggere il contratto con la conseguente perdita di tutto il suo bilancio.

[19] *Checks Effects Interactions pattern.* URL: https://docs.soliditylang.org/en/develop/security-considerations.html?highlight=check%5C%20effects#use-the-checks-effects-interactions-pattern.
[20] *Reentrancy Attack.* URL: https://consensys.github.io/smart-contract-best-practices/attacks/reentrancy/.
[21] *Ethereum Istanbul Hard Fork.* URL: https://ethereum.org/en/history/#istanbul.
[22] *EIP-1884.* URL: https://eips.ethereum.org/EIPS/eip-1884.
[23] *Assembly.* URL: https://docs.soliditylang.org/en/v0.8.16/assembly.html.

```
contract Destructor
{
    function doWork() external
    {
        selfdestruct(0);
    }
}

contract Worker
{
    function doWork(address _internalWorker) public
    {
        // unsafe
        _internalWorker.delegatecall(bytes4(keccak256("doWork()")));
    }
}
```

**Listing 4.1:** Esempio di delegatecall malevola

**Unreliable Partecipants** Quando si sviluppano smart contract, non bisogna mai fare affidamento su una specifica azione dell'utente, quest'ultimo potrebbe non volerla compiere. Quindi se ci dovessere essere il rischio di queste situazioni, le soluzioni più utilizzate sono:

* provvedere a un modo di aggirare l'azione richiesta all'utente che non partecipa;
* considerare l'aggiunta di incentivi economici per i partecipanti in tutte le situazioni in cui si potrebbero trovare (ad esempio se si richiede all'utente di compiere un certo numero di operazioni, sarebbe ottimale incentivarlo a compierle tutte e a non fermarsi).

### 4.2.3 Precauzioni in caso di errori

La tematica della sicurezza in ambito blockchain è molto importante, poichè un fallimento può comportare costi molto elevati. A meno di una verifica formale e rigorosa è molto difficile sapere in anticipo se il nostro codice è sicuro, ma è comunque possibile fare in modo che un contratto fallisca "graziosamente" e con il minimo danno nel caso di errori.
In questa sezione sono riportate delle tecniche che sono state valutate e prese in considerazione per prepare il contratto a un possibile fallimento.

#### Upgradeability

Come citato nella sezione 2.6, la blockchain è immutabile e lo sono anche i dati e i contratti che vengono caricati. Può succedere comunque di avere necessità di mettere mano al codice nel caso in cui si trovi un errore o si voglia inserire un miglioramento. Esistono tuttavia due strade principali per creare un meccanismo di aggiornamento. La più semplice è quella di avere un contratto contenente esclusivamente l'indirizzo dell'ultima versione del contratto che attua la logica, mentre il secondo è quello di usare la funzione `delegatecall()`[24] in modo tale che sia possibile inoltrare le chiamate al

[24] *delegatecall*. URL: https://docs.soliditylang.org/en/latest/introduction-to-smart-contracts.html%5C#delegatecall-callcode-and-libraries.

contratto più aggiornato[25]. Quest'ultimo è il più utilizzato ed è noto con il nome di **proxy upgrade pattern**[26].
Per entrambe le soluzioni, bisogna comunque cercare di mantenere una certa modularità e separazione tra le componenti, per evitare di avere funzionalità rotte o dati orfani. In particolare, per prevenire questi problemi è consigliabile separare la complessità logica dai dati, in modo tale che non sia necessario ricreare tutti i dati nel caso di cambio di funzionalità.
L'upload di un nuovo contratto è spesso un'attività che non è immediata, quindi è consigliato avere un modo per poter agire il più velocemente possibile in caso di attacco o bug, ad esempio con un **emergency stop o circuit-breaker**[27]. Indipendentemente da questo, un contratto che non ha modo di essere aggiornato (anche in modo limitato) diventerà facilmente inutilizzabile nel momento in cui verranno scovati bug.

**Circuit Breakers** Gli Circuit Breakers fermano l'esecuzione nel caso in cui certe condizioni siano verificate, solitamente quando sono scoperti errori. Ad esempio, un contratto che ha a che fare con i fondi degli utenti potrebbe sospendere la maggior parte delle azioni messe a dispozione e lasciare come unica possibilità attiva il prelievo (e non il deposito).

**Speed Bumps** Gli Speed Bumps rallentano lo svolgimento di determinate azioni, in modo che se si verificano azioni malevole, ci sia il tempo per recuperare. Per esempio, i modelli di Decentralized Autonomous Organization (DAO)[28] richiedono un periodo di tempo anche lungo tra l'approvazione della richiesta di scissione e la possibilità di farlo. The DAO[29], uno dei framework più utilizzati per la creazione di queste organizzazioni, ad esempio, richiede 27 giorni. Sfruttando questo periodo di tempo, è possibile mantenere i fondi nel contratto ed elaborare una strategia nel caso in cui si dovesse intervenire.

**Rate Limiting** Imporre, quando necessario, dei limiti o delle richieste di approvazione quando si stanno per eseguire modifiche sostanziali. Ad esempio, se un verificatore volesse revocare un record, potrebbe essere richiesto che questa operazione venga approvata anche da uno o più diversi verificatori.

### 4.2.4 Studio di una soluzione con proxy pattern

Per valutare la possibilità di creare degli smart contract aggiornabili usando il proxy pattern, è stato condotto uno studio relativo alle strategie e alle regole necessarie alla sua realizzazione.

[25] *Example of delegatecall*. URL: https://consensys.github.io/smart-contract-best-practices/development-recommendations/precautions/upgradeability/%5C#example-2-use-a-delegatecall-to-forward-data-and-calls.
[26] OpenZeppelin. *Proxies*. URL: https://docs.openzeppelin.com/upgrades-plugins/1.x/proxies.
[27] *Circuit Breaker*. URL: https://consensys.github.io/smart-contract-best-practices/development-recommendations/precautions/circuit-breakers/.
[28] *What is a DAO and how does it work?* URL: https://cointelegraph.com/decentralized-automated-organizations-daos-guide-for-beginners/what-is-decentralized-autonomous-organization-and-how-does-a-dao-work.
[29] *The DAO*. URL: https://github.com/blockchainsllc/DAO.

### Concetti alla base

Il pattern si sviluppa sulla base di tre accorgimenti:

* **usare upgradeable smart contract libraries:** se si progetta un pacchetto di contratti aggiornabile bisogna sempre tenere in mente che non riguarda solo i nostri contratti ma anche quelli che andiamo a importare. Un esempio possono essere proprio quelli di OpenZeppelin[30], ma a prescindere è molto importante ricordarsi di utilizzare una distribuzione che utilizzi degli inizializzatori al posto dei costruttori;
* **evitare valori di inizializzazione dei campi:** Solidity concede di definire dei valori di inizializzazione per i campi al momento della loro inizializzazione, questo approccio è molto simile a quello di inizializzare i valori all'interno del contratto e non funziona con smart contract aggiornabili, poichè se l'assegnazione non avviene all'interno di un inizializzatore, alcune istanze di contratti aggiornabili potrebbero non avere lo stesso valore (fanno eccezione le costanti, il compilatore non riserva uno slot di memoria per loro perchè ogni occorrenza è sostituita con la rispettiva espressione);
* **sempre inizializzare il contratto che implementa la logica:** un contratto non inizializzato potrebbe essere soggetto ad attacchi, per prevenire il problema è buona prassi invocare la funzione `_disableInitializers()` all'interno del costruttore.

### Come si crea una nuova istanza dal codice del contratto

Quando si va a creare una nuova istanza di un contratto dal codice Solidity, la creazione è gestita da Solidity stesso e non da eventuali librerie come ad esempio OpenZeppelin. Solitamente il pattern di creazione è quello di accettare semplicemente la nuova istanza del contratto come parametro e quindi fare l'injection dopo la sua creazione. Ecco un esempio:

```
pragma solidity ^0.6.0;

import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.
    sol";
import "@openzeppelin/contracts-upgradeable/token/ERC20/
    IERC20Upgradeable.sol";

contract MyContract is Initializable {
    IERC20Upgradeable public token;

    function initialize(IERC20Upgradeable _token) public initializer {
        token = _token;
    }
}
```

**Listing 4.2:** Esempio di injection

### Potenziali operazioni non sicure

Lavorando con contratti aggiornabili, si dovrà interagire sempre con le istanze di contratti e mai con il contratto che gestisce la logica. Comunque, nulla vieta a un

[30] OpenZeppelin. *Upgradeable Contracts*. URL: https://docs.openzeppelin.com/contracts/4.x/upgradeable.

attore malevolo di inviare transazioni direttamente al contratto che contiene la logica. Per questo motivo è bene sottolineare che:

* ogni modifica che viene apportata a logic contracts non deve avere effetto sui contratti che li invocano;
* lo storage dei contratti che contengono la logica non dovrebbe essere utilizzato;

Inoltre è opportuno ricordare nuovamente il pericolo di delegare una chiamata a un contratto non fidato, se una di queste dovesse riuscire a far lanciare un'operazione selfdestruct al logic contract, a quel punto il contratto verrebbe distrutto e tutte le istanze delegherebbero le chiamate ad un indirizzo che non esiste. Questo potrebbe rompere il funzionamento del sistema (ulteriori considerazioni sono già state fatte in sezione 4.2.2).

