CertiChain

Smart Contract System and Security Design Document

Riferimento	
Versione	1.0
Data	
Destinatario	Prof. Alfredo De Santis, Prof. Christian Esposito
Presentato da	Matteo Ercolino

Introduzione

La piattaforma **CertiChain** è un sistema decentralizzato basato su blockchain per l'emissione, la gestione e la verifica dei certificati digitali. L'obiettivo principale è garantire **autenticità**, **immutabilità** e **verificabilità pubblica** dei certificati accademici e professionali senza la necessità di un'autorità centralizzata.

L'uso della tecnologia blockchain consente di superare i limiti dei sistemi tradizionali, che spesso si basano su database centralizzati, esposti a rischi di manomissione, perdita di dati e falsificazioni. Con CertiChain, ogni certificato è rappresentato da un **NFT (Non-Fungible Token)**, che assicura l'unicità e l'integrità del certificato digitale, mentre i metadati sono archiviati in modo distribuito su **IPFS (InterPlanetary File System)**.

Obiettivo del Documento

Questo documento analizza le **scelte di progettazione** e i **design pattern** utilizzati nello sviluppo degli smart contract di CertiChain. Verranno descritti i pattern impiegati, il loro ruolo nel sistema e i vantaggi che apportano in termini di **sicurezza**, **scalabilità** ed **efficienza**.

Smart Contract

Lo sviluppo degli smart contract di CertiChain segue i principi della **programmazione modulare** e **sicura**, adottando diversi **design pattern** per garantire un'architettura scalabile, sicura ed efficiente. Di seguito vengono descritti i principali pattern utilizzati.

AccessControlManager.sol

Ruolo: Gestione dei permessi e dei ruoli all'interno del sistema.
Il contratto **AccessControlManager** è responsabile della gestione degli utenti autorizzati all'emissione di certificati. Utilizza il modulo **AccessControl** di OpenZeppelin per definire due livelli di autorizzazione:

- **Admin** (DEFAULT_ADMIN_ROLE) – Ha il permesso di assegnare o revocare ruoli agli utenti.
- **Issuer** (ISSUER_ROLE) – Può emettere e revocare certificati.

Funzioni principali:

- *addIssuer(address account)*: assegna il ruolo di Issuer a un indirizzo.
- *removeIssuer(address account)*: revoca il ruolo di Issuer da un indirizzo.
- *hasIssuerRole(address account)*: verifica se un indirizzo ha il ruolo di Issuer.

CertificateNFT.sol

Ruolo: Emissione e gestione dei certificati digitali come NFT (ERC721).
Il contratto **CertificateNFT** rappresenta i certificati sotto forma di **NFT (Non-Fungible Token**), conformi allo standard **ERC721URIStorage**. Ogni NFT ha un **token ID univoco** e un **URI IPFS** che punta ai metadati del certificato.
Funzioni principali:

- *setBaseURI(string memory newBaseURI*): imposta un nuovo URI di base per gli NFT.
- *mintCertificate(address beneficiary, uint256 tokenId, string memory tokenURI)*: crea un nuovo NFT per un beneficiario.
- *burnCertificate(uint256 tokenId)*: elimina un certificato (revoca definitiva).

CertificateRegistry.sol

Ruolo: Registro principale dei certificati, gestisce emissioni, revoche e verifiche.

Il contratto **CertificateRegistry** è il cuore del sistema, fungendo da registro decentralizzato che memorizza le informazioni essenziali dei certificati. Ogni certificato viene collegato a un ID NFT e contiene l'URI del file su IPFS.

Funzioni principali:

- *function issueCertificate(address beneficiary, bytes32 hashedSecret, string memory ipfsURI):* Emissione di un nuovo certificato NFT, registrando i dati nel mapping certificates.
- *revokeCertificate(uint256 tokenId, string memory reason)*: Revoca un certificato, aggiornandone lo stato.
- *verifyCertificate(uint256 tokenId)*: Controlla la validità di un certificato e restituisce tutti i suoi dettagli.
- *getUserCertificates(address user)*: Restituisce l'elenco dei certificati posseduti da un determinato utente.

Design Pattern

Access Restriction Pattern

Contratti coinvolti: *AccessControlManager.sol, CertificateNFT.sol, CertificateRegistry.sol*.

