



UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

**UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II**

DIPARTIMENTO DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE
INGEGNERIA INFORMATICA

Blockchain and Decentralized Application

Matricola:

M63001680
M63001649

Studente:

Rocco Marotta
Daniele Caputo

Anno accademico:

2024/2025

Indice

1 Descrizione del Progetto	2
Componenti principali	2
Strumenti di Sviluppo	3
Hardhat	3
Circom	3
React	3
Pinata	3
Express	4
2 Smart Contract	5
HealthRecordNFT.sol	5
Verifier.sol	6
3 Zero Knowledge Proof: Circom	7
Introduzione alle Zero Knowledge Proof (ZKP)	7
Circom	8
Sequence diagram: flusso di verifica età tramite ZKP	9
4 Confronto tra il nostro progetto e Patientory	11
Introduzione	11
Rappresentazione delle cartelle cliniche	11
Privacy e Zero Knowledge Proof	11
Gestione dei permessi	11
Architettura e blockchain privata	12
Considerazioni finali	12
5 Conclusioni	13
Difficoltà incontrate	13
Sviluppi futuri	13
Suddivisione delle fasi e contributi	14

1 Descrizione del Progetto

Il progetto **HealthApp_ETH** è una piattaforma basata su blockchain Ethereum per la gestione sicura dei dati sanitari tramite NFT (Non-Fungible Token) e Zero-Knowledge Proofs (ZKP). La gestione delle cartelle cliniche digitali in ambito sanitario presenta criticità legate a privacy, sicurezza e trasparenza degli accessi. Il nostro progetto mira a risolvere questi problemi sfruttando la blockchain per:

- Rendere trasparente e verificabile la gestione degli accessi ai dati sanitari.
- Garantire la privacy dei pazienti tramite Zero Knowledge Proofs.
- Offrire un sistema decentralizzato, sicuro e aggiornabile per la conservazione e la condivisione delle cartelle cliniche.

Questa soluzione potrebbe essere interessante per pazienti, medici e strutture sanitarie che necessitano di un controllo rigoroso e trasparente sui dati sensibili.

Componenti principali

- **Smart contract Solidity:** gestiscono la creazione e la verifica di NFT che rappresentano record sanitari, e la verifica di prove crittografiche (ZKP) per garantire privacy.
- **Script di deploy e test:** automatizzano il deploy dei contratti e testano le funzionalità.
- **Circuiti ZKP (Cartella Circom):** definiscono e generano prove crittografiche per validare informazioni (es. età) senza rivelare dati sensibili.
- **Frontend React:** interfaccia utente per interagire con la blockchain, visualizzare e gestire i record sanitari NFT.
- **Storage e IPFS:** gestisce il salvataggio e la condivisione sicura dei dati sanitari tramite IPFS.

Il progetto consente di gestire, condividere e verificare dati sanitari in modo sicuro e privato usando tecnologie blockchain e crittografia avanzata.

Strumenti di Sviluppo

Hardhat

Hardhat è un ambiente di sviluppo per smart contract su Ethereum. Permette di compilare, testare, effettuare il deploy e il debug dei contratti scritti in Solidity. Fornisce una rete locale per simulare la blockchain, strumenti per l'automazione dei test e plugin per l'integrazione con altri strumenti. Grazie a Hardhat, è possibile:

- Scrivere e compilare smart contract in modo semplice e modulare.
- Eseguire test automatici per verificare la correttezza dei contratti.
- Effettuare il deploy su reti locali o pubbliche.
- Debuggare le transazioni e tracciare gli errori.

Circom

Circom è un linguaggio e un compilatore per la creazione di circuiti Zero-Knowledge Proof (ZKP). Serve a definire circuiti crittografici che permettono di dimostrare la validità di un'informazione senza rivelare il dato stesso. Il flusso di lavoro tipico con Circom prevede:

- Scrivere il circuito in linguaggio Circom, specificando le regole di validazione.
- Compilare il circuito per ottenere i file necessari alla generazione delle prove (R1CS, zkey, ecc.).
- Generare una prova (proof) e un testimone (witness) a partire da input privati.
- Verificare la prova tramite smart contract o strumenti dedicati.

Circom è fondamentale per integrare la privacy tramite ZKP all'interno della piattaforma.