### Regole sulle modifiche ammesse

C'è una restrizione molto importante di cui tenere conto: **non** è possibile **cambiare l'ordine** con cui il contratto istanzia le variabili e nemmeno il **tipo**. Non tenere conto di questa restrizione potrebbe causare facilmente errori critici nella nostra applicazione. Il contratto aggiornato arriverebbe ad avere un insieme disorganizzato di dati all'interno del suo storage.
Quindi le operazioni che non sono concesse sono:

* cambiare il tipo di una variabile;
* cambiare l'ordine con il quale sono dichiarate;
* inserire una variabile prima di quelle già presenti;
* cancellare una variabile esistente;

Questo significa che:

* è possibile introdurre variabili solo dopo quelle già esistenti;
* rinominare una variabile è possibile, ma questa continuerà ad avere lo stesso valore di quella precedente (semanticamente rimarrà la stessa);
* cancellare una variabile non pulirà lo storage del contratto, sarà quindi ancora leggibile;
* non è possibile aggiungere nuove variabili ai contratti base se il figlio possiede una variabile che ne rappresenta l'istanza, perchè potrebbe succedere che una variabile vada a rimpiazzarne un'altra (quella del padre sostituirebbe quella del figlio).

```
// Scenario
contract Base {
    uint256 base1;
}

contract Child is Base {
    uint256 child;
}
```

```
// Se "Base" venisse modificato aggiungendo una variabile
contract Base {
    uint256 base1;
    uint256 base2;
}

// alla variabile "base2" verrebbe assegnato lo slot che aveva child
     nella versione precedente
```

**Listing 4.3:** Scenario in cui ci sono conflitti tra padre e figlio

### Proxy Upgrade Pattern

Il pattern consiste in un contratto che fungerà da wrapper o "proxy" con il quale l'utente interagirà direttamente e che provvederà a inoltrare le transazioni al secondo contratto che conterrà la logica. Il concetto chiave è che il logic contract potrà essere rimpiazzato mentre il proxy non cambierà mai.
Schema del funzionamento:

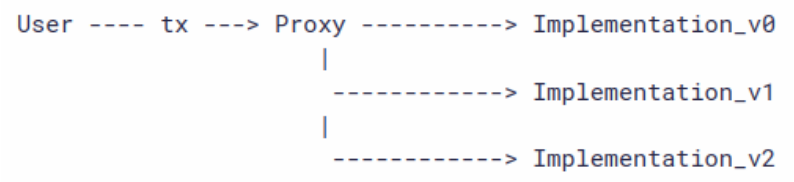



**Figura 4.3:** Schema proxy pattern

**Proxy Forwarding** Il principale problema affrontato da questo modello è la necessità di esporre l'intera interfaccia del logic contract senza richiedere un mapping uno a uno, che sarebbe difficile da mantenere, incline ad errori e lo renderebbe non aggiornabile. È necessario un meccanismo di inoltro dinamico. Una soluzione possibile sfrutta codice Solidity a basso livello:

```
assembly {
    let ptr := mload(0x40)

    // (1) copy incoming call data
    calldatacopy(ptr, 0, calldatasize)

    // (2) forward call to logic contract
    let result := delegatecall(gas, _impl, ptr, calldatasize, 0, 0)
    let size := returndatasize

    // (3) retrieve return data
    returndatacopy(ptr, 0, size)

    // (4) forward return data back to caller
    switch result
    case 0 { revert(ptr, size) }
    default { return(ptr, size) }
}
```

**Listing 4.4:** Codice usato per l'inotro dinamico

Questo codice può essere inserito nella funzione di fallback di un proxy e quest'ultimo inoltrerà ogni chiamata a una qualsiasi funzione con qualsiasi insieme di parametri al logic contract senza la necessità di sapere qualcosa relativamente all'interfaccia del contratto. Una cosa molto importante da notare è che il codice fa uso dell'opcode delegatecall dell'EVM, che esegue il codice nel contesto dello stato del chiamante. In altre parole, il contratto logico controlla lo stato del proxy e lo stato del contratto logico è privo di significato. Pertanto, il proxy non solo inoltra le transazioni da e verso il contratto logico, ma rappresenta anche lo stato della coppia.
In conclusione lo stato è nel proxy e la logica è nella particolare implementazione a cui il proxy punta.

**Unstructured Storage Proxies** Un altro problema che sorge con l'uso dei proxy è quello relativo a come vengono salvate le variabili in un proxy contract. Si può facilmente incorrere in collisioni sullo storage, ad esempio:

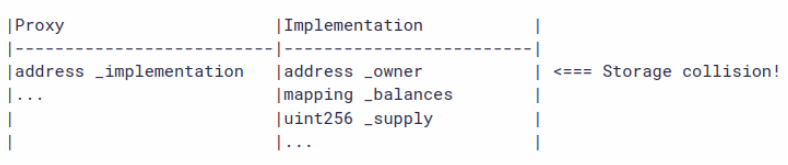



**Figura 4.4:** Esempio di storage collision

Sia `_implementation` che `_owner` sono composte da 32 bytes. Questo significa che quando il logic contract scrive su `_owner`, lo fa nello scope dello stato del proxy e in realtà sta scrivendo su `_implementation`.
Fortunatamente esistono delle soluzioni al problema, ad esempio OpenZeppelin propone di non salvare `_implementation` sul primo slot di storage del proxy, ma sceglie invece uno slot randomico. Questo slot è sufficientemente casuale, che la probabilità di andare a dichiarare una variabile nello stesso slot è trascurabile. Questo approccio viene usato per praticamente ogni variabile nel proxy. Ulteriori informazioni sono contenute nella proposta EIP-1967[31].
Questa soluzione prende il nome dal fatto che nessun contratto deve tenere conto della struttura dell'altro o adattarsi di conseguenza.

**Storage Collisions tra diverse versioni** L'approccio unstructured storage non salvaguardia dalla situazione riportata in Figura 4.5, sarà quindi compito dell'utente rispettare la gerarchia dello storage senza modificarla.

**Attenzione ai costruttori** In Solidity, il codice che si trova all'interno di un costruttore o di una dichiarazione di variabile globale non fa parte del bytecode di un contratto deployato. Quel tipo di codice viene eseguito solo una volta, al momento del deploy del contratto. Di conseguenza, quanto presente all'interno del costruttore di un contratto logico non sarà mai eseguito nel contesto dello stato del proxy (che detiene lo stato della coppia come spiegato in sezione 4.2.4). Per riassumere, i proxy sono completamente ignari dell'esistenza dei costruttori. È come se non esistessero.
Il problema è facilmente risolvibile. I contratti logici dovrebbero spostare il codice

[31] *EIP-1967*. URL: https://eips.ethereum.org/EIPS/eip-1967.

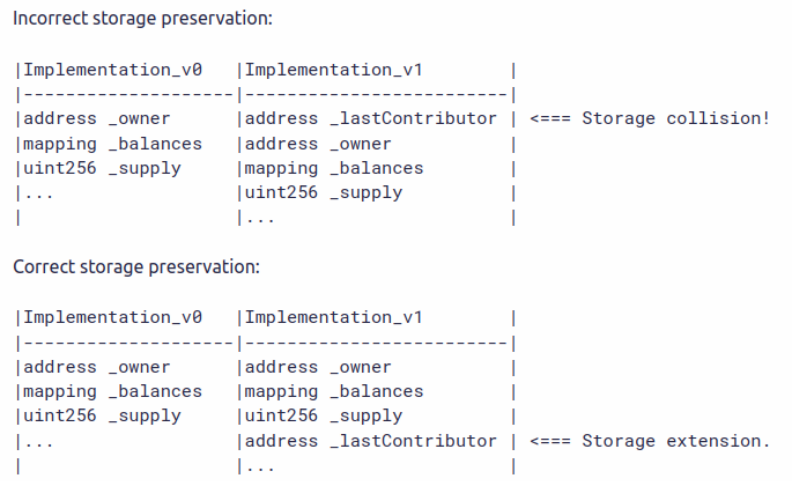



**Figura 4.5:** Esempio di storage collision e storage extension tra diverse versioni

all'interno del costruttore in una normale funzione "inizializzatrice" e fare in modo che questa funzione sia chiamata ogni volta che il proxy si collega a questo contratto logico. Occorre prestare particolare attenzione a questa funzione inizializzatrice, in modo che possa essere richiamata una sola volta, il che è una delle proprietà dei costruttori nella programmazione generale. Esempio di codice:

```
pragma solidity ^0.6.0;

import "@openzeppelin/contracts-upgradeable/proxy/utils/
    Initializable.sol";

contract MyContract is Initializable {
    function initialize(
        address arg1,
        uint256 arg2,
        bytes memory arg3
    ) public payable initializer {
        // "constructor" code...
    }
}
```

**Listing 4.5:** Esempio di funzione inizializzatrice

**Function Clashes: cosa succede con funzioni con la stessa firma** Potrebbe verificarsi la situazione in cui proxy e logic contract hanno una funzione con la stessa firma, in questo caso non è possibile capire quale funzione intenda chiamare l'utente. Ogni funzione che è parte di un Application Binary Interface (ABI)[32] pubblica è definita, a livello del bytecode, da un identificatore di 4 byte. Questo identificatore dipende dal nome e dall'arità della funzione e per via della dimensione finita a 4 byte appunto, c'è la possibilità che due funzioni differenti con nomi diversi possano finire per avere lo stesso identificatore. Solidity traccia e gestisce quando questo accade con funzioni dello stesso contratto, ma non quando accade tra più contratti.

[32] *What is an ABI? Example and Usage.* URL: https://www.alchemy.com/overviews/what-is-an-abi-of-a-smart-contract-examples-and-usage.

La soluzione al problema è l'utilizzo di un **transparent proxy pattern**. Un transparent proxy deciderà se una chiamata sia delegata al logic contract basandosi sull'indirizzo del chiamante.

* se il chiamate è l'admin del proxy, quest'ultimo non delega nessuna chiamata e risponde direttamente;
* se il chiamate è un qualsiasi altro indirizzo, il proxy delegherà sempre le chiamate a prescindere;

## 4.3 Codifica

In questa sezione verranno trattate la parti principali e più interessanti della codifica.