Descrizione:

La gestione degli accessi è un aspetto critico per prevenire modifiche non autorizzate ai certificati digitali.

- *AccessControlManager* utilizza il modulo *AccessControl* di OpenZeppelin per definire due livelli di accesso:
  1. **Admin** (DEFAULT_ADMIN_ROLE) – Può gestire gli issuer.
  2. **Issuer** (ISSUER_ROLE) – Autorizzato a emettere certificati.
- *CertificateNFT* utilizza il modulo *Ownable* per limitare la creazione e la modifica dei certificati al proprietario del contratto.
- CertificateRegistry impone che solo gli utenti con il ruolo di issuer possano emettere o revocare certificati.

Vantaggi:

- Impedisce la modifica non autorizzata dei certificati.
- Separa i privilegi, riducendo il rischio di compromissione.
- Permette una gestione gerarchica degli utenti.

Factory Creation Pattern

Contratti coinvolti: *CertificateRegistry.sol*.

Descrizione:

La piattaforma segue un approccio **factory-based**, in cui i certificati digitali vengono **generati dinamicamente** tramite il contratto *CertificateRegistry*.

- Ogni volta che viene emesso un certificato, viene creato un **nuovo token NFT** con un identificatore univoco.
- Il mapping *certificates* collega ogni ID NFT ai dettagli del certificato.
- La funzione *issueCertificate* gestisce la creazione e associazione del certificato al beneficiario.

Vantaggi:
- Centralizza il processo di emissione dei certificati, garantendo coerenza.
- Automatizza la gestione degli NFT, riducendo errori manuali.
- Permette un'estensibilità futura, supportando nuovi tipi di certificati.

Registry Pattern

Contratti coinvolti: *CertificateRegistry.sol*.

Descrizione:

Il contratto *CertificateRegistry* agisce come registro centrale per la memorizzazione delle informazioni sui certificati.
- L'ID di ogni certificato è associato a una struttura *CertificateInfo* che contiene dettagli come il nome dell'istituzione, il titolo, la data di rilascio e lo stato di revoca.
- La funzione *getUserCertificates* consente ai beneficiari di visualizzare i certificati a loro associati.

Vantaggi:
- Permette una **facile consultazione** dei certificati.
- Garantisce la **trasparenza** e la tracciabilità delle certificazioni.
- Ottimizza la gestione delle revoche senza eliminare i dati.

Checks-Effects-Interactions Pattern

Contratti coinvolti: *CertificateRegistry.sol*.

Descrizione:

Questo pattern è fondamentale per la sicurezza degli smart contract e viene utilizzato per prevenire attacchi di reentrancy, in cui un contratto malevolo potrebbe eseguire più volte una funzione prima che lo stato venga aggiornato. Il principio base è che l'ordine delle operazioni all'interno di una funzione dovrebbe sempre seguire questi tre passaggi:
1. Checks (Controlli) – Verifica delle condizioni necessarie prima di eseguire un'azione (es. controllare permessi o validità dei dati).
2. Effects (Effetti) – Aggiornamento dello stato del contratto prima di qualsiasi chiamata esterna.
3. Interactions (Interazioni) – Solo dopo aver aggiornato lo stato, si interagisce con contratti esterni o si trasferiscono fondi.

CertiChain adotta questo pattern in più punti per garantire la sicurezza delle operazioni.

Vantaggi:
- Previene attacchi di reentrancy, assicurando che lo stato sia aggiornato prima delle chiamate esterne.
- Migliora la prevedibilità e la sicurezza delle transazioni.
- Riduce il rischio di errori logici dovuti a interazioni tra più smart contract.

Guard Check Pattern

Contratti coinvolti: *CertificateRegistry.sol*.

Descrizione:
In CertiChain viene implementato attraverso require() ed eccezioni try-catch per prevenire errori bloccanti. Ad esempio, la funzione *verifyCertificate* utilizza un **blocco** *try-catch* per gestire le eccezioni quando si verifica il proprietario di un NFT.
- Se l'NFT non esiste o è stato eliminato, la funzione restituisce **valori di fallback**, evitando crash del sistema.

Vantaggi:
- Migliora l'affidabilità del contratto.
- Evita errori che potrebbero rendere inutilizzabili alcune funzioni.
- Rende il codice più resiliente a scenari imprevisti.

