



UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea triennale in Informatica

Il ruolo degli NFT nella protezione dei beni

Relatore:

Dott. Andrea Visconti

Candidato:

Daniele Reccagni, mat. 975771

Anno accademico 2022/2023

Indice

1	Introduzione	4
2	Concetti preliminari	6
2.1	Funzioni hash	6
2.2	Hash pointer	8
2.3	Blockchain	8
2.4	Smart contract	14
2.5	NFT	15
2.6	Firma digitale	17
2.7	Crypto wallet	18
2.8	Blockchain explorer	21
3	Scelta della blockchain	22
3.1	Compatibilità con EVM	22
3.2	Ecosostenibilità	23
3.3	Identificazione della blockchain	27
4	Scelta dell'approccio	28
4.1	Approccio 1:N	29
4.2	Approccio 1:1	29
4.3	Definizione dell'approccio corretto	31
5	Il metodo	32
5.1	Documento di specifica per il cliente	32
5.2	Operazioni preliminari	35
5.3	Creazione di uno smart contract	36
5.4	Verifica dello smart contract	41
5.5	Creazione degli NFT	43
5.6	Trasferimento degli NFT al cliente	47

6	Idee di utilizzo	51
6.1	Come associare l’NFT all’oggetto fisico	51
6.2	Come presentare l’NFT	52
6.3	Come vendere il bene	53
7	Conclusioni	54
8	Ringraziamenti	55
	Bibliografia	56

Capitolo 1

Introduzione

La nascita e l'evoluzione del Web 3.0 stanno portando il controllo e la proprietà delle informazioni in transito su Internet nelle mani di chi effettivamente lo utilizza: l'approccio decentralizzato di gestione dei dati sta creando un sistema più equo e autonomo per gli utenti di Internet, rimuovendo gradualmente il necessario controllo esterno delle grandi aziende caratteristico dell'approccio centralizzato del Web 2.0. Termini come criptovalute, blockchain, finanza decentralizzata, token, NFT, meta-verso e app decentralizzate sono concetti oramai diffusi, consolidati e ben noti ai più, e ci stiamo rendendo conto di quanto questi strumenti, se utilizzati in maniera intelligente, siano veramente molto potenti e possano dare una grande mano nell'evoluzione della società contemporanea.

La domanda che sorge quasi spontanea a questo punto è la seguente: è possibile utilizzare le entità sopra citate, o almeno alcune tra di esse, per risolvere problemi noti in modo semplice, veloce, economico e affidabile? La risposta è un convinto "sì" e la tecnologia può venire in nostro aiuto.

In questo lavoro di tesi, infatti, l'obiettivo è stato di progettare e realizzare una soluzione concreta a uno di questi problemi sfruttando le tecniche crittografiche apprese durante il mio percorso di studi. Più precisamente, abbiamo ideato un sistema per garantire protezione ad un qualsiasi bene materiale, come ad esempio una scultura, una casa, una giacca di lusso o una bottiglia di un liquore pregiato.

Cosa significa "garantire protezione" ad uno di questi oggetti? Immaginiamo una generica operazione di vendita tra un venditore ed un acquirente. Il nostro obiettivo è assicurare che in questo processo siano rispettate tre proprietà fondamentali:

- Il venditore e l'acquirente siano realmente chi dicono di essere e non dei truffatori;
- Il venditore posseda veramente il bene che sta provando a vendere all'acquirente;
- Il bene venduto dal venditore all'acquirente sia autentico e non un falso.

Se ci pensiamo bene, queste sono proprio le garanzie che dà un notaio durante un atto di vendita. Ebbene, l'obiettivo è quindi di automatizzare il lavoro di un notaio utilizzando alcuni dei potenti strumenti del Web 3.0.

Si noti che, dal momento che il Web 3.0 è in costante crescita, alcune delle scelte/tecnologie descritte in questa tesi potrebbero risultare "superate" a stretto giro a causa delle continue evoluzioni tecnologiche.

Concetti preliminari

Elenchiamo e analizziamo ora i più importanti strumenti che sono stati utilizzati durante la creazione del sistema di protezione, scendendo nei dettagli delle loro caratteristiche e del loro funzionamento.

2.1 Funzioni hash

Le funzioni hash sono delle funzioni crittografiche facilmente computabili che ricevono in input una stringa di dimensione qualsiasi e restituiscono in output una stringa di dimensione fissa. Esse posseggono tre proprietà principali:

- **Collision-free**

Non si è in grado di trovare X e Y tali che $H(X) = H(Y)$ e $X \neq Y$. In realtà queste collisioni esistono, ma dei regular users non sono in grado di trovarle.

Grazie a questa proprietà, possiamo assumere $H(X) = H(Y) \rightarrow X = Y$ e, di conseguenza, possiamo salvare solo $H(X)$ e $H(Y)$ che, oltre ad occupare molto meno spazio, permettono di comparare dati usando molti meno bit rispetto ai dati in chiaro.

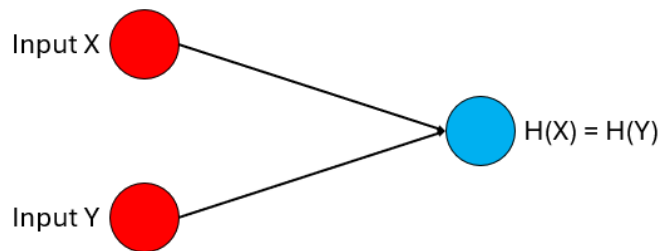


Figura 1: Proprietà collision-free [1]

- **Preimage resistant**

Le funzioni hash sono one-way function: dato $H(X)$ è computazionalmente impossibile riuscire a risalire ad X . Grazie a questa proprietà, l'unico modo per trovare X sarebbe un attacco bruteforce.

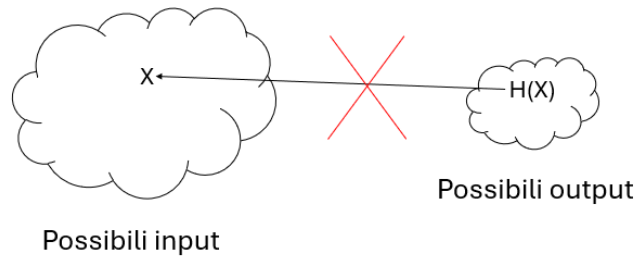


Figura 2: Proprietà preimage resistant [1]

- **Second preimage resistant**

Conoscendo X , $H(X)$ e $H(Y)$ non si è in grado di risalire ad Y tale che $H(X) = H(Y)$ e $X \neq Y$.