### 4.3.1 Ordine di sviluppo

Come riportato in precedenza nella descrizione dello stage, ogni smart contract presente nel pacchetto è strettamente collegato agli altri. Ognuno di questi si occupa di soddisfare una particolare richiesta per la realizzazione degli obiettivi. È stato ugualmente individuato un ordine con cui procedere nella codifica, anche se a causa dell'insperienza di sviluppo in questo ambito, in alcuni casi è stato necessario rivedere la struttura del codice.
L'ordine di sviluppo è il seguente:

1. contratto per la creazione degli NFT che segue lo standard ERC721 utilizzato dal contratto Issuer;
2. contratto che si occupa dell'emissione degli NFT access token (rispettivamente **Issuer**). Gestisce la costruzione dei token associando i relativi metadata (spiegati nella sottosezione 3.3.3) e implementa tutti i casi d'uso riportati in sottosezione A.2.4;
3. contratto che gestisce i processi di verifica, revoca e gli account verificatori (rispettivamente **Verifier e Revoker**). Fornisce un meccanismo di gestione dei verificatori e dei relativi verification record, seguendo rispettivamente quanto riportato in sottosezione 3.3.1 e in sottosezione 3.3.2. Implementa tutti i casi d'uso riportati in sezione 3.2.1 e sottosezione A.2.3;
4. contratto per il tracciamento dei contratti trusted e untrusted. Questo contratto è una realizzazione della best practice riportata in sezione 4.2.2, e non si riferisce a requisiti o casi d'uso specifici;
5. sviluppo collaborativo del PoC;

Successivo allo sviluppo di ognuno di questi punti, si è svolto un confronto con il tutor aziendale per valutare la soluzione proposta, eventuali modifiche e come procedere.

### 4.3.2 Differenze dalla progettazione

Il prodotto sviluppato differisce parecchio da quello pensato durante la fase di progettazione. Sono state necessarie diverse modifiche per adattare le richieste ai problemi legati principalmente allo sviluppo su blockchain e ad aspetti di cui non si era a conoscenza, in particolare:

* **Sistema di emissione:** si è resa necessaria una modifica per problemi legati alla privacy. Ci si è resi conto, dopo un primo studio preliminare, che non è possibile generare e tenere traccia delle credenziali verificabili direttamente all'interno di uno smart contract;

* **Sistema di verifica:** in un sistema decentralizzato la mancanza di fiducia in questo caso risulta essere un problema, è quindi necessario aggiungere dei meccanismi per costruirla. In particolare la soluzione:

  – non deve caricare o salvare i dati dell'utente on-chain, in quanto tra le tante problematiche legate alla privacy, viola anche quanto presente nel regolamento n.2016/679 del garante per la protezione dei dati personali (GDPR)

  – non può sfruttare un hash della credenziale in quanto non può essere decrifrato dallo smart contract (come riportato nella sezione 3.2.1, un contratto non può celare una chiave privata utilizzabile).

  Una possibile strategia potrebbe essere quella di utilizzare un oracolo (discusso nella sottosezione 2.9.2) che permetta di mettere in comunicazione gli smart contract con quello che avviene off-chain. L'architettura di questa soluzione deve prevedere due componenti: uno smart contract (oracolo on-chain) e un servizio dedicato nel backend (oracolo off-chain). Il contratto dovrà avere un metodo pubblico, ad esempio `createRequest`, a cui passare l'indirizzo web da interrogare e l'attributo di cui si vuole sapere il valore (in questo caso ad esempio l'esito di un'eventuale verifica o il dato di una credenziale). Questo metodo una volta invocato, emetterà un evento che permetterà di notificare l'oracolo off-chain di una nuova richiesta. Quest'ultimo è composto da diversi servizi, distribuiti su più nodi, che interrogano l'indirizzo in questione e restituiscono al contratto la risposta. L'oracolo on-chain verifica quindi se è stato raggiunto il numero minimo di risposte uguali, e in tal caso, emette un evento che indica il fatto di aver raggiunto il consenso sul valore. In questo modo, lo smart-contract client che ha interrogato l'oracolo sa di avere la sua risposta.
  Per via del tempo limitato e della complessità di questa soluzione, il progetto realizza un processo non automatizzato che richiede di riflettere manualmente (compito del verificatore) le operazioni effettuate off-chain direttamente on-chain.

Queste modifiche sono state viste come una normale evoluzione di un'idea ancora acerba che ha lo scopo di testare i limiti e le potenzialità di una nuova tecnologia.

### 4.3.3 Problematiche riscontrate

Nonostante le diverse modifiche e accorgimenti richiesti, la fase di codifica si è conclusa con l'adempimento di tutti i requisiti individuati originariamente nel piano di lavoro. Di seguito sono esposte le problematiche principali riscontrate durante lo sviluppo di questo progetto, e la relativa soluzione adottata:

| Problema | Soluzione |
|---|---|
| Difficoltà nella gestione delle chiamate alla blockchain: alcuni dati venivano ottenuti con tempistiche anche molto differenti | Risolto: dopo numerose ricerche e tentativi il problema è stato risolto utilizzando un hook di ReactJs (`useEffect()`) che permette di aggiornare dinamicamente il componente al momento dell'arrivo corretto dei dati. |
| Errori nella cifratura usando la chiave privata del DID: la libreria scelta inizialmente per eseguire l'operazione ritornava sempre un errore | Risolto: dopo una ricerca più accurata è stata scelta JOSE come libreria ed è stato sviluppato un backend NodeJs per poterla utilizzare. |

## 4.4 Testing e Validazione

In questa sezione vengono riportati i processi di verifica e validazione del prodotto, descrivendo i tool utilizzati per valutare il corretto funzionamento e la qualità del prodotto.

### 4.4.1 Testing del codice

Per quanto non fosse un requisito dello stage, durante la codifica, alcune parti del codice hanno richiesto la stesura di test di unità.
Nonostante il codice sia stato scritto seguendo il maggior numero di best practice possibile, l'interazione tra i diversi contratti è particolarmente "delicata", infatti ha richiesto una maggior attenzione. Per questo motivo son stati sviluppati dei test di unità automatici che vanno a verificare i requisiti che i metodi di queste classi devono rispettare.
Per la redazione di tali test sono stati utilizzati:

* **Hardhat:** è un ambiente di sviluppo e test per l'ecosistema Ethereum. Mette a disposizione diverse componenti per l'editing, compilazione, debugging e deploying di smart contract e DApps.
* **Chai.js:** è una libreria javascript per fare testing. Fornisce metodi e interfacce per verificare l'esecuzione del codice.

```
IssuerManager
  ✔ Should get some accounts for verification and not
  Dependences Initialization
    ✔ Deploy VerificationRegistry smart contract to the testnet (186ms)
    ✔ Deploy MonokeeERC721 smart contract to the testnet (130ms)
    ✔ Should deploy IssuerManager smart contract to testnet (82ms)
    ✔ Grant MINTER_ROLE to IssuerManager so that can mint NFT access token (69ms)
    ✔ Create trusted verifiers that can add verifications to the registry (132ms)
    ✔ Create Verification Records for some accounts (251ms)
  Contract tests
    ✔ Test NFT issuing with automatic verification (1278ms)
    ✔ Consume NFT when a user changes it for the VC (210ms)
    ✔ Only the user that owns the NFT should do the approve (254ms)
    ✔ Only the contract owner should consume the token after user has done the approve (146ms)

TrustedSmartContractRegistry
  ✔ Should deploy (76ms)
  Test write contract functions
    ✔ Register a new trusted smart contract (71ms)
    ✔ A generic user not able to register a new trusted contract
    ✔ The sc's owner should edit registered trusted smart contract trust (124ms)
    ✔ A generic user should not able to change the trust of a registered contract
  Test read contract functions
    ✔ getContractCount() should return the number of registered contracts
    ✔ getContract() should return contract informations
    ✔ isTrusted(address) should return true if the contract is trusted

VerificationRegistry2
  ✔ Get some hardhat accounts to do testing
  ✔ Should deploy (102ms)
  ✔ Should not find a verifier for an untrusted address
  ✔ The contract's owner should become a registered verifier (70ms)
  ✔ Should ensure owner address maps to a verifier
  ✔ Should have one verifier
  ✔ Should find a verifier for owner address
  ✔ Should update an existing verifier (73ms)
  ✔ Should remove a verifier (136ms)
  ✔ Should register a new verifier (81ms)
  ✔ Should format a structured verification result
  ✔ Should sign and verify typed data
  ✔ Should see the subject has no registered valid verification record
  ✔ Should register the subject as verified and create a Verification Record (141ms)
  ✔ Should verify the subject has a registered and valid verification record
  ✔ Get all verifications for a subject address
  ✔ Get all verifications for a verifier address
  ✔ Get a verification using its uuid
  ✔ Revoke a verification based on its uuid (76ms)
  ✔ Should remove a verification (221ms)


39 passing (4s)
```

**Figura 4.6:** Report test di unità

### 4.4.2 Validazione

Di seguito viene descritto il processo di validazione del codice prodotto durante il periodo di stage.
Questo processo è avvenuto in due fasi:

1. **presentazione lavoro svolto individualmente:** dimostrazione di una demo eseguita sulla shell linux con lo scopo di mostrare all'azienda ciò che si è realizzato nella prima parte del percorso di stage e determinare il grado di soddisfazione dei requisiti stabiliti. La prima demo è avvenuta direttamente in azienda, con presenti il tutor aziendale e il titolare;

2. **proof-of-concept svolto collaborativamente:** è stata presentata la webapp sviluppata e illustrato il percorso che l'informazione compie dalla creazione e gestione delle chiavi fino all'ottenimento di una credenziale rilasciata tramite la consumazione dell'NFT. Questa volta la presentazione è avvenuta da remoto, con presenti nuovamente sia il tutor aziendale che il titolare.