Event Logging Pattern
Contratti coinvolti: *CertificateRegistry.sol*
Descrizione:
Il contratto CertificateRegistry utilizza eventi Solidity per tracciare le operazioni chiave.
- *CertificateIssued(tokenId, issuer, beneficiary)*: notifica l'emissione di un certificato.
- *CertificateRevoked(tokenId, issuer, reason)*: registra la revoca di un certificato.

Vantaggi:
- Fornisce **trasparenza** sulle operazioni della piattaforma.
- Facilita il **monitoraggio off-chain** da parte di applicazioni esterne.
- Permette di costruire una **cronologia verificabile** delle certificazioni.

Interazioni tra i Contratti
L'architettura di CertiChain si basa sulla collaborazione tra i tre smart contract. Il flusso di operazioni è il seguente:
1. **Emissione di un certificato**:
   L'utente con ruolo di Issuer chiama issueCertificate() su CertificateRegistry.sol, che:
   - Registra i dati del certificato nel mapping.
   - Chiama mintCertificate() su CertificateNFT.sol, creando un nuovo NFT.
2. **Verifica di un certificato**:
   Chiunque può chiamare verifyCertificate() su CertificateRegistry.sol per controllare la validità di un certificato.
3. **Revoca di un certificato**:
   Se un certificato deve essere invalidato, un Issuer chiama revokeCertificate(), che aggiorna lo stato on-chain e mantiene la cronologia della revoca.

Scelte Architetturali
L'architettura di CertiChain è stata progettata per garantire sicurezza, scalabilità ed efficienza nella gestione dei certificati digitali. L'obiettivo principale è creare un sistema decentralizzato e affidabile, in grado di eliminare gli intermediari e ridurre i rischi di contraffazione o perdita dei dati.

Centralizzazione vs. Decentralizzazione

Uno dei problemi principali dei sistemi tradizionali di certificazione è la dipendenza da database centralizzati, vulnerabili a manomissioni, attacchi informatici e perdita di dati. Per superare questi limiti, CertiChain utilizza una blockchain pubblica come Ethereum, che assicura trasparenza e immutabilità. In questo modo, i certificati possono essere verificati direttamente sulla blockchain senza la necessità di contattare l'ente certificatore, riducendo i tempi di attesa e i costi amministrativi.

L'uso della blockchain, tuttavia, presenta delle sfide. Le transazioni su Ethereum hanno un costo in gas, il che può rappresentare un ostacolo in termini di scalabilità.

Scelta della Blockchain Ethereum

Ethereum è stato scelto per la sua affidabilità e compatibilità con gli smart contract Solidity. La piattaforma sfrutta lo standard **ERC721** per rappresentare i certificati come NFT, garantendo che ogni certificato sia univoco e non modificabile. Un altro vantaggio di Ethereum è la sua interoperabilità con wallet come **MetaMask**, che permette agli utenti di autenticarsi e gestire i propri certificati in modo sicuro.

Tuttavia, Ethereum non è esente da limiti. Il costo delle transazioni può variare in base alla congestione della rete, e la conferma delle operazioni può richiedere tempo. Per affrontare questi problemi, la piattaforma è progettata in modo da poter essere facilmente adattata a soluzioni **Layer 2**, come **Polygon**, che offrono commissioni più basse e transazioni più rapide.

Storage Off-Chain con IPFS

Salvare tutti i dati on-chain sarebbe troppo costoso e inefficiente. Per questo motivo, CertiChain utilizza **IPFS (InterPlanetary File System)** per l'archiviazione dei certificati. Quando un certificato viene emesso, i suoi metadati vengono caricati su IPFS e l'URI del file viene salvato all'interno dello smart contract. Questo approccio garantisce che il certificato rimanga accessibile in qualsiasi momento senza la necessità di mantenere un'infrastruttura centralizzata.

Struttura degli Smart Contract

- L'architettura è suddivisa in tre smart contract principali, ognuno con un ruolo specifico. *AccessControlManager* gestisce le autorizzazioni, distinguendo tra amministratori e issuer.
- *CertificateNFT* è il contratto che rappresenta i certificati sotto forma di NFT ERC721 e permette di associare metadati ai certificati.
- *CertificateRegistry* funge da registro principale e gestisce l'emissione, la revoca e la verifica dei certificati.