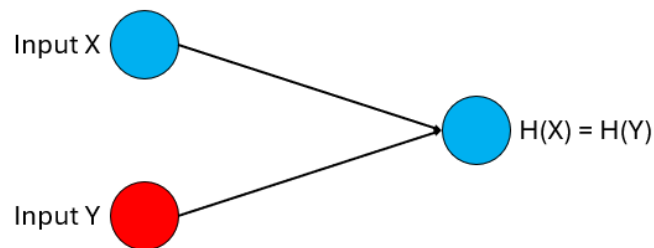


Figura 3: Proprietà second preimage resistant [1]

Queste particolari funzioni sono usate, ad esempio, per provare l'integrità dei dati (confrontando l'hash dei dati in momenti diversi) o in ambito blockchain per mezzo di hash pointers. Proprio quest'ultimo è l'utilizzo sul quale ci concentreremo. [1]

2.2 Hash pointer

Un hash pointer è un'evoluzione del concetto di puntatore classico.

Se i puntatori normali memorizzano unicamente l'indirizzo di memoria dei dati a cui fanno riferimento e permettono di accedere facilmente ad essi, un hash pointer memorizza anche l'hash crittografico dei dati permettendo di verificare anche la loro integrità. Utilizzando questi hash pointers è possibile ottenere importantissimi concetti e strutture dati come le blockchain e gli alberi di Merkle. [2]

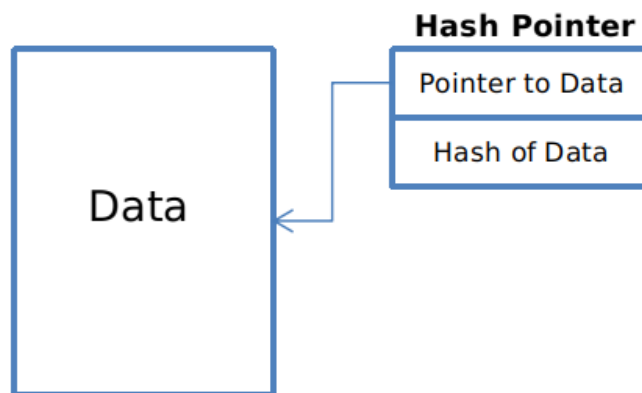


Figura 4: Hash pointer [3]

2.3 Blockchain

Una blockchain è un enorme registro globale distribuito su cui vengono salvati i dati di una serie di operazioni: le transazioni. Essa prende il nome dal modo in cui memorizza questi dati, ossia creando una catena di blocchi.

Un blocco è un insieme di dati di transazione che ha raggiunto una certa dimensione considerevole. Quando questo avviene, tutte le transazioni a cui fa riferimento vengono convalidate e quindi archiviate in modo permanente all'interno della blockchain. I blocchi appena citati sono concatenati tra di loro formando il cosiddetto libro mastro immutabile. [4]

2.3.1 Layers

Esistono diversi livelli di blockchain, ognuno dei quali definisce delle caratteristiche importanti della chain stessa:

- **Blockchain di livello 1**

Le blockchain di livello 1 sono le principali: eseguono e convalidano le transazioni da sole, senza assistenza, e hanno le proprie criptovalute associate.

Alcuni esempi di queste blockchain sono Ethereum, Solana e Avalanche.

- **Blockchain di livello 2**

Le blockchain di livello 2 sono utili per risolvere i problemi di scalabilità delle blockchain di livello 1. Esse elaborano le transazioni internamente e memorizzano solo un riepilogo delle azioni svolte sulla chain di livello 1.

Grazie a questa proprietà, le richieste alla blockchain principale sono ridotte di molto, migliorando la scalabilità e abbassando i costi dei gas fees, delle "tasse" da pagare per poter utilizzare la blockchain.

Optimism e Arbitrum sono degli esempi di questo tipo di blockchain.

- **Blockchain di livello 3**

Le blockchain di livello 3 aggiungono ulteriore scalabilità e interoperabilità tra diverse blockchain. Esse sono particolarmente utili per eseguire applicazioni decentralizzate.

Alcuni esempi sono Polkadot e Cosmos.

- **Sidechain**

Le sidechain sono molto simili alle blockchain di livello 2: sono anche esse blockchain separate e collegate ad una chain di livello 1, ma sono molto più indipendenti e spesso hanno una criptovaluta personale.

Un classico esempio di sidechain è Polygon, una sidechain di Ethereum: gli utenti ottengono un'esperienza simile all'utilizzo di Ethereum, ma con gas fees più bassi e tempi di transazione più rapidi (vedi la Sezione 3.3). [4]

2.3.2 Meccanismi di consenso

Un'altra caratteristica importante delle blockchain riguarda i meccanismi di consenso, cioè il modo in cui le transazioni vengono convalidate, ossia registrate in modo permanente nella blockchain. Esistono due meccanismi principali:

- **Proof-of-Stake**

Le blockchain che utilizzano il metodo Proof-of-Stake verificano le transazioni usando delle persone dette validators.

Quando un nuovo blocco di transazioni deve essere aggiunto alla blockchain, viene selezionato un validator in modo casuale. Egli creerà concretamente il blocco di transazioni e lo porrà in esame ad un comitato di altri validators, che voteranno se quel blocco è valido e quindi può essere memorizzato nella chain. Le blockchain Proof-of-Stake sono molto più veloci, meno costose e più rispettose dell'ambiente rispetto a quelle Proof-of-Work, ma sono meno sicure.

Alcuni esempi sono Ethereum (post-Merge) e Polygon.

- **Proof-of-Work**

Le blockchain che utilizzano il metodo Proof-of-Work verificano le transazioni usando delle persone dette miners.

Quando un nuovo blocco di transazioni deve essere aggiunto alla blockchain, un gruppo di miners concorrerà per trovare un determinato numero, detto nonce, tale per cui una funzione di hash applicata sul blocco di transazioni e sul numero restituisca un altro numero con una certa quantità di zeri in testa.

Ogni qualvolta un miner comunichi pubblicamente di aver trovato il nonce corretto affinché si verifichi la condizione di cui sopra, gli altri miners si occuperanno di verificare la validità di ciò e, in caso tutti siano d'accordo sulla correttezza del nonce, allora il blocco sarà considerato valido e registrato in blockchain e il miner che ha trovato il numero verrà ricompensato con dei "contributi involontari" degli utenti che svolgono le transazioni.

Le blockchain Proof-of-Work sono più lente, più costose e meno rispettose dell'ambiente rispetto a quelle Proof-of-Stake, ma sono più sicure.