# Capitolo 5

# Conclusioni

*In questo capitolo verranno riportate le conclusioni relative allo stage e al prodotto realizzato, possibili miglioramenti e una valutazione personale.*

## 5.1 Possibili miglioramenti ed estensioni future

Durante l'ultima settimana di stage e la stesura del seguente documento, sono emerse diverse possibili migliorie del prodotto realizzato, che per mancanza di tempo o per le compentenze richieste, non è stato possibile affrontare. Se ne riportano qui alcune considerazioni.
Alcuni puristi della tecnologia blockchain affermano che gli smart contract aggiornabili favoriscano comportamenti illeciti, infatti nell'arco di poche ore il proprietario dei contratti potrebbe deployare una nuova versione in cui è inserito codice malevolo, ingannando gli utenti che non si sono resi conto delle modifiche. Per quanto questa operazione possa sembrare scorretta (o addirittura illegale) è perfettamente lecita e permessa dalla filosofia su cui si basa questa tecnologia. Senza entrare ulteriormente nella controversia, dopo aver dedicato questo periodo di stage a studiare queste tematiche, personalmente sostengo che sia comunque necessario sviluppare e ideare un meccanismo che permetta di salvaguardare i dati e i fondi di un contratto in caso di errori e nell'eventualità di dover deployare un nuovo contratto. Dover spostare quanto registrato nel vecchio contratto potrebbe essere un'operazione costosissima e rischierebbe di danneggiare la fiducia che gli utenti ripongono sull'azienda o sulla realtà di riferimento. Inoltre, secondo me, questo meccanismo di salvaguardia dovrebbe essere accompagnato da un processo di educazione sia per chi vuole offrire un servizio di questo tipo, sia per chi ne vuole usufruire, in modo da rendere tutti più consapoveli delle potenzialità offerte e delle responsabilità da assumersi quando si utilizza questa tecnologia.
Come riportato anche nelle sezioni di descrizione del prodotto, quest'ultimo è un ottimo punto di partenza per lo sviluppo di una soluzione più completa e strutturata in ambito Self-Sovereign Identity e blockchain. Dopo aver preso le precauzioni sopra descritte, un prossimo passo potrebbe essere quello di verificare se ci sono tutte le caratteristiche per registrare un nuovo standard.

## 5.2 Conoscenze acquisite

Oltre alle conoscenze apprese grazie al corso di studi, sono state fondamentali anche altre conoscenze apprese durante lo stage, come:

* Modello SSI e credenziali verificabili: la gestione dell'identità digitale è una tematica che ha un'importanza e una diffusione sempre maggiore, aver appreso un nuovo modello utilizzato per trattarla è stato molto interessante e sicuramente sarà una conoscenza preziosa per il mio percorso da informatico.
* NFT: lo standard associato a questi token digitali è alla base delle conoscenze necessarie per sviluppare sul mondo blockchain. Averlo studiato e usato concretamente mi permetterà di essere più competente e preparato.
* Solidity: è il linguaggio utilizzato per sviluppare gli smart contract, nonostante avessi già appreso le sue basi durante il progetto di ingegneria del software, questo stage mi ha permesso di maturare molto e imparare best practice e pattern che non conoscevo.
* ReactJS: è una delle librerie javascript più richieste e utilizzate a livello globale e questo periodo mi ha permesso di consolidare ulteriormente le mie conoscenze sul suo utilizzo.

## 5.3 Valutazione personale

Il progetto di stage offerto da Athesys e Monokee, a mio parere, si è svolto al meglio. Il lavoro è stato correttamente ponderato per le 320 ore lavorative previste, e l'azienda e il tutor, Mattia Zago, son sempre stati disponibili per chiarimenti o incontri in caso di problemi.
Nonostante il disorientamento provato nelle prime due settimane, dovuto alla libertà di studio e di progettazione concessa su argomenti sconosciuti, e in alcuni casi addirittura troppo nuovi per avere una documentazione sufficiente, alla fine si è rivelata a mio parere una scelta vincente. Mi ha permesso di imparare e spaziare al massimo le mie conoscenze su quello che questa realtà ha da offrire e ha potenziato non solo le mie competenze da programmatore ma anche una serie di soft-skills sull'analisi e progettazione di soluzioni a problemi non elementari.

# Appendice A

# Documento tecnico a supporto

*In questo capitolo verrà riportato il documento che riporta lo studio dei casi d'uso effettuato.*

## A.1 Introduzione

Lo scopo di questo documento è riportare i casi d'uso individuati per lo sviluppo del pacchetto di smart contract richiesti.

### A.1.1 Struttura del documento

Il documento riporta due sezioni principali:

* una descrizione formale dei casi d'uso;
* una tabella di tracciamento dei requisiti con relativa descrizione.

## A.2 Casi d'uso

I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo Unified Modeling Language (UML) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.
Ogni caso d'uso ha un codice gerarchico ed univoco che lo identifica, nella forma:

`UC<CodicePadre>.<CodiceFiglio>`

Il codice progressivo può includere diversi livelli di gerarchia separati da un punto.

### A.2.1 Attori

Gli attori individuati dopo un'attenta analisi sono i seguenti (Figura A.1):

* **Utente:** rappresenta un utente generico che può interagire con le funzionalità base del sistema;
* **Contract owner:** rappresenta il proprietario del contratto;

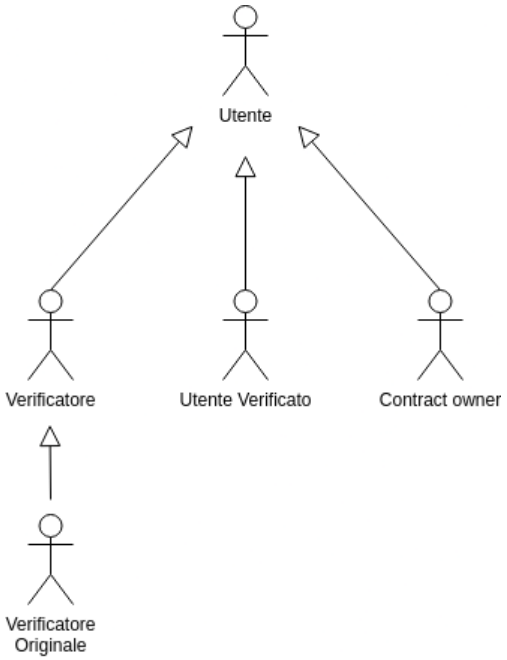



**Figura A.1:** Attori individuati

* **Verificatore:** è un utente che è stato inserito come verificatore da parte del proprietario del contratto;
* **Verificatore originale:** è un utente verificatore che ha registrato un record on-chain, ed è quindi l'unico a poter modificare quel record;
* **Utente verificato:** è un utente che possiede un record di verifica valido associato;

### A.2.2 Gestione dei verificatori

I casi d'uso riguardanti la gestione dei verificatori possono essere riassunti mediante il seguente diagramma UML:

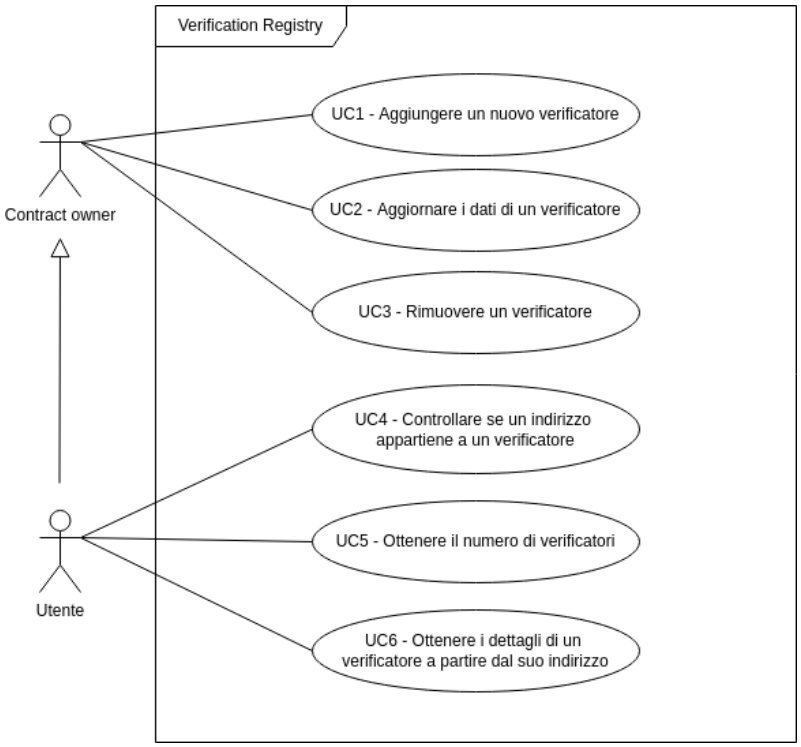



**Figura A.2:** Use Case gestione dei verificatori

### UC1 - Aggiungere un nuovo verificatore

* **attore primario:** Contract owner;
* **attore secondario:** nessuno;
* **descrizione:** il proprietario del contratto può aggiungere un nuovo verificatore on-chain;
* **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - l'indirizzo del verificatore che si vuole aggiungere non deve essere già presente;
* **postcondizione:** il proprietario del contratto ha aggiunto un nuovo verificatore;
* **scenario principale:**
  1. il proprietario del contratto invoca il metodo dello smart contract per registrare un nuovo verificatore passando l'indirizzo on-chain del soggetto e un array contenente i dettagli;