React

React è una libreria JavaScript per la creazione di interfacce utente dinamiche e reattive. Nel progetto viene utilizzata per sviluppare il frontend, permettendo agli utenti di interagire con la blockchain e visualizzare i dati sanitari in modo intuitivo. React consente di offrire un'esperienza utente moderna e interattiva, fondamentale per l'usabilità della piattaforma.

Pinata

Pinata è un servizio che facilita la gestione e il pinning di file su IPFS. Nel progetto viene utilizzato per caricare e conservare in modo affidabile i dati sanitari (come referti o documenti) sulla rete IPFS, garantendo che restino accessibili e immutabili. L'integrazione con Pinata permette di ottenere un CID (Content Identifier) per ogni file caricato, che viene poi associato agli NFT sanitari.

Express

Express è un framework backend per Node.js utilizzato nel progetto per gestire le API e le operazioni di upload dei file verso IPFS/Pinata. Attraverso Express vengono esposti endpoint che permettono agli utenti di caricare documenti sanitari, i quali vengono poi inviati a Pinata e collegati agli NFT tramite il CID restituito. Express funge quindi da ponte tra il frontend e l'infrastruttura di storage decentralizzato.

2 Smart Contract

Alla base del funzionamento della nostra DApp ci sono i due smart contract principali sviluppati per il progetto: **HealthRecordNFT.sol** e **Verifier.sol**.

HealthRecordNFT.sol

Questo smart contract realizza un sistema di gestione delle cartelle cliniche digitali sotto forma di NFT (Non-Fungible Token). Le sue principali funzionalità sono:

- **Minting:** consente a soggetti autorizzati (es. medici definiti con il ruolo di ORACLE) di creare nuovi NFT che rappresentano un record sanitario associato a un paziente. Ogni NFT contiene un riferimento (CID IPFS) ai dati sanitari memorizzati off-chain.
- **Ruoli e permessi:** utilizza il sistema di ruoli di OpenZeppelin per gestire i permessi di minting, aggiornamento e accesso ai record. Sono previsti ruoli come ADMIN, ORACLE.
- **Upgradeability:** il contratto è implementato come proxy UUPS, permettendo aggiornamenti futuri senza perdere i dati già registrati. Tutti gli account con il ruolo di Admin possono modificare il contratto implementazione a cui il proxy contract fa riferimento.
- **Verifica per fasce di età:** il contratto implementa una funzione che consente la verifica dell'età solo per alcune soglie predefinite (14, 16, 18, 21 anni). Questo accorgimento impedisce al medico di inviare numerose richieste con limiti diversi e risalire per inferenza all'età esatta del paziente, rafforzando la privacy.
- **Verifica età con ZKP:** integra la verifica di Zero Knowledge Proofs tramite il contratto Verifier. Quando un paziente fornisce una proof, questa viene verificata tramite HealthRecordNFT.sol: solo se la prova è valida e i vincoli del circuito (età superiore a una soglia definita dal medico) sono rispettati, il medico può visualizzare il risultato della verifica, senza che vengano rivelate altre informazioni sensibili.
- **Gestione metadati:** permette di associare e recuperare i metadati (CID IPFS) relativi ai record sanitari NFT.

Verifier.sol

Il contratto Verifier è responsabile della verifica delle prove crittografiche Zero Knowledge Proof (ZKP) generate off-chain tramite Circom e snarkjs. Le sue caratteristiche principali sono:

- **Verifica ZKP:** implementa la funzione `verifyProof` che riceve come input una prova zk-SNARK e i parametri pubblici, restituendo true se la prova è valida.
- **Integrazione con HealthRecordNFT:** viene utilizzato dal contratto HealthRecordNFT.sol per verificare, tramite la proof fornita dal paziente, se i vincoli definiti nel circuito sono rispettati.
- **Sicurezza:** la logica di verifica è generata automaticamente dagli strumenti zk-SNARK (Circom/snarkjs) e non può essere alterata, garantendo l'integrità delle verifiche.

Questi due smart contract lavorano insieme per garantire privacy, sicurezza e controllo rigoroso e trasparente sugli accessi ai dati sanitari digitali.