Un esempio è Bitcoin. [4]

2.3.3 Implementazione

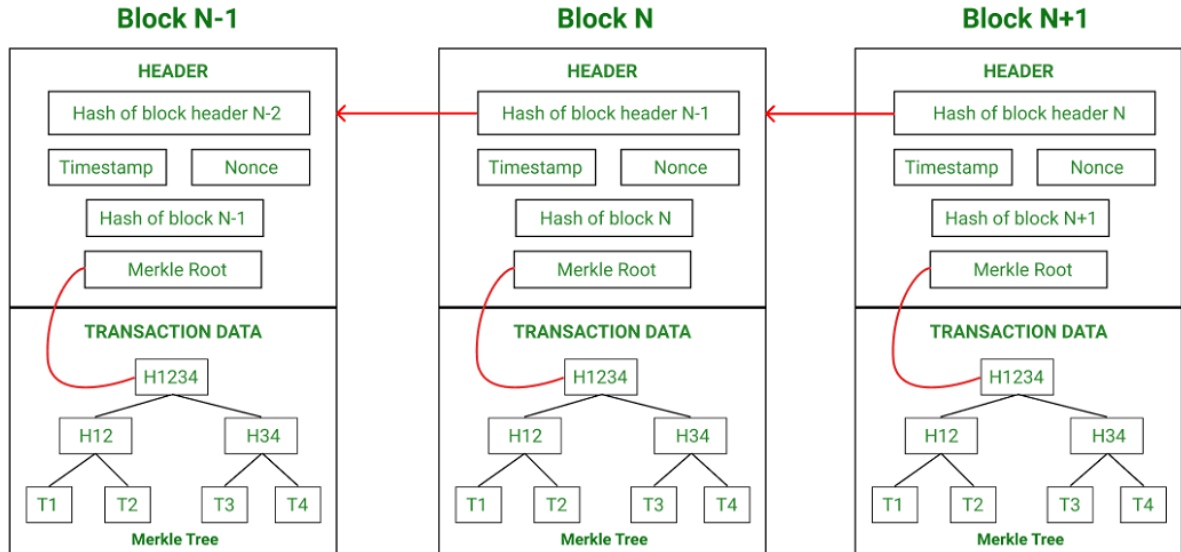


Figura 5: Struttura di una blockchain [2]

Possiamo notare che una blockchain è una linked list di hash pointers in cui:

- Ogni blocco è un nodo della lista;
- Gli hash pointers collegano il blocco N con il blocco N-1;
- Il contenuto di ogni nodo è composto dall'header e dai dati delle transazioni.

Analizziamo ora i due componenti fondamentali di ogni blocco.

- **Header**

L'intestazione di ogni blocco è composta da diverse informazioni. In particolare:

- Hash dell'header del blocco precedente e puntatore;
- Hash del blocco corrente;
- Timestamp e nonce del blocco corrente;
- Radice del Merkle Tree delle transazioni del blocco corrente.

- **Dati delle transazioni (Merkle Tree)**

I dati effettivi delle transazioni sono memorizzati in un Merkle Tree, un particolare albero binario che prende il nome dal suo creatore, Ralph Merkle.

La necessità di utilizzare un Merkle Tree anziché una semplice linked list di hash pointers è dovuta al fatto che usando una lista sarebbe molto oneroso svolgere alcune operazioni sulle transazioni, come ad esempio verificare se una transazione appartiene ad un determinato blocco. Infatti, oltre a dover scorrere i blocchi uno ad uno, dovremmo passare anche la linked list delle transazioni per ogni blocco!

L'albero di Merkle, di cui ne esiste uno per blocco, è usato sia per capire se le transazioni di un blocco sono tutte integre (l'errore, in caso ci fosse, si propaga fino alla radice) sia per provare l'appartenenza di una transazione ad un blocco in modo efficiente in termini di tempo.

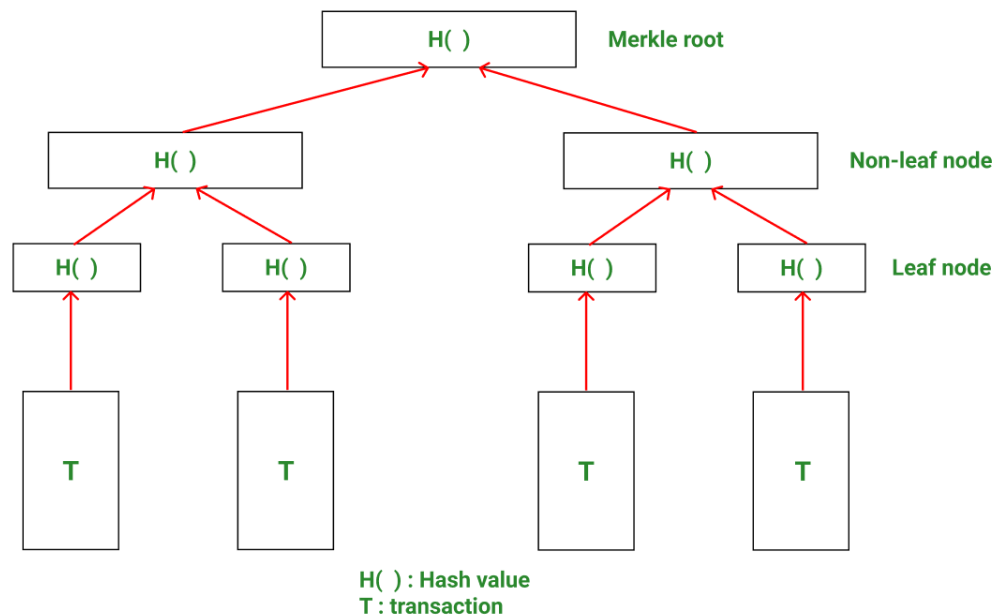


Figura 6: Struttura di un Merkle Tree [2]

Osservando la Figura 6 nella pagina precedente, possiamo notare che i Merkle Tree sono costruiti a partire dal basso.

Inoltre, esistono diversi tipi di nodi:

- **Nodo radice**

La radice dell'albero di Merkle è memorizzata nell'intestazione del blocco.

- **Nodi non foglia**

I nodi non foglia, detti anche nodi intermedi, contengono il valore hash dei propri due nodi figli.

- **Nodi foglia**

I nodi foglia contengono l'hash delle transazioni.

Le **transazioni** non sono considerate parte dell'albero.

Infine, un Merkle Tree è di natura binaria, ossia il numero di nodi foglia deve essere pari affinché l'albero di Merkle sia costruito correttamente. Nel caso in cui ci sia un numero dispari di transazioni, viene semplicemente duplicato l'ultimo hash. [2]

2.4 Smart contract

Gli smart contracts sono ciò che automatizza effettivamente il lavoro di un notaio. Essi permettono di valutare il successo o il fallimento di una transazione attraverso un codice pre-programmato caricato ed eseguito su una blockchain e basato sui parametri della transazione concordata tra le parti, come il costo degli articoli, dove viene inviato il pagamento e chi autorizza la transazione.

I contratti intelligenti sono necessari per permettere la creazione e il funzionamento di molti componenti del Web 3.0, come gli NFT, le applicazioni decentralizzate e i protocolli di finanza decentralizzata. Ad esempio, uno smart contract è in grado di tenere traccia dei registri di proprietà di un NFT, consentendo di verificare e trasferire facilmente il token da un proprietario ad un altro senza la necessità di un intermediario di terze parti.