**UC2 - Aggiornare i dati di un verificatore**

* **attore primario:** Contract owner;
* **attore secondario:** nessuno;
* **descrizione:** il proprietario del contratto può aggiornare i dati di un verificatore on-chain inserito in precedenza;
* **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - l'indirizzo del verificatore che si vuole aggiornare deve essere già presente (UC1);
* **postcondizione:** il proprietario del contratto ha aggiunto un nuovo verificatore;
* **scenario principale:**
  1. il proprietario del contratto invoca il metodo dello smart contract per aggiornare un verificatore passando l'indirizzo on-chain del soggetto e un array contenente i dettagli aggiornati;

**UC3 - Rimuovere un verificatore**

* **attore primario:** Contract owner;
* **attore secondario:** nessuno;
* **descrizione:** il proprietario del contratto può rimuovere i dati di un verificatore on-chain inserito in precedenza;
* **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - l'indirizzo del verificatore che si vuole rimuovere deve essere già presente (UC1);
* **postcondizione:** il proprietario del contratto ha rimosso i dati del verificatore;
* **scenario principale:**
  1. il proprietario del contratto invoca il metodo dello smart contract per eliminare il verificatore passando l'indirizzo on-chain del soggetto;

**Considerazioni** In questo caso d'uso è opportuno specificare un dettaglio. La blockchain per sua natura è immutabile, questo significa che questo processo di eliminazione è molto diverso da quello che avviene ad esempio all'interno di un database. La rimozione di un verificatore consiste nel prendere il record on-chain e inizializzarlo ai valori di default stabili dal linguaggio.

**UC4 - Controllare se un indirizzo appartiene a un verificatore**

* **attore primario:** Utente;
* **attore secondario:** nessuno;
* **descrizione:** un utente può interrogare il contratto per sapere se un indirizzo è associato a un verificatore;
* **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - l'indirizzo del verificatore che si vuole verificare deve essere già presente (UC1);
* **postcondizione:** l'utente sa se un indirizzo appartiene o meno a un verificatore registrato;
* **scenario principale:**
  1. l'utente invoca il metodo dello smart contract per verificare se l'indirizzo passato appartiene o meno a un verificatore registrato;

**UC5 - Ottenere il numero di verificatori**

* **attore primario:** Utente;
* **attore secondario:** nessuno;
* **descrizione:** un utente può interrogare il contratto per sapere quanti verificatori sono stati registrati;
* **precondizione:** il contratto deve essere stato correttamente deployato on-chain;
* **postcondizione:** l'utente sa il numero di verificatori registrati;
* **scenario principale:**
  1. l'utente invoca il metodo dello smart contract per ottenere il numero di verificatori registrati;

**UC6 - Ottenere i dettagli dei un verificatore a partire dal suo indirizzo**

* **attore primario:** Utente;
* **attore secondario:** nessuno;
* **descrizione:** un utente può interrogare il contratto per ottenere i dettagli di un verificatore;
* **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - l'indirizzo del verificatore di cui si vuole ottenere i dettagli deve essere già presente (UC1);
* **postcondizione:** l'utente ha ricevuto i dettagli del verificatore registrato;

* **scenario principale:**

    1. l'utente invoca il metodo dello smart contract per ottenere i dettagli del verificatore registrato;

### A.2.3 Gestione dei verification record

I casi d'uso riguardanti la gestione dei verification record possono essere riassunti mediante il seguente diagramma UML:

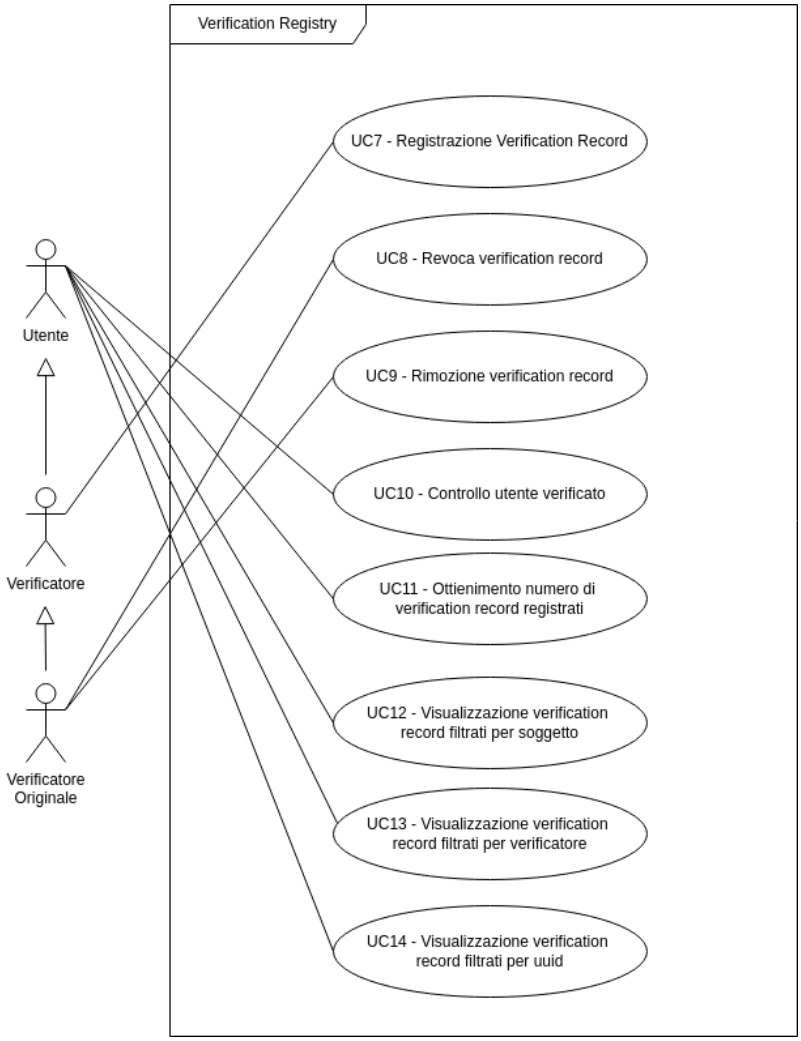



**Figura A.3:** Use Case gestione dei verication record

#### UC7 - Registrazione Verification Record

* **attore primario:** Verificatore;
* **attore secondario:** nessuno;

- **descrizione:** un verificatore può registrare un nuovo verification record;
- **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - il verificatore che vuole aggiungere un verification record deve essere stato registrato precedentemente (UC1);
- **postcondizione:** il verificatore ha registrato un nuovo verification record on-chain;
- **scenario principale:**
  1. il verificatore invoca il metodo dello smart contract per registrare un nuovo record passando un verification result e una typed signature dei dati;

**UC8 - Revoca verification record**

- **attore primario:** Verificatore originale;
- **attore secondario:** nessuno;
- **descrizione:** il verificatore originale del record può revocarlo;
- **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - il verification record che si vuole revocare deve essere stato registrato on-chain dal verificatore originale (UC7)
- **postcondizione:** il verificatore ha revocato il verification record on-chain;
- **scenario principale:**
  1. il verificatore originale invoca il metodo dello smart contract per revocare un verification record passando l'identificativo univoco e il tipo di verifica a cui è associato;

**Considerazioni** Per eseguire la revoca di un verification record si è optato per la scelta di aggiornare semplicemente un suo flag. È un'operazione abbastanza banale ma che grazie alla tecnologia blockchain offre diverse garanzie. Quando viene invocato il metodo, dopo che il flag revocato viene settato a *true*, viene lanciato un evento **VerificationRevoked(uuid)** che testimonia che il record con quel particolare uuid è stato revocato dal suo verificatore. Quest'informazione non può essere in alcun modo contraffatta e anzi dà la sicurezza che ad averlo revocato è stato proprio il suo verificatore.

**UC9 - Rimozione verification record**

- **attore primario:** Verificatore originale;
- **attore secondario:** nessuno;
- **descrizione:** il verificatore originale del record può rimuoverlo;

- **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - il verification record che si vuole revocare deve essere stato registrato on-chain dal verificatore originale (UC7)
- **postcondizione:** il verificatore ha rimosso il verification record on-chain;
- **scenario principale:**
  1. il verificatore originale invoca il metodo dello smart contract per rimuovere un verification record passando l'identificativo univoco e il tipo di verifica a cui è associato;

**UC10 - Controllo utente verificato**

- **attore primario:** Utente;
- **attore secondario:** nessuno;
- **descrizione:** un utente può verificare se un indirizzo è associato a un verification record valido;
- **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - l'utente che si sta controllando deve avere un verification record non revocato o scaduto (UC7)
- **postcondizione:** un utente sa se un indirizzo ha un verification record valido associato;
- **scenario principale:**
  1. l'utente invoca il metodo dello smart contract per sapere se un indirizzo ha un verification record non revocato o scaduto;

**Considerazioni** Per questo caso d'uso è opportuno specificare che un utente è definito verificato nel momento in cui possiede un verification record di cui è il soggetto, che non sia stato revocato (UC8) e non sia scaduto.