Questa suddivisione consente una maggiore modularità del sistema. Separando la gestione dei permessi dalla registrazione dei certificati, il codice risulta più sicuro e manutenibile. Inoltre, ogni contratto può essere aggiornato o sostituito senza impattare il resto della piattaforma, rendendo il sistema più flessibile.

Ottimizzazione dei Costi di Transazione

L'uso della blockchain comporta costi legati al consumo di gas per ogni operazione. Per ridurre questi costi, CertiChain adotta diverse strategie di ottimizzazione. Ad esempio, l'uso di eventi Solidity permette di registrare informazioni importanti senza memorizzarle direttamente on-chain. Inoltre, minimizzare le scritture sulla blockchain e spostare la gestione dei dati su IPFS consente di ridurre il numero di operazioni costose.

Sicurezza e Mitigazione dei Rischi

La sicurezza è un elemento chiave nella progettazione degli smart contract di CertiChain, poiché la piattaforma gestisce certificati digitali che devono essere immutabili, verificabili e resistenti ad attacchi informatici. Di seguito analizziamo le principali minacce e le contromisure adottate per garantire un sistema affidabile.

Protezione dell'Accesso e Gestione delle Autorizzazioni

Per evitare accessi non autorizzati, la piattaforma utilizza un sistema di **controllo degli accessi basato su ruoli**, implementato attraverso il contratto *AccessControlManager*. Gli amministratori hanno il controllo sui ruoli, mentre solo gli utenti con il ruolo di Issuer possono emettere certificati. In *CertificateNFT*, il pattern **Ownable** di OpenZeppelin assicura che solo il proprietario del contratto possa eseguire operazioni critiche.

Questa gestione delle autorizzazioni impedisce a utenti non autorizzati di manipolare i certificati e limita il rischio di azioni malevole.

Prevenzione di Attacchi ai Contratti Smart

Gli smart contract Solidity sono esposti a diverse tipologie di attacchi. CertiChain ha adottato misure per mitigare i più comuni rischi di sicurezza:

- **Reentrancy Attack**:
  Per evitare attacchi di rientranza, che potrebbero portare a prelievi ripetuti non autorizzati, CertiChain non effettua chiamate esterne durante transazioni critiche. Inoltre, l'uso del pattern **Checks-Effects-Interactions** garantisce che tutte le modifiche allo stato siano eseguite prima di interagire con altri contratti.
- **Front-running**:
  Poiché le transazioni su Ethereum sono pubbliche prima della loro esecuzione, un attore malevolo potrebbe tentare di sfruttare questa caratteristica per ottenere vantaggi indebiti. Per mitigare il problema, CertiChain utilizza meccanismi di **hash commitment**, riducendo la prevedibilità di alcune operazioni.
- **Attacchi di Spoofing e Sybil Attack**:
  L'autenticazione degli utenti avviene tramite **wallet Ethereum**, il che garantisce che solo gli indirizzi autorizzati possano eseguire operazioni critiche. Inoltre, il sistema di ruoli impedisce la creazione arbitraria di certificati senza autorizzazione.

Protezione da Revoche e Certificati Contraffatti

Una volta emesso, un certificato può essere revocato solo dall'ente certificatore. Per prevenire abusi, il sistema registra l'evento di revoca e aggiorna lo stato del certificato nel

mapping certificates. In fase di verifica, chiunque può controllare se un certificato è stato revocato, evitando l'uso di credenziali non valide. Inoltre, grazie all'architettura decentralizzata, non è possibile creare certificati falsi senza passare per il registro ufficiale on-chain.

Logging degli Eventi e Monitoraggio

Gli smart contract registrano eventi chiave come:

- *CertificateIssued*, che traccia l'emissione di un certificato.
- *CertificateRevoked*, che registra le revoche con la relativa motivazione.

Questi eventi possono essere utilizzati per audit automatici e per integrare sistemi di monitoraggio off-chain.

Sviluppi e Miglioramenti Futuri

CertiChain è progettato per essere una piattaforma scalabile e flessibile, ma ci sono diversi miglioramenti che potrebbero essere implementati in futuro per aumentarne l'efficienza, la sicurezza e l'adozione. Ad esempio:

- Integrazione con Layer 2 per la Riduzione dei Costi.
- Decentralized Identity (DID) e Integrazione con Standard Verifiable Credentials.
- Adozione di Smart Contract Upgradabili.
- Certificati Programmabili e Interoperabilità con DeFi.