3 Zero Knowledge Proof: Circom

Introduzione alle Zero Knowledge Proof (ZKP)

Una Zero Knowledge Proof (ZKP) è un protocollo crittografico che permette a una parte (il prover) di dimostrare a un'altra parte (il verifier) di conoscere un'informazione o di soddisfare una certa proprietà, senza rivelare alcun dettaglio sull'informazione stessa. Questo garantisce privacy e sicurezza, poiché il dato sensibile non viene mai esposto durante la verifica. Di seguito viene mostrata l'immagine del flusso di lavoro tipico per la creazione e l'utilizzo di una Zero Knowledge Proof (zk-SNARK) utilizzando Circom e SnarkJS:

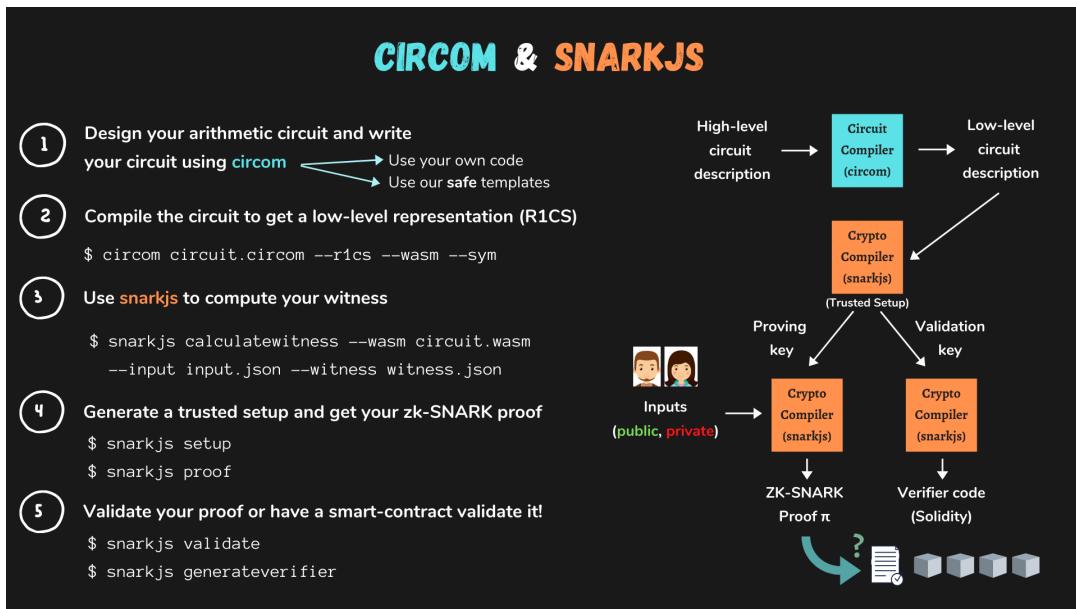


Figura 3.1: ZKP Circom SnarkJS

Vengono illustrati i principali passaggi: dalla progettazione del circuito aritmetico (step 1), alla compilazione in una rappresentazione a basso livello (step 2), al calcolo del witness (step 3), alla generazione della proof tramite trusted setup (step 4), fino alla validazione della proof o alla generazione del codice di verifica per smart contract (step 5). A destra, uno schema riassume come la descrizione del circuito viene compilata, come vengono prodotti i parametri crittografici e come questi vengono utilizzati per generare la proof e il codice di verifica. L'immagine evidenzia la

separazione tra input pubblici e privati e il ruolo centrale di Circom e SnarkJS nel processo.

Nel contesto del nostro progetto, le ZKP vengono utilizzate per permettere ai pazienti di dimostrare di possedere determinati requisiti (avere un'età superiore a una soglia) senza rivelare la loro data di nascita o altri dati personali.