Vediamo ora quali sono i vantaggi di utilizzare gli smart contracts:

- **Disponibilità pubblica e sicurezza**

Gli smart contracts sono registrati sulla blockchain e, quindi, sono pubblicamente disponibili a chiunque possa accedere ad essa. Inoltre, dato che ereditano anche la natura open e decentralizzata della blockchain, possono essere facilmente autenticati.

- **Velocità ed efficienza**

Dato che gli smart contracts hanno in genere uno scopo specifico possono essere programmati guardando l'efficienza. Le transazioni relative agli smart contracts, quindi, possono essere elaborate rapidamente sulla blockchain.

- **Immutabilità**

La cronologia di uno smart contract non può essere cancellata o eliminata. [5]

2.5 NFT

Un NFT (token non fungibile) è un oggetto digitale unico e autentico memorizzato su una blockchain.

Essi sono creati basandosi su uno smart contract caricato in precedenza sulla blockchain che ha il compito di memorizzare tutta la cronologia delle operazioni che si possono svolgere su un NFT, come la creazione o un passaggio di proprietà. [6]

2.5.1 FT vs NFT

Qual è la differenza tra un FT (token fungibile) e un NFT (token non fungibile)?

Quando un oggetto è fungibile significa che è intercambiabile con un altro oggetto dello stesso tipo. Un esempio molto semplice è una moneta da 1 Euro: se la si scambiasse con qualcuno, entrambi avrebbero ancora 1 Euro.

Non fungibile, invece, significa che sia l'oggetto sia il suo valore sono assolutamente unici. Un esempio sono due auto della stessa marca e modello: esse, infatti, potrebbero avere valori diversi in base al numero di chilometri percorsi, agli incidenti o al fatto che in precedenza fossero di proprietà di una celebrità. [6]

2.5.2 Autenticità

I fatti che tutte le operazioni che si possono svolgere su un NFT sono registrate sulla blockchain e che la chain stessa è accessibile da chiunque sono a tutti gli effetti un certificato di autenticità dell'NFT.

Di norma, infatti, quando si acquista un'opera d'arte o un oggetto da collezione viene fornito anche un certificato di autenticità cartaceo. Esso, però, è spesso perso o distrutto, di fatto rendendo il sistema di garanzia dell'autenticità molto debole.

Le blockchain, invece, offrono una soluzione semplice ma molto più sicura. [6]

2.5.3 Utilizzi

Gli NFT possono essere usati per rappresentare oggetti completamente digitali ed essere acquistati a fini collezionistici, ma possono anche essere creati a partire da oggetti fisici realmente esistenti nonché venire associati ad essi: in questo caso prendono il nome di NFT phygital (physical + digital).

Gli NFT phygital hanno davvero molti vantaggi e saranno il punto cardine del sistema che andremo a sviluppare. [6]

2.6 Firma digitale

La firma digitale è un metodo che permette di dimostrare importanti proprietà come l'autenticazione del mittente e del destinatario, l'integrità dei dati e la non ripudiabilità di un'operazione. Essa si basa sugli algoritmi di cifratura asimmetrica o a chiave pubblica.

Vediamo un paio di esempi di alcune firme digitali e spieghiamone il funzionamento: supponiamo che un mittente Alice voglia firmare digitalmente un messaggio prima di inviarlo ad un destinatario Bob.

2.6.1 RSA Digital Signature

1. Alice genera le seguenti informazioni:

- p e q numeri primi
- $n = p * q$
- e con $MCD(e, \phi(n)) = 1$, con $\phi(n) =$ numero di coprimi con n
- d con $d * e \equiv 1 \text{ MOD } \phi(n)$

(e, n) formano la chiave pubblica di Alice.

(p, q, d) formano la chiave privata di Alice.

2. Alice pone la propria firma digitale su un messaggio m

$$y \equiv m^d \text{ MOD } n$$

3. Alice spedisce (m, y) a Bob

4. Bob deve autenticare il messaggio. Quindi calcola $z \equiv y^e \text{ MOD } n$

Se $z \equiv m \text{ MOD } n$, allora la firma è valida e le proprietà citate in precedenza sono garantite. [7]

2.6.2 ElGamal Digital Signature

1. Alice genera le seguenti informazioni:

- p numero primo
- α radice primitiva di p
- a numero casuale tale che $a < p - 1$
- $\beta \equiv \alpha^a \text{ MOD } p$

(p, α, β) formano la chiave pubblica di Alice.

a è la chiave privata di Alice.

2. Alice pone la firma digitale seguendo questo metodo

- Genera k , un numero segreto tale che $MCD(k, p - 1) = 1$
- Calcola r ed s
$$r \equiv \alpha^k \text{ MOD } p$$
$$s \equiv k^{-1}(m - a * r) \text{ MOD } p - 1$$

3. Alice spedisce (m, r, s) a Bob

4. Bob deve autenticare il messaggio. Quindi calcola

- $v1 \equiv \beta^r * r^s \text{ MOD } p$
- $v2 \equiv \alpha^m \text{ MOD } p$

Se $v1 \equiv v2 \text{ MOD } p$, allora la firma è valida e le proprietà citate in precedenza sono garantite. [7]

2.7 Crypto wallet

Un crypto wallet è uno strumento virtuale che aiuta ad acquistare, vendere e conservare le criptovalute e gli NFT. In particolare, fornisce un indirizzo blockchain (chiave pubblica) e una chiave privata.

Le criptovalute e gli NFT, inoltre, non sono conservati sul crypto wallet, ma risiedono sempre e solo sulla blockchain: il wallet permette solo di controllarli e gestirli. [8]

2.7.1 Tipi di crypto wallet

Esistono due tipi principali di crypto wallet:

- **Custodial o hosted**

I portafogli sono gestiti da una società terza. È come se la chiave privata del crypto wallet fosse in una struttura sicura che ha il compito di verificare l'identità di chi richiede la chiave prima di fornirla.

Essi richiedono meno responsabilità, ma c'è un rischio che la propria chiave privata venga rubata in caso di attacchi alla struttura della terza parte.

- **Non custodial o self-custodied**

I portafogli non sono gestiti da una società terza. È come se la chiave privata del crypto wallet fosse custodita in casa.

Essi forniscono il pieno controllo sulla propria chiave, ma richiedono una grande attenzione da parte dell'utente, che potrebbe perdere la chiave accidentalmente.

I portafogli non custodial si dividono ulteriormente in altri due tipi:

- **Portafogli software o hot wallet**

È un programma che risiede sul proprio computer, dispositivo mobile o sul browser Internet.

Questo rende i portafogli software un'ottima opzione per acquistare, vendere e trasferire NFT e criptovalute in modo rapido e conveniente.