**UC11 - Ottenimento numero di verification record registrati**

- **attore primario:** Utente;
- **attore secondario:** nessuno;
- **descrizione:** un utente può ottenere il numero di verification record registrati;
- **precondizione:** il contratto deve essere stato correttamente deployato on-chain;
- **postcondizione:** un utente ottiene il numero di verification record registrati;
- **scenario principale:**
  1. l'utente invoca il metodo dello smart contract per sapere il numero di verification record registrati;

**UC12 - Visualizzazione verification record filtrati per soggetto**

- **attore primario:** Utente;
- **attore secondario:** nessuno;
- **descrizione:** un utente può ottenere una lista di verification record associati a un determinato soggetto;
- **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - deve esistere almeno un record associato al soggetto (UC7);
- **postcondizione:** un utente ottiene la lista di verification record associati a un determinato soggetto;
- **scenario principale:**
  1. l'utente invoca il metodo dello smart contract per ottenere la lista di verification record associati a un determinato soggetto;

**UC13 - Visualizzazione verification record filtrati per verificatore**

- **attore primario:** Utente;
- **attore secondario:** nessuno;
- **descrizione:** un utente può ottenere una lista di verification record associati a un determinato verificatore;
- **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - deve esistere almeno un record associato al verificatore (UC7);
- **postcondizione:** un utente ottiene la lista di verification record associati a un determinato verificatore;
- **scenario principale:**
  1. l'utente invoca il metodo dello smart contract per ottenere la lista di verification record associati a un determinato verificatore;

**UC14 - Visualizzazione verification record filtrati per uuid**

- **attore primario:** Utente;
- **attore secondario:** nessuno;
- **descrizione:** un utente può ottenere il verification record associato a un preciso uuid;
- **precondizione:** il contratto deve essere stato correttamente deployato on-chain;
- **postcondizione:** un utente ottiene il verification record associato allo specifico uuid o un record vuoto;

- **scenario principale:**

  1. l'utente invoca il metodo dello smart contract per ottenere il verification record associato a uno specifico uuid;

**Considerazioni UC12, UC13, UC14** Una funzionalità molto utile è sicuramente quella di poter filtrare i dati presenti nello smart contract, purtroppo non è possibile adottare un'unica struttura dati in quanto l'unico modo per interrogarla sarebbe quello di iterarla, processo che obbliga ogni operazione che richiede l'accesso ai dati ad avere un costo minimo $n$ pari al numero di record registrati. Per questo motivo una best practise spesso usata nell'ambito dello sviluppo di smart contract è quello di prediligere una rindondanza dei dati. In questo caso, per esempio, è possibile creare tre diverse mappe (una per ogni tipologia di filtro) indicizzate utilizzando un'associazione chiave-valore. Quindi questo permette di accedere ai dati direttamente tramite dereferenziazione, con costo costante.

### A.2.4 Gestione rilascio credenziali verificabili

I casi d'uso riguardanti la gestione del rilascio delle credenziali verificabili possono essere riassunti mediante il seguente diagramma UML:

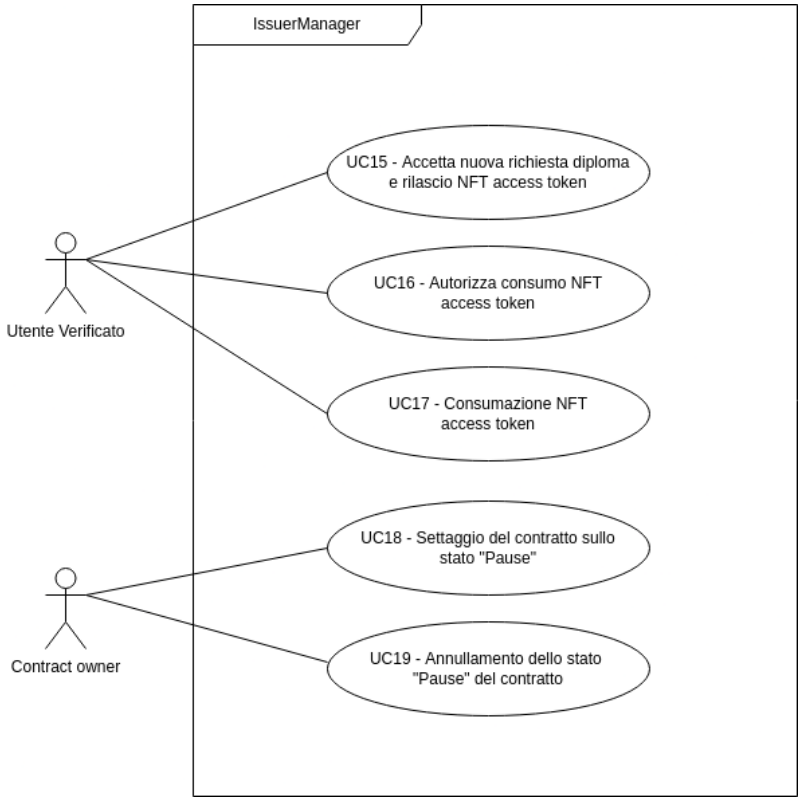



**Figura A.4:** Use Case emissione credenziali verificabili

**UC15 - Accettazione nuova richiesta credenziale verificabile e rilascio NFT access token**

* **attore primario:** Utente verificato;
* **attore secondario:** nessuno;
* **descrizione:** un utente verificato può richiedere il rilascio di una credenziale verificabile e ottenere un NFT access token;
* **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - l'utente deve possedere un record di verifica valido associato (UC7);
* **postcondizione:** l'utente verificato ottiene un NFT che funge da access token per ottenere la credenziale verificabile;
* **scenario principale:**
  1. l'utente verificato invoca il metodo dello smart contract per richiedere il rilascio di una credenziale verificabile;
  2. il contratto inoltra la richiesta allo smart contract Ethereum Request for Comment (ERC)721 per creare l'NFT, associandolo all'indirizzo dell'utente;

**Considerazioni** Essendo un contratto con lo scopo di rilasciare credenziali verificabili, questo caso d'uso definisce appunto come debba svolgersi questa procedura. In particolare essendo in ambito SSI è l'utente che muove il primo passo, richiedendo la credenziale verificabile lui stesso, cui seguirà il rilascio di un token NFT solo nel caso in cui possegga un verification record valido nel contratto **Verifier** (UC10).

**UC16 - Autorizza consumo NFT access token**

* **attore primario:** Utente verificato;
* **attore secondario:** nessuno;
* **descrizione:** un utente può autorizzare l'issuer a consumare il suo NFT access token;
* **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - l'utente verificato deve possedere un NFT access token (UC15);
* **postcondizione:** l'utente verificato ha approvato la consumazione del NFT access token;
* **scenario principale:**
  1. l'utente verificato invoca il metodo dello smart contract per approvare la consumazione del token da parte dell'issuer;

**Considerazioni:** Un NFT dimostra la proprietà di un contenuto digitale di un indirizzo (utente), in questo caso è utilizzato come access token per poter ottenere la credenziale verificabile desiderata. L'idea quindi è quella di sfruttare le garanzie dell'NFT per poter emettere la credenziale verificabile off-chain direttamente sul wallet dell'utente.
Una volta utilizzato, il token deve essere consumato (UC17) per non poter essere nuovamente spendibile, quindi è necessario che l'utente fornisca l'approvazione affinchè il contratto issuer possa procedere all'operazione di burn[1], poichè solo il proprietario del token ha i permessi di trasferirlo.

**UC17 - Consumazione NFT access token**

* **attore primario:** Contract owner;
* **attore secondario:** nessuno;
* **descrizione:** il proprietario del contratto richiede la consumazione del NFT access token;
* **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - un utente verificato deve aver approvato all'issuer il permesso di trasferire il proprio NFT (UC16);
* **postcondizione:** il proprietario del contratto ha consumato l'NFT;
* **scenario principale:**
  1. il proprietario del contratto invoca il metodo dello smart contract per richiedere la consumazione dell'NFT;

**UC18 - Settaggio del contratto sullo stato "pause"**

* **attore primario:** Contract owner;
* **attore secondario:** nessuno;
* **descrizione:** il proprietario del contratto richiede di settare il contratto sullo stato "pause";
* **precondizione:** il contratto deve essere stato correttamente deployato on-chain;
* **postcondizione:** il proprietario del contratto ha settato il contratto sullo stato "pause";
* **scenario principale:**
  1. il proprietario del contratto invoca il metodo dello smart contract per settare lo stato su "pause";

[1] OpenZeppelin, *ERC721 _burn*.

**UC19 - Annullamento dello stato "pause" del contratto**

* **attore primario:** Contract owner;
* **attore secondario:** nessuno;
* **descrizione:** il proprietario del contratto richiede di settare il contratto sullo stato "unpause";
* **precondizioni:**
  - il contratto deve essere stato correttamente deployato on-chain;
  - il contratto deve essere stato settato sullo stato "pause" (UC18).
* **postcondizione:** il proprietario del contratto ha settato il contratto sullo stato "unpause";
* **scenario principale:**
  1. il proprietario del contratto invoca il metodo dello smart contract per settare lo stato su "unpause";

**Considerazioni UC18, UC19** Questo caso d'uso è il complementare di UC18 e permette al proprietario del contratto di annullare lo stato di pausa del contratto. Questi due casi d'uso hanno lo scopo di prevenire eventuali emissioni di credenziali nel momento in cui si dovesse scoprire la presenza di un bug nel funzionamento del contratto, in modo da "bloccare" il sistema ed evitare ad esempio emissioni non autorizzate. L'adozione di questi casi d'uso è ormai un pattern consolidato e raccomandato.