Circom

La cartella **Circom** è organizzata in modo da coprire tutte le fasi del ciclo di vita di una Zero Knowledge Proof. Di seguito vengono distinte le principali categorie di file presenti:

- **Circuito vero e proprio:**

- **age_proof.circom:** questo circuito permette a un utente (paziente) di dimostrare, tramite una Zero Knowledge Proof, di essere nato prima (o entro) una certa data limite senza rivelare la propria data di nascita esatta. Di seguito viene brevemente illustrato il funzionamento generale del circuito
 - * Il circuito prende come input segreti la data di nascita (anno, mese, giorno) e un salt, e come input pubblici la data limite e il commitment (hash) della data di nascita.
 - * Verifica che il commitment calcolato dagli input segreti corrisponda a quello pubblico, garantendo che il paziente stia dimostrando qualcosa sulla propria vera data di nascita.
 - * Confronta la data di nascita con la data limite: restituisce una prova valida solo se la data di nascita è precedente o uguale alla data limite (cioè se il paziente ha almeno l'età richiesta).
 - * La logica di confronto è completa: considera anno, mese e giorno, gestendo tutti i casi di uguaglianza e disuguaglianza.
 - * L'output del circuito è 1 solo se la condizione è rispettata.

- **File di trusted setup e generati dalla compilazione:**

- **File di trusted setup:** questi file sono fondamentali per la sicurezza del protocollo zk-SNARK e vengono generati durante una fase iniziale detta "cerimonia di trusted setup". Servono a creare i parametri crittografici necessari per la generazione e la verifica delle prove, garantendo che nessuno possa falsificare le prove stesse.
- **build/:** cartella che contiene tutti i file generati dalla compilazione del circuito, come la rappresentazione R1CS, le chiavi di prova/verifica, il witness, la proof e i dati pubblici, oltre ai file necessari per la verifica on-chain.

- **File per l'automatizzazione:**

- **setup_circuit.js**: script Node.js che automatizza la compilazione del circuito, la generazione dei file di trusted setup e la creazione delle chiavi di prova e verifica. Permette di eseguire in sequenza tutte le operazioni necessarie per preparare l’ambiente ZKP.
- **run_zkp.sh**: script bash che automatizza l’intero processo di generazione della proof, dalla creazione del commitment, alla generazione del witness, fino alla produzione e verifica della proof stessa. Consente di eseguire tutti i passaggi con un solo comando, facilitando l’integrazione e i test.

Questa struttura permette di gestire in modo completo il ciclo di vita delle prove ZKP: dalla definizione del circuito, passando per il trusted setup, fino all’automatizzazione della generazione e verifica delle prove, integrandole poi nella logica degli smart contract del progetto.

Sequence diagram: flusso di verifica età tramite ZKP

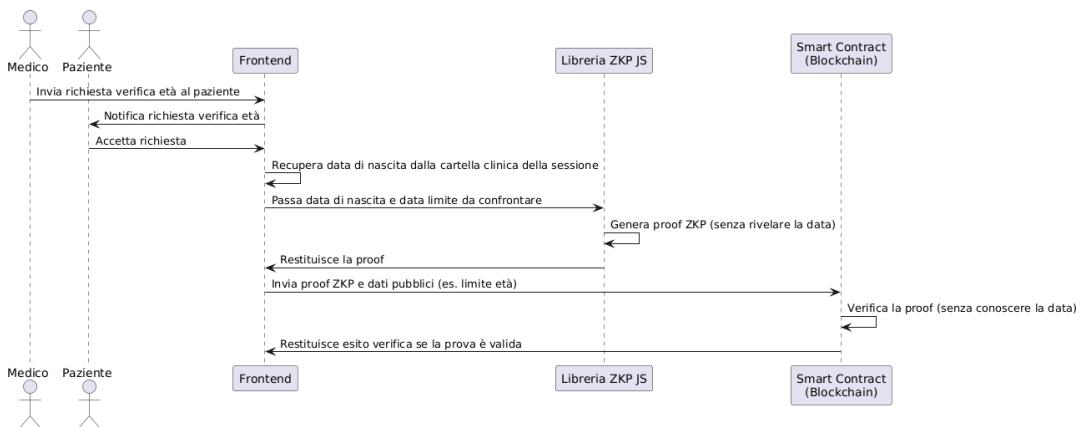


Figura 3.2: SD utilizzo ZKP

Il diagramma sopra mostra il flusso completo della verifica dell’età tramite Zero Knowledge Proof nel sistema:

- Il **medico** invia una richiesta di verifica età al paziente tramite il frontend.
- Il **paziente** riceve la notifica e accetta la richiesta.
- Il frontend recupera automaticamente la data di nascita del paziente dalla cartella clinica associata alla sessione (senza che il paziente debba inserirla manualmente).
- Il frontend passa la data di nascita e la data limite da confrontare alla libreria ZKP JS, che genera la proof ZKP senza mai rivelare la data di nascita reale.
- La proof viene restituita al frontend, che la invia allo smart contract insieme ai dati pubblici (nel nostro caso il risultato del confronto).
- Lo smart contract verifica la proof senza conoscere la data di nascita e restituisce l’esito della verifica.