- **Portafogli hardware o cold wallet**

È un dispositivo fisico che potrebbe essere necessario collegare al computer per poter essere utilizzato.

Dato che non è sempre connesso al computer, al dispositivo mobile o al browser, è un'ottima opzione per l'archiviazione sicura a lungo termine, ma è meno conveniente per le transazioni veloci o frequenti.

Questi wallet, infine, richiedono di impostare una password o un PIN per accedere. Essa è diversa dalla seed phrase, ossia la frase utilizzata per accedere al wallet in caso di smarrimento della password. [8]

2.7.2 Funzionamento

Come già detto in precedenza, i crypto wallet forniscono sia un indirizzo blockchain (chiave pubblica) sia una chiave privata. Queste due informazioni sono utilizzate per applicare la firma digitale alle transazioni che saranno caricate sulla blockchain.

La **chiave privata** ha il compito di dimostrare la proprietà della criptovaluta o dell’NFT archiviato in uno specifico indirizzo blockchain pubblico.

Essa è una stringa di caratteri (in genere 64) generata in modo casuale. È tenuta segreta e serve per controllare il wallet e tutto ciò che è memorizzato presso il relativo indirizzo pubblico.

L’**indirizzo pubblico** è una lunga stringa di caratteri che identifica le operazioni di ogni utente sulla blockchain, nonché l’informazione conosciuta da tutti gli altri utenti se volessero interagire con noi ad esempio per scambi, acquisti o vendite di criptovalute ed NFT.

Sulla blockchain di Ethereum e su quelle EVM-compatibili (vedi Sezione 3.1) si tratta di una serie di 42 caratteri che iniziano con "0x" come prefisso.

Si utilizzano quindi degli algoritmi per apporre la firma digitale sulla transazione.

Il funzionamento è molto semplice: l’utente firma digitalmente la transazione con la propria chiave privata ed essa sarà verificata utilizzando l’indirizzo pubblico dell’utente, ossia la sua chiave pubblica. [8]

2.8 Blockchain explorer

I blockchain explorer sono dei particolari siti che consentono agli utenti di cercare, confermare e convalidare facilmente le transazioni che vengono memorizzate sulla blockchain a cui fanno riferimento. Questi strumenti possono essere utilizzati da chiunque, dagli sviluppatori ai proprietari degli NFT.

Andiamo nel dettaglio dei servizi offerti da questi explorer:

- **Funzionalità di ricerca:** i blockchain explorer consentono agli utenti di cercare indirizzi sulla blockchain. Essi possono mostrare, ad esempio, la cronologia delle transazioni, i saldi dei token e le interazioni con gli smart contracts. Analogamente ai motori di ricerca, è presente una barra in cui gli utenti possono digitare la transazione o le chiavi hash che stanno cercando.
- **Monitoraggio delle transazioni:** i blockchain explorer consentono agli utenti di visualizzare e tracciare le transazioni in tempo reale. Gli utenti possono vedere dettagli come l'hash della transazione, gli indirizzi del mittente e del destinatario, i gas fees e lo stato della transazione.
- **Analisi degli smart contracts:** i blockchain explorer consentono agli utenti di visualizzare il codice sorgente, il bytecode e la cronologia delle transazioni degli smart contracts, nonché permettere una serie di operazioni come il trasferimento di proprietà degli NFT.
- **Monitoraggio dei token:** i blockchain explorer consentono agli utenti di controllare la situazione dei token, tra cui anche chi li detiene e il volume degli scambi nelle ultime 24 ore.
- **Statistiche di rete:** i blockchain explorer forniscono agli utenti le statistiche della rete in tempo reale e alcuni dati storici sulle sue prestazioni passate. [9]

Alcuni esempi di blockchain explorer sono Etherscan [10] per la blockchain Ethereum e PolygonScan [11] per la blockchain Polygon.

Scelta della blockchain

Prima di addentrarci nell'analisi del sistema di protezione, è necessario compiere alcune considerazioni su alcune scelte preliminari effettuate durante l'ideazione del metodo vero e proprio.

La prima è la seguente: visto l'elevatissimo numero di blockchain esistenti, è assolutamente necessario scegliere quella che più si adatta alle caratteristiche del nostro progetto, seguendo una serie di criteri di selezione.

3.1 Compatibilità con EVM

Il primo criterio utilizzato è stato quello di scegliere una blockchain EVM-compatibile. EVM, acronimo di Ethereum Virtual Machine, è un ambiente software che ha il compito di eseguire tutte le operazioni e le transazioni sulla blockchain di Ethereum: archivia i dati sulla blockchain, elabora le transazioni e calcola i gas fees.

L'EVM è una virtual machine Turing-completa, ossia è in grado di eseguire qualsiasi algoritmo o programma. Il vantaggio di ciò è che Ethereum è ampiamente programmabile, ossia che degli sviluppatori possono scrivere comodamente degli smart contracts e delle applicazioni decentralizzate ed eseguirli sulla rete Ethereum.

L'EVM, inoltre, esegue il codice in modo deterministico: ogni smart contract produrrà sempre lo stesso output dato lo stesso input, indipendentemente da dove viene eseguito o da chi lo sta eseguendo. Questa proprietà garantisce che gli smart contracts vengano eseguiti senza interferenze da fonti esterne.

L'EVM è progettato per essere isolato dal resto del sistema operativo del computer. Questo implica che l'EVM può interagire solo con la rete Ethereum, impedendo ai malintenzionati di accedere al sistema sottostante o di compromettere la sicurezza della rete.

Fatta questa doverosa premessa, cosa significa e che vantaggi dà il fatto di utilizzare una blockchain EVM-compatibile?

La compatibilità EVM si riferisce alla capacità di una blockchain di eseguire l'EVM e gli smart contracts di Ethereum. Una blockchain EVM-compatibile garantisce quindi dei notevoli vantaggi per gli sviluppatori e non solo:

- Essendo Ethereum una delle blockchain più popolari, è molto più semplice trovare online la soluzione ad eventuali problematiche riscontrate durante lo sviluppo delle applicazioni decentralizzate e degli smart contracts, nonché eventuali plugin ed estensioni che rendono lo sviluppo più semplice e veloce.
- Le applicazioni decentralizzate e gli smart contracts sviluppati per una blockchain compatibile con EVM possono essere facilmente migrati su altre blockchain compatibili con EVM con modifiche minime al codice.
- L'EVM fornisce un ambiente standardizzato per scrivere smart contracts e sviluppare applicazioni decentralizzate: ciò significa che posso scrivere tutto il codice in Solidity, uno dei linguaggi di programmazione più utilizzati per Ethereum, e di seguito adattarlo a più situazioni con costi veramente molto bassi.
- Gli smart contracts e le applicazioni decentralizzate sviluppate su blockchain EVM-compatibili sono costruite sulla rete Ethereum per natura, il che significa che sono automaticamente disponibili ai milioni di wallet collegati ad essa. [12]

3.2 Ecosostenibilità

Il secondo criterio utilizzato è stato quello di scegliere una blockchain eco-friendly. In particolare, ci siamo concentrati sulla ricerca di una blockchain che permetta di risparmiare il più possibile in termini di consumi elettrici e di emissioni di anidride carbonica. Dopo un'attenta ricerca in rete, abbiamo individuato uno studio dell'ottobre 2022 del Crypto Carbon Ratings Institute (CCRI) [13], una compagnia di ricerca tedesca che si occupa di studiare e fornire dati sugli aspetti sostenibili delle blockchain e delle criptovalute.