## A.3 Tracciamento dei requisiti

In questa sezione, vengono riportati i requisiti del progetto, classificati per obbligatorietà. Ciascun requisito possiede un codice identificativo, il cui formalismo viene riportato di seguito:

```
R<NumeroRequisito>.<NumeroSottoRequisito>-<Classificazione>
```

La classificazione andrà a specificare l'obbligatorietà del requisito, e potrà essere indicata tramite una delle due seguenti sigle:

* **O**: requisito obbligatorio
* **D**: requisito desiderabile

### A.3.1 Requisiti

| Requisito | Descrizione | Caso d'uso |
|---|---|---|
| R1-O | Implementazione di uno smart contract per l'emissione di credenziali verificabili (issuer) | - |
| R1.1-O | L'issuer deve rilasciare un NFT che funga da access token per ottenere la credenziale verificabile | UC15 |

| | | |
|---|---|---|
| R1.2-O | L'issuer deve essere in grado di bruciare l'NFT quando viene utilizzato per ottenere la credenziale | UC16, UC17 |
| R1.3-D | Il proprietario del contratto issuer deve poterlo mettere in pausa in caso di bug nel suo funzionamento | UC18 |
| R1.4-D | Il proprietario del contratto issuer deve poterlo far uscire dallo stato di pausa nel caso il bug non creasse più problemi | UC19 |
| R2-O | Implementazione di uno smart contract per la revoca di credenziali verificabili (revoker) | - |
| R2.1-O | Il revoker deve poter revocare la verifica di una credenziale verificabile | UC8 |
| R2.2-D | Il revoker deve poter eliminare i dati associati alla verifica di una credenziale verificabile | UC9 |
| R3-O | Implementazione di uno smart contract per la verifica di credenziali verificabili (verifier) | - |
| R3.1-O | Il verifier deve permettere a chiunque di controllare se un soggetto è verificato | UC10 |
| R3.2-O | Il verifier deve poter restituire i dati associati a un verification record | UC12, UC13, UC14 |
| R3.3-O | Il verifier deve poter restituire le verifiche associate a un soggetto | UC12 |
| R3.4-O | Il verifier deve poter restituire le verifiche associate a un verificatore | UC13 |
| R3.5-O | Il verifier deve poter identificare univocamente una verifica | UC14 |
| R3.6-D | Il verifier deve ritornare il numero di verifiche registrate | UC11 |
| R3.7-O | Il verifier deve permettere al proprietario del contratto di aggiungere un verificatore | UC1 |
| R3.8-O | Il verifier deve permettere al proprietario del contratto di aggiornare un verificatore | UC2 |
| R3.9-O | Il verifier deve permettere al proprietario del contratto di rimuovere un verificatore | UC3 |
| R3.10-O | Il verifier deve permettere a chiunque di controllare se un utente è un verificatore | UC4 |
| R3.11-D | Il verifier deve poter restituire il numero di verificatori registrati | UC5 |
| R3.12-O | Il verifier deve permettere a chiunque di recuperare i dati relativi a un verificatore | UC6 |

**Tabella A.1:** Tracciamento dei requisiti

# Acronimi e abbreviazioni

**ABI** Application Binary Interface. 50

**API** Application Program Interface. 22, 23, 37, 40, 73, 74

**CID** Content Identifier. 28

**DAO** Decentralized Autonomous Organization. 45

**DApp** Decentralized Application. 2, 15, 24, 25, 33, 37–39, 53

**DID** Decentralized Identifier. 2, 6, 10–13, 23, 31–33, 52

**DID document** Decentralized Identifier document. 11, 12, 31

**DLT** Distributed Ledger Technologies. 23, 73

**DNS** Domain Name System. 26

**EBSI** European Blockchain Services Infrastructure. 23

**ECDSA** Elliptic Curve Digital Signature Algorithm. 41, 42

**EIP** Ethereum Improvement Proposals. 41–43, 49, 73

**ERC** Ethereum Request for Comment. 51, 67

**EVM** Ethereum Virtual Machine. 2, 21, 23, 29, 38, 48

**GDPR** garante per la protezione dei dati personali. 51

**HTTP** Hypertext Transfer Protocol. 37

**IAM** Identity and Access Management. 1, 74

**IDP** Identity Provider. 6, 13

**IPFS** Interplanetary File System. 27, 28, 33, 39, 41

**NFT** Non-Fungible Token. 26, 27, 30, 33, 34, 39–41, 51, 54, 56, 67–70

**PoC** Proof-of-Concept. 1, 42, 51, 73

**PoS** Proof-of-Stake. 15, 19, 73

**PoW** Proof-of-Work. 15, 19, 20, 73

**PSK** Pre-Shared Key. 5

**SDK** Software Development Kit. 2, 37

**SSI** Self-Sovereign Identity. 1–3, 6–8, 11, 13, 31, 37, 55, 56, 67

**UML** Unified Modeling Language. 57, 58, 62, 66, 74

**URI** Uniform Resource Identifier. 12, 41

**URL** Uniform Resource Locator. 27

**UUID** Universally Unique Identifier. 33

**VDR** Verifiable Data Registry. 12, 13

**ZKP** zero-knowledge proof. 7, 8

# Glossario

**Distributed Ledger Technologies** Fa riferimento a "libri mastri" (o registri) elettronici, distribuiti geograficamente su un'ampia rete di nodi, i cui dati sono protetti da potenziali attacchi informatici grazie al fatto che le stesse informazioni sono ridondate, verificate e validate mediante l'adozione di diversi protocolli (o regole) comunemente accettati da ciascun partecipante.. 71

**Ethereum Improvement Proposal** sono delle proposte che descrivono standard presenti nella piattaforma Ethereum, includono specifiche di protocolli, client e standard di smart contract. Ulteriori informazioni al seguente link. 71

**Proof-of-Concept** dall'inglese significa letteralmente "prova di concetto" ma può essere tradotto in prova di fattibilità, e si intende una realizzazione incompleta o abbozzata di un progetto, con il fine di dimostrarne la fattibilità di principi o concetti. 71

**Proof-of-Stake** Algoritmo di consenso introdotto nel 2011 con l'obiettivo di risolvere i problemi relativi al PoW. Si basa su un processo di elezione pseudo-casuale per selezionare un nodo che agirà da validatore del blocco successivo. I nodi che vogliono partecipare al processo di forging (L'inserimento di nuovi blocchi viene chiamato "forging" (forgiare dall'inglese) invece di mining (come avviene in PoW)) devono congelare una certa somma di monete all'interno della rete (è letteralmente il significato di staking). Il criterio di selezione si basa su una combinazione di fattori che possono includere periodo di staking, randomizzazione e fondi di proprietà del nodo. 71

**Proof-of-Work** Protocollo di validazione dei blocchi di una blockchain nato nel 2008 con Bitcoin. È un metodo che incentiva i miner a competere tra loro nell'elaborazione dei blocchi. La competizione consiste nel risolvere un problema matematico che richiede una grande potenza di calcolo (esempi di problemi possono essere trovare l'input di una funzione di hash partendo da un output; fare una scomposizione in numeri primi). 20, 72

**AGILE** La "metodologia agile" è un approccio allo sviluppo del software basato sulla distribuzione continua di software efficienti creati in modo rapido e iterativo. In pratica, le metodologie di sviluppo software agile consistono nel rilasciare rapidamente modifiche al software in piccole porzioni con l'obiettivo di migliorare la soddisfazione dei clienti. Le metodologie utilizzano approcci flessibili e il lavoro di gruppo per concentrarsi sul miglioramento continuo. La metodologia agile consente di adottare un approccio più leggero alla stesura della documentazione software e di integrare le modifiche in qualsiasi fase del ciclo di vita, anziché ostacolarle.. 2

**API** in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 71

**Identity and Access Management** è un servizio di gestione degli utenti e delle relative autorizzazioni con lo scopo di proteggere le risorse e i dati aziendali. 71

**metadata** si intende un insieme di informazioni che descrivono uno o più dati. 10, 13, 26, 33, 39, 51

**miner** Nodo speciale del network che mette a disposizione i suoi computer per il processo di mining. 16

**mining** Il termine mining deriva dall'inglese *to mine*, che significa estrarre, e nel caso dei bitcoin rappresenta il processo di condivisione della potenza di calcolo degli hardware dei partecipanti alla rete. La sicurezza nel sistema è affidata alla tecnologia blockchain, ossia un registro pubblico e condiviso delle transazioni in ordine cronologico. Ogni 10 minuti circa, il sistema produce un nuovo blocco con le nuove transazioni in attesa di approvazione, che sarà aggiunto alla catena una volta convalidato. Il mining è quindi in sintesi il processo che porta all'aggiunta di nuovi blocchi alla blockchain.. 74

**off-chain** Con l'espressione off-chain si intende qualsiasi dato o funzione esterna alla blockchain. È il contrario dell'espressione . 2, 16, 25–27, 30, 32, 33, 52, 68, 74

**on-chain** Con l'espressione on-chain si intende un qualsiasi dato o funzione registrata sulla blockchain. È il contrario dell'espressione . 2, 13, 16, 24, 26, 30–34, 51, 52, 58–61, 63–65, 67–69, 74

**open source** È un software per computer rilasciato con una licenza in cui il detentore del copyright concede agli utenti i diritti di utilizzare, studiare, modificare e distribuire il software e il suo codice sorgente a chiunque e per qualsiasi scopo. 2, 20, 21, 23

**phishing** è un tipo di truffa effettuata su Internet attraverso la quale un malintenzionato cerca di ingannare la vittima convincendola a fornire informazioni personali, dati finanziari o codici di accesso, fingendosi un ente affidabile in una comunicazione digitale. 8

**UML** in ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 72

**WhatsApp** è un servizio di messaggistica istantanea centralizzata multipiattaforma e freeware disponibile a livello internazionale e un servizio di voice-over-IP di proprietà della società americana Meta Platforms. 13

# Bibliografia

## Riferimenti bibliografici

Nakamoto, Satoshi. *Bitcoin: A Peer-to-Peer Electronic Cash System*. URL: https://bitcoin.org/bitcoin.pdf.

## Siti web consultati

*Algorand*. URL: https://www.algorand.com/.

*Apache License 2.0*. URL: https://www.apache.org/licenses/LICENSE-2.0.

*Assembly*. URL: https://docs.soliditylang.org/en/v0.8.16/assembly.html.

*Avalanche*. URL: https://www.avax.network/.

*Bitcoin.org*. URL: https://bitcoin.org/en/.