Questo flusso garantisce che l'informazione sensibile (la data di nascita) non venga mai condivisa né con il medico né con la blockchain, ma resti sempre privata e gestita solo localmente dal paziente.

4 Confronto tra il nostro progetto e Patientory

Introduzione

Tra le varie soluzioni analizzate nel panorama della gestione decentralizzata dei dati sanitari, Patientory è risultata quella più affine al nostro approccio, pur presentando differenze tecniche e architetturali rilevanti che meritano di essere approfondite.

Rappresentazione delle cartelle cliniche

Una prima differenza sostanziale riguarda la rappresentazione delle cartelle cliniche: nel nostro progetto ogni cartella clinica è modellata come un NFT (token ERC-721), il che consente di tracciare in modo trasparente la proprietà, i trasferimenti e i permessi di accesso. Questa scelta abilita una gestione granulare e interoperabile delle cartelle cliniche, permettendo anche l'integrazione con altri servizi Web3 e la possibilità di sfruttare le potenzialità degli NFT in termini di tracciabilità e composabilità. Patientory, invece, non utilizza NFT: i dati sono gestiti tramite account e permessi su blockchain, ma non sono rappresentati come asset unici e trasferibili. Questo comporta una minore flessibilità e interoperabilità rispetto alla nostra soluzione.

Privacy e Zero Knowledge Proof

Un altro aspetto distintivo è l'adozione delle Zero Knowledge Proof (ZKP) nel nostro sistema. Grazie a questa tecnologia, il paziente può dimostrare di possedere determinati requisiti (ad esempio, essere maggiorenne) senza mai rivelare la propria data di nascita o altri dati sensibili, né al medico né alla blockchain. Patientory, pur ponendo attenzione alla privacy, non integra ZKP: la protezione dei dati si basa su permessi e cifratura, ma non è possibile effettuare verifiche crittografiche di proprietà senza rivelazione del dato.

Gestione dei permessi

La gestione dei permessi nel nostro progetto è nativamente legata agli NFT: il paziente può concedere o revocare l'accesso a singole cartelle cliniche, anche per specifici

operatori. In Patientory, i permessi sono gestiti a livello di account e ruoli, con una granularità inferiore e senza la tracciabilità nativa offerta dagli NFT.

Architettura e blockchain privata

Dal punto di vista architettonico, Patientory adotta una blockchain privata (o di tipo consortium). Questa scelta è motivata da esigenze di compliance normativa (come GDPR e HIPAA), che impongono un controllo rigoroso sugli accessi e la possibilità di revocare permessi in modo affidabile. Una blockchain privata consente di limitare la partecipazione alla rete solo a soggetti autorizzati, garantendo maggiore privacy e controllo rispetto a una blockchain pubblica. Inoltre, permette di ottimizzare le performance e ridurre i costi di transazione, aspetti particolarmente rilevanti in ambito sanitario. Tuttavia, questa scelta comporta una minore decentralizzazione e una governance più centralizzata rispetto a una soluzione completamente pubblica.

Considerazioni finali

In sintesi, mentre Patientory rappresenta una soluzione matura e conforme alle normative per la gestione dei dati sanitari su blockchain, il nostro progetto introduce elementi di innovazione grazie all'uso degli NFT e delle Zero Knowledge Proof, offrendo maggiore privacy, tracciabilità e interoperabilità, pur dovendo affrontare sfide aggiuntive in termini di compliance e accettazione in contesti regolamentati.