Blockchain/Entità	Consumo elettrico annuo [kWh]
Famiglia statunitense	10'600
Polkadot	70'246
Polygon	118'934
Tezos	123'500
Tron	160'210
Avalanche	592'240
Cardano	638'488
Algorand	721'187
Ethereum post-Merge	2'105'304
Solana	3'900'968
Ethereum pre-Merge	21'986'314'331
Bitcoin	97'111'432'951

Tabella 1: Consumo elettrico annuo [kWh] di varie blockchain confrontato con quello di una famiglia statunitense [13]

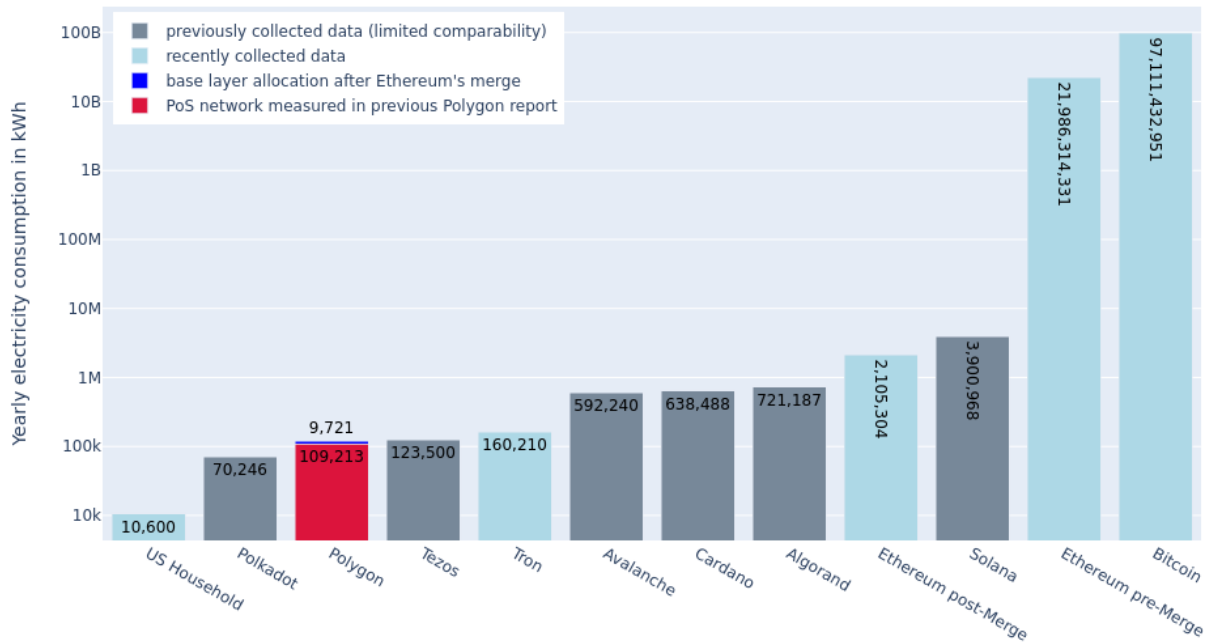


Figura 7: Consumo elettrico annuo [kWh] di varie blockchain confrontato con quello di una famiglia statunitense [13]

Blockchain/Entità	Emissioni di CO2 annue [tonnellate]
Volo intercontinentale	6.1
Polkadot	33.37
Polygon	55.0
Tezos	58.66
Tron	76.1
Avalanche	281.31
Cardano	303.28
Algorand	342.56
Ethereum post-Merge	704.055
Solana	1852.96

Tabella 2: Emissioni di CO2 annue [tonnellate] di varie blockchain confrontate con quelle di un volo in business class da Monaco di Baviera e San Francisco [13]

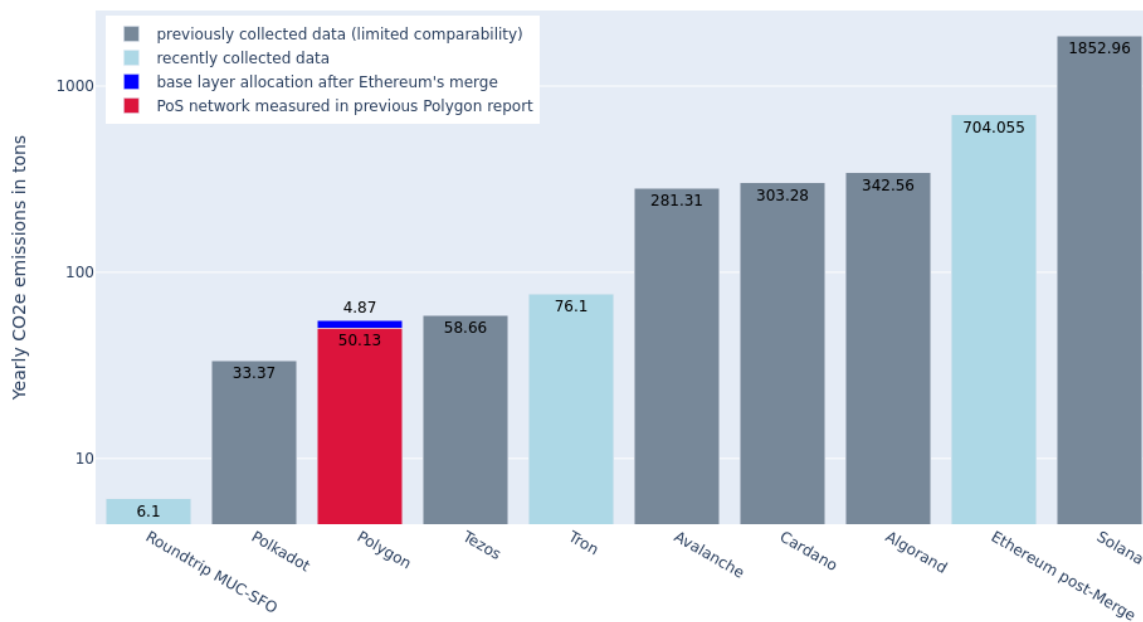


Figura 8: Emissioni di CO2 annue [tonnellate] di varie blockchain confrontate con quelle di un volo in business class da Monaco di Baviera e San Francisco [13]

Osserviamo i dati riguardanti i consumi elettrici: il primo fatto che salta all'occhio è sicuramente l'enorme differenza di elettricità consumata dalla blockchain di Bitcoin e quella di Ethereum pre-Merge, ossia prima che diventasse una blockchain Proof-of-Stake, rispetto a tutte le altre. Questo ci fa comprendere quanto scegliere una blockchain green sia realmente importante.