*Chainlink*. URL: https://chain.link/.

*Chakra UI*. URL: https://chakra-ui.com/.

*Checks Effects Interactions pattern*. URL: https://docs.soliditylang.org/en/develop/security-considerations.html?highlight=check%5C%20effects#use-the-checks-effects-interactions-pattern.

*Cheqd.io*. URL: https://cheqd.io/.

*Circuit Breaker*. URL: https://consensys.github.io/smart-contract-best-practices/development-recommendations/precautions/circuit-breakers/.

*Content addressing and CIDs*. URL: https://docs.ipfs.tech/concepts/content-addressing/.

*Corda*. URL: https://www.corda.net/.

Crowe. *The Financial Cost of Fraud*. URL: https://f.datasrvr.com/fr1/521/90994/0031_Financial_Cost_of_Fraud_2021_v5.pdf.

*Decentralized Identifiers*. URL: https://tykn.tech/decentralized-identifiers-dids/.

*DeFi Yield Attacks Database*. URL: https://defiyield.app/rekt-database.

*delegatecall*. URL: https://docs.soliditylang.org/en/latest/introduction-to-smart-contracts.html%5C#delegatecall-callcode-and-libraries.

*EIP-170: Contract Size Limit*. URL: https://eips.ethereum.org/EIPS/eip-170.

*EIP-1884*. URL: https://eips.ethereum.org/EIPS/eip-1884.

*EIP-1967*. URL: https://eips.ethereum.org/EIPS/eip-1967.

*EIP-2535: Diamond Standard*. URL: https://eips.ethereum.org/EIPS/eip-2535.

*EIP-721*. URL: https://eips.ethereum.org/EIPS/eip-721.

*Equifax data breach*. URL: https://www.cnet.com/news/privacy/equifaxs-hack-one-year-later-a-look-back-at-how-it-happened-and-whats-changed/.

*Ethereum Casper Explained*. URL: https://academy.binance.com/en/articles/ethereum-casper-explained.

*Ethereum Istanbul Hard Fork*. URL: https://ethereum.org/en/history/#istanbul.

*Ethereum.org*. URL: https://ethereum.org/en/.

*European Blockchain Services Infrastructure*. URL: https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home.

*Example of delegatecall*. URL: https://consensys.github.io/smart-contract-best-practices/development-recommendations/precautions/upgradeability/%5C#example-2-use-a-delegatecall-to-forward-data-and-calls.

*Fantom*. URL: https://fantom.foundation/.

*Hyperledger Besu*. URL: https://www.hyperledger.org/use/besu.

*Hyperledger Documentation*. URL: https://besu.hyperledger.org/en/stable/.

*Hyperledger Fabric*. URL: https://www.hyperledger.org/use/fabric.

*Hyperledger Indy*. URL: https://www.hyperledger.org/use/hyperledger-indy.

*Hyperledger.org*. URL: https://www.hyperledger.org/.

*Identità digitale*. URL: https://innovazione.gov.it/progetti/identita-digitale-spid-cie/.

*IPFS Hash*. URL: https://docs.ipfs.io/concepts/hashing/#important-hash-characteristics.

*JSON Web Almost Everything*. URL: https://www.npmjs.com/package/jose.

*Lightning Network*. URL: https://lightning.network/.

*Link rot*. URL: https://www.techopedia.com/definition/20414/link-rot.

*Manifesto Agile*. URL: http://agilemanifesto.org/iso/it/.

*MetaMask*. URL: https://metamask.io/.

*Multi-Factor Authentication*. URL: https://www.cisa.gov/publication/multi-factor-authentication-mfa.

*New steps in the development of the European Blockchain Services Infrastructure (EBSI)*. URL: https://digital-strategy.ec.europa.eu/en/news/new-steps-development-european-blockchain-services-infrastructure-ebsi.

OpenZeppelin. *AccessControl*. URL: https://docs.openzeppelin.com/contracts/4.x/api/access#AccessControl.

— *ECDSA*. URL: https://docs.openzeppelin.com/contracts/2.x/api/cryptography#ECDSA.

— *EIP712*. URL: https://docs.openzeppelin.com/contracts/4.x/api/utils#EIP712.

— *ERC721*. URL: https://docs.openzeppelin.com/contracts/4.x/api/token/erc721.

— *ERC721 _burn*. URL: https://docs.openzeppelin.com/contracts/4.x/api/token/erc721#ERC721-_burn-uint256-.

— *ERC721 approve*. URL: https://docs.openzeppelin.com/contracts/4.x/api/token/erc721#IERC721-approve-address-uint256-.

— *ERC721Burnable*. URL: https://docs.openzeppelin.com/contracts/4.x/api/token/erc721#ERC721Burnable.

— *ERC721URIStorage*. URL: https://docs.openzeppelin.com/contracts/4.x/api/token/erc721#ERC721URIStorage.

— *Ownable*. URL: https://docs.openzeppelin.com/contracts/4.x/api/access#Ownable.

— *Pausable*. URL: https://docs.openzeppelin.com/contracts/4.x/api/security#Pausable.

— *Proxies*. URL: https://docs.openzeppelin.com/upgrades-plugins/1.x/proxies.

— *Upgradeable Contracts*. URL: https://docs.openzeppelin.com/contracts/4.x/upgradeable.

— *Writing Upgradeable Smart Contracts*. URL: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable.

*Oracles*. URL: https://ethereum.org/en/developers/docs/oracles/.

Ore, Il Sole 24. *Scandalizzati da Facebook? I vostri dati sono in vendita da anni*. URL: https://www.ilsole24ore.com/art/scandalizzati-facebook-vostri-dati-sono-vendita-anni-AEocgTXE?refresh_ce=1.

personali, Garante per la protezione dei dati. *GDPR*. URL: https://www.garanteprivacy.it/regolamentoue.

*Pinata*. URL: https://pinata.cloud/.

*Plasma*. URL: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/plasma/.

*Polkadot*. URL: https://polkadot.network/.

*PolyNetwork*. URL: https://www.poly.network/.

*Progetto IBSI*. URL: https://progettoibsi.org/.

*Promise*. URL: https://www.w3schools.com/js/js_promise.asp.

*Proof of Stake*. URL: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/.

*Proof of Work*. URL: https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/.

*Raiden Network*. URL: https://raiden.network/.

*ReactJS*. URL: https://reactjs.org/.

*Reentrancy Attack*. URL: https://consensys.github.io/smart-contract-best-practices/attacks/reentrancy/.

*Ronin*. URL: https://bridge.roninchain.com/.

*Rust*. URL: https://www.rust-lang.org/.

*SEBA Bank*. URL: https://www.seba.swiss/.

*Solidity*. URL: https://docs.soliditylang.org/.

*Sovrin.org*. URL: https://sovrin.org/.

*State Channels*. URL: https://ethereum.org/en/developers/docs/scaling/state-channels/.

*Tezos*. URL: https://tezos.com/.

TG24, SKY. *Lo scandalo di Cambridge Analytica*. URL: https://tg24.sky.it/mondo/approfondimenti/facebook-gestione-dati-accuse.

*The DAO*. URL: https://github.com/blockchainsllc/DAO.

*The Graph*. URL: https://thegraph.com/.

*The Merge aggiornamenti*. URL: https://ethereum.org/en/upgrades/merge/.

*Understanding Blockchain Layers*. URL: https://cointelegraph.com/blockchain-for-beginners/a-beginners-guide-to-understanding-the-layers-of-blockchain-technology.

*Unix Timestamp*. URL: https://www.unixtimestamp.com/.

*Verifiable Data Registry*. URL: https://www.w3.org/TR/vc-data-model/#dfn-verifiable-data-registries.

*Vyper*. URL: https://vyper.readthedocs.io/en/stable/.

*W3C*. URL: https://www.w3.org/TR/vc-data-model/.

*Wagmi*. URL: https://wagmi.sh/.

*walt.id*. URL: https://walt.id/.

*WebAssembly*. URL: https://webassembly.org/.

*What is a DAO and how does it work?* URL: https://cointelegraph.com/decentralized-automated-organizations-daos-guide-for-beginners/what-is-decentralized-autonomous-organization-and-how-does-a-dao-work.

*What is an ABI? Example and Usage*. URL: https://www.alchemy.com/overviews/what-is-an-abi-of-a-smart-contract-examples-and-usage.

*What is IPFS?* URL: https://docs.ipfs.tech/concepts/what-is-ipfs/.

*What is Ripple?* URL: https://www.forbes.com/advisor/investing/cryptocurrency/what-is-ripple-xrp/.

*What is the Blockchain Oracle Problem?* URL: https://blog.chain.link/what-is-the-blockchain-oracle-problem/.

*What is the European Blockchain Services Infrastructure?* URL: https://ec.europa.eu/digital-building-blocks/wikis/display/EBSIDOC/What+is+EBSI.

*What is web3 and.* URL: https://ethereum.org/it/web3/.

*WHY IS THERE A LIMIT?* URL: https://ethereum.org/en/developers/tutorials/downsizing-contracts-to-fight-the-contract-size-limit/.

*Wormhole.* URL: https://wormhole.com/.

*Your Keys, Your Bitcoin. Not Your Keys, Not Your Bitcoin.* URL: https://cointelegraph.com/news/antonopoulos-your-keys-your-bitcoin-not-your-keys-not-your-bitcoin.

*Zero Knowledge Proofs.* URL: https://wiki.hyperledger.org/display/CP/Zero-Knowledge+Proof%3A+Verifying+Blockchain+Transactions+with+Less+Risk.

*Zero-Knowledge Proof: Verifying Blockchain Transactions with Less Risk.* URL: https://www.hyperledger.org/blog/2017/06/06/zero-knowledge-proofs.