5 Conclusioni

Difficoltà incontrate

Durante lo sviluppo del progetto sono emerse diverse problematiche di natura tecnica. L'integrazione tra Zero Knowledge Proof e smart contract ha richiesto una comprensione delle limitazioni imposte dalla blockchain, in particolare per quanto riguarda la gestione dei parametri pubblici e la compatibilità tra i formati delle prove generate off-chain e quelle verificate on-chain. La fase di trusted setup per zk-SNARK si è rivelata delicata e fondamentale per la sicurezza complessiva del sistema: automatizzare e validare questa fase senza introdurre errori è stato un compito necessario. Un'ulteriore sfida è stata la gestione della privacy, che ha richiesto attenzione nella progettazione dei circuiti e delle interfacce per garantire che nessun dato sensibile venisse esposto, né on-chain né off-chain. Dal punto di vista dell'esperienza utente, rendere accessibili e intuitive tecnologie avanzate come ZKP e NFT ha richiesto numerose iterazioni sull'interfaccia e sulla documentazione. Infine, l'interoperabilità tra frontend, backend, smart contract e storage decentralizzato ha imposto una progettazione modulare e una fase di test approfondita per garantire la coerenza e la robustezza dell'intero sistema.

Sviluppi futuri

Il progetto offre numerose opportunità di evoluzione. Un primo ambito riguarda l'estensione dei circuiti ZKP per coprire nuovi tipi di vincoli, come la verifica di patologie, abilitazioni o consensi, oppure la gestione di dati aggregati. Un altro aspetto di interesse è la gestione avanzata dei ruoli, con l'introduzione di ruoli più granulari e logiche di delega per una gestione degli accessi ancora più flessibile. L'integrazione con soluzioni di identità digitale decentralizzata (DID) rappresenta un ulteriore passo per rafforzare i meccanismi di autenticazione e privacy. Dal punto di vista tecnico, sarà importante lavorare sull'ottimizzazione dei circuiti e dell'infrastruttura per ridurre i costi di gas e migliorare le performance nella generazione e verifica delle prove. Infine, l'adozione in contesti sanitari reali e la raccolta di feedback da parte di utenti e operatori potranno guidare ulteriori miglioramenti in termini di usabilità e sicurezza. Un aspetto importante da sottolineare è che nel progetto non è stata posta particolare attenzione all'aspetto della sicurezza, in particolare per quanto riguarda il calcolo della data limite per la verifica dell'età. Attualmente, questa data viene calcolata direttamente dal front-end, esponendo il sistema a potenziali

manomissioni: un utente malintenzionato potrebbe alterare il calcolo e inviare allo smart contract una data limite non corretta, pur selezionando un'età che rientra nelle fasce consentite. Questo comportamento vanifica gli sforzi fatti per garantire la verifica per fasce d'età, poiché il contratto si fida di un'informazione che può essere manipolata lato client.

In una fase iniziale dell'implementazione di questo controllo, per evitare l'inferenza tramite richieste multiple, avevamo valutato di calcolare la data limite direttamente nello smart contract utilizzando il timestamp di Solidity. Tuttavia, dato che Solidity non offre un supporto nativo per la gestione delle date, l'unica opzione realmente decentralizzata e precisa sarebbe stata quella di eseguire questo calcolo tramite uno script su Chainlink Functions, mantenendo così l'architettura trustless e decentralizzata.

Per risolvere questo problema, si potrebbe quindi introdurre un oracolo che, sulla base dell'età scelta dal medico, calcoli la data limite in modo affidabile e la invii direttamente allo smart contract. In questo modo si eliminerebbe la possibilità di manomissione da parte dell'utente e si aumenterebbe la sicurezza complessiva della soluzione.

Suddivisione delle fasi e contributi

Entrambi i membri del gruppo hanno contribuito sia allo sviluppo del frontend che del backend (inclusa l'integrazione con Pinata per la gestione dei file), oltre che alla stesura della documentazione tecnica e progettuale.

Per quanto riguarda lo smart contract, Rocco si è occupato della realizzazione completa ad eccezione della funzione `accessWithProof`, che è stata sviluppata da Daniele per garantire l'integrazione tra contratto principale, Verifier e frontend.

Sul fronte Zero Knowledge Proof, Rocco ha approfondito la generazione delle prove, il trusted setup e la produzione di tutti i file necessari per integrare il circuito nel frontend. Daniele ha poi realizzato l'integrazione di questi file all'interno del frontend, utilizzando la funzione `accessWithProof` per interagire con il contratto `Verifier.sol` e completare il flusso di verifica.