Inoltre, si può notare anche la netta differenza tra il consumo elettrico di una famiglia statunitense rispetto alle blockchain che consumano di meno. Questo ci permette di concludere che, al momento, la tecnologia blockchain è ancora molto esosa in termini di risorse elettriche, anche se si sta lavorando molto per ridurre questo aspetto.

Per quanto riguarda le emissioni di CO2 il discorso è molto simile a quello fatto per il consumo elettrico. Anche qui, infatti, ci sono delle blockchain che emettono significativamente molta più CO2 rispetto alle altre, vedasi Solana ed Ethereum post-Merge, così come è vero che anche le blockchain che emettono meno anidride carbonica inquinano molto di più l'ambiente rispetto ad un volo intercontinentale in business class.

3.3 Identificazione della blockchain

Dopo un'attenta analisi, la scelta della blockchain è ricaduta su Polygon.

Polygon è una blockchain creata per ridurre i costi di transazione di Ethereum e migliorarne la velocità complessiva. A causa della sua popolarità, infatti, Ethereum ha dei tempi di transazione e dei gas fees parecchio elevati. Inoltre, Polygon ha un token nativo, il MATIC, e utilizza il metodo di convalida Proof-of-Stake.

Polygon è una sidechain di Ethereum. Come spiegato in precedenza, le sidechain sono delle blockchain layer 2 collegate ad una blockchain layer 1 tramite un bridge bidirezionale, ma separate e indipendenti da essa. Tra le varie competenze, esse sono in grado di ricevere del lavoro da svolgere dalla blockchain principale e, dopo averlo compiuto, restituiscono unicamente i risultati, permettendo di guadagnare in termini di efficienza.

Oltre alle caratteristiche appena descritte, Polygon soddisfa molto bene i due criteri di selezione che abbiamo definito. Infatti, oltre ad essere una blockchain EVM-compatibile ed avere uno stretto legame con Ethereum, è una delle più attente all'ambiente (vedi Tabella 1, Figura 7, Tabella 2 e Figura 8).

Se ciò non bastasse, sulla pagina web [14] è possibile osservare che la sostenibilità e il bene del pianeta sono due obiettivi cardine del progetto Polygon.

Capitolo 4

Scelta dell'approccio

Una volta selezionata la blockchain sulla quale far vivere il nostro sistema, una seconda scelta necessaria riguarda la cardinalità degli NFT. Dato che una parte essenziale del metodo prevede il minting di NFT, ossia la loro creazione e il loro conio sulla blockchain, è assolutamente fondamentale capire quanti e quali NFT creare, in particolare per i casi in cui ho tante copie di un bene da proteggere.

Facciamo un esempio per chiarire il problema: supponiamo di voler proteggere un orologio di lusso

- Se l'orologio fosse prodotto in un'unica copia, allora sarebbe sufficiente creare un unico NFT e associarlo all'orologio.
- Se l'orologio fosse prodotto in più copie indistinguibili tra loro, invece, la situazione sarebbe più interessante: è meglio creare un unico NFT e associarlo a tutte le copie oppure conviene mintare un NFT per ogni singola copia dell'orologio?

È semplice intuire che esistono due modi di agire: l'approccio 1:N e l'approccio 1:1. Analizziamo nel dettaglio le caratteristiche, i pro e i contro di ognuno di essi.

4.1 Approccio 1:N

L'approccio 1:N di creazione degli NFT è il più semplice e meno dispendioso in termini di risorse. Esso prevede semplicemente di creare un unico NFT e associarlo a tutte le copie dell'oggetto.

Sebbene questo metodo permetta di risparmiare molte risorse in caso un oggetto sia prodotto in un gran numero di copie, ha anche il grande svantaggio di essere molto vulnerabile alle truffe. Per via del fatto che non è chiaramente possibile trasferire l'NFT all'effettivo proprietario dell'oggetto fisico dopo la prima vendita, è molto complesso trovare un metodo per associare in modo sicuro l'NFT al bene senza che un potenziale truffatore copi il metodo di associazione e lo apponga su un oggetto fasullo. Un banale QR code, ad esempio, non sarebbe assolutamente sufficiente, e questo ci costringerebbe a cercare un metodo di associazione molto più complesso.

4.2 Approccio 1:1

L'approccio 1:1 di creazione degli NFT è invece più oneroso e leggermente più complesso. Prevede, infatti, di creare un NFT per ogni oggetto effettivo: se l'orologio dell'esempio precedente fosse prodotto in 1000 copie, allora sarà necessario creare 1000 NFT e numerarli come "Orologio #1", "Orologio #2", etc.

A differenza dell'approccio 1:N, ora abbiamo un'associazione univoca tra NFT e oggetto fisico: questo ci permette di trasferire l'NFT all'effettivo proprietario del bene ogni qualvolta esso viene venduto. Grazie a questa proprietà, come spiegato più chiaramente nella Sezione 6.1, è sufficiente un semplicissimo QR code come metodo di associazione.

Il problema di questo approccio, però, è che richiede una grande quantità di risorse sotto tre principali aspetti:

- **Soldi**

Assumendo un costo dei gas fees di Polygon di circa 0.01\$, creare 100.000 NFT ha un costo base di 1000\$, una cifra che potrebbe essere poco sostenibile per il piccolo artigiano locale, scoraggiandolo dal voler utilizzare il sistema da noi ideato.

Tuttavia, si può assumere che un numero così grande di NFT sarà richiesto esclusivamente da grandi aziende che, di conseguenza, non avrebbero problemi a spendere una cifra medio-alta per garantire protezione ai loro prodotti.

- **Spazio**

Creando una grande quantità di NFT occuperemo molto spazio sulla blockchain, ma anche questo non è un problema visto che, se non per delle limitazioni tecniche, le blockchain hanno una capacità potenzialmente illimitata.

Un problema leggermente più grave, tuttavia, è il fatto che potrebbero essere caricati inutilmente sulla blockchain NFT di beni che rimarranno invenduti.

- **Tempo**

Il principale problema dell'approccio 1:1 di creazione degli NFT è il seguente: assumendo un tempo di circa 15 secondi per il minting di un NFT su Polygon e senza considerare eventuali conflitti o errori, il tempo per caricare, ad esempio, 200.000 NFT è di circa 34 giorni!

Per mascherare i problemi rimasti in termini di tempo e spazio, una possibile soluzione può essere di creare gli NFT dinamicamente a mano a mano che gli oggetti fisici vengono venduti (vedi Sezione 6.3).

4.3 Definizione dell'approccio corretto

Analizzati i pro e i contro descritti in questo capitolo, abbiamo optato per l'adozione dell'approccio 1:1.

La motivazione di questa scelta è legata al principale svantaggio che l'approccio 1:N porta con sé. Esso è molto grave e rischierebbe di danneggiare l'intero sistema rendendolo sostanzialmente inutilizzabile. Gli svantaggi dell'approccio 1:1, invece, sono mascherabili, mentre i suoi vantaggi sono particolarmente importanti.

Oltre a questa motivazione c'è anche una questione semantica. Gli NFT sono stati progettati per rappresentare oggetti unici nel loro genere, perciò utilizzare un NFT per rappresentare più oggetti fisici è una contraddizione. Al contrario, è preferibile distinguere i beni prodotti in più copie attraverso l'uso di numeri incrementali, in modo da renderli effettivamente unici (ad esempio: non esisteranno mai due oggetti "Orologio #1").

Capitolo 5

Il metodo

Siamo finalmente giunti alla descrizione del metodo vero e proprio, ossia il sistema per proteggere un qualsiasi bene come precedentemente descritto.

Si noti che questo metodo, nei dettagli implementativi che saranno forniti, è garantito funzionare solo con blockchain EVM-compatibili, in particolare con Polygon.

La sua struttura generale, tuttavia, si può prendere come modello per *qualsiasi* altro caso d'uso, quindi non solo in termini di blockchain diverse.

5.1 Documento di specifica per il cliente

La prima fase prevede la definizione di un documento di specifica per il cliente.

In questo documento sono elencate tutte le informazioni che il cliente (o un suo delegato) deve produrre ed esibire affinché si possa creare un NFT rappresentante un bene materiale di cui è proprietario.

Qui di seguito sono elencati tutti i passi necessari alla stesura del documento.

5.1.1 Identificazione del cliente

Il cliente (o un suo delegato) deve presentare delle prove che attestino la sua identità. Questo passo è necessario per evitare che qualsiasi persona possa commissionare la creazione di NFT fingendosi qualcun altro.

Esempi: bigliettino da visita, pagina LinkedIn, contratto di assunzione, ...

5.1.2 Certificato di autenticità del prodotto

Il cliente (o un suo delegato) deve presentare un documento fisico o digitale che attesti che il prodotto su cui si vuole creare un NFT è di sua proprietà (o della persona da cui è stato delegato). Si noti bene che verrà archiviata una copia firmata di questo documento.

Questo passo è necessario per evitare che qualsiasi persona possa commissionare la creazione di NFT su prodotti dei quali non possiede il diritto di proprietà.

Esempi contenuto: certificato di un ente ufficiale, marchio registrato, ...

Esempi formato: cartaceo, PDF, ...



Figura 9: Esempio di certificato di autenticità [15]

5.1.3 Scheda del prodotto

Il cliente (o un suo delegato) deve presentare un documento fisico o digitale in cui siano elencate delle caratteristiche fondamentali dell'oggetto che lo rendono unico e riconoscibile. Queste informazioni diverranno i metadati dell'NFT.

Esempi formato: cartaceo, PDF, DOC, DOCX, ...

5.1.4 Rappresentazione grafica del prodotto

Il cliente (o un suo delegato) deve presentare un'immagine statica o animata che rappresenterà l'NFT insieme ai metadati descritti nel punto precedente.

Esempi formato: JPG, PNG, GIF, ...

Esempio immagine: si veda la Figura 10



Figura 10: Esempio di immagine [16]

5.2 Operazioni preliminari

Dopo aver raccolto tutte le informazioni necessarie dal cliente, la seconda fase prevede di svolgere una serie di operazioni in vista delle fasi successive.

Entrando nel dettaglio, le operazioni sono le seguenti:

- Accedere ad Alchemy [17], una piattaforma per blockchain developers che fornisce un'interfaccia per comunicare facilmente con alcune blockchain;
- Autenticarsi su un crypto wallet a scelta. In particolare, noi abbiamo utilizzato MetaMask [18].

Durante lo sviluppo del sistema, inoltre, è stata utilizzata Polygon Mumbai, una Testnet di Polygon, e il sito web [19] per la generazione di MATIC da utilizzare su Mumbai.

Nei prossimi punti, invece, descriveremo il metodo finale usando la Polygon Mainnet.

5.3 Creazione di uno smart contract

Siamo giunti finalmente alla fase di scrittura dello smart contract che ci servirà successivamente per mintare gli NFT.

In particolare, i passi da seguire sono i seguenti:

1. Creare una nuova app in Alchemy

Per facilitare lo sviluppo dello smart contract e il minting degli NFT, iniziamo creando una nuova app in Alchemy, selezionando in particolare la blockchain e la rete sulle quali caricheremo lo smart contract e, di conseguenza, gli NFT.

Nel nostro caso, quindi, sceglieremo come blockchain *Polygon PoS* e come rete *Polygon Mainnet*.

2. Creare un nuovo progetto in locale

È necessario creare una nuova cartella nel file system locale. In essa verrà inserito tutto il progetto che stiamo costruendo.

Una volta creata, spostiamoci in essa ed eseguiamo da riga di comando

```
npm init
```

npm è un gestore di pacchetti per JavaScript, uno dei linguaggi di programmazione che utilizzeremo in seguito.

3. Installare Hardhat

Eseguiamo da riga di comando

```
npm install --save-dev hardhat
```

per installare Hardhat, un ambiente di sviluppo che aiuta a compilare, debuggare e testare lo smart contract localmente prima di pubblicarlo sulla blockchain.

4. Creare un progetto Hardhat

Eseguiamo da riga di comando

```
npx hardhat init
```

e successivamente selezioniamo l'opzione

```
Create an empty hardhat.config.js
```

Questo permetterà di creare un nuovo progetto Hardhat.

5. Aggiungere cartelle per organizzare meglio il progetto

All'interno della cartella del nostro progetto creiamo la cartella *contracts*, che conterrà il codice dello smart contract, e la cartella *scripts*, che includerà i codici JavaScript che permetteranno di operare con lo smart contract.

6. Scrivere il codice dello smart contract in Solidity

Attraverso il comando

```
npm install @openzeppelin/contracts@^4.9.3
```

includiamo nel nostro progetto una libreria che ci servirà in seguito.

Successivamente è arrivato il momento di scrivere il codice dello smart contract in un file *X.sol* (con X a scelta) e di salvarlo all'interno della cartella *contracts*.

Qui di seguito è riportato un codice d'esempio di uno smart contract.

```
1 // Contract based on [https://docs.openzeppelin.com/contracts
  // /3.x/erc721]
2 // SPDX-License-Identifier: MIT
3
4 // Versione di Solidity utilizzata
5 pragma solidity ^0.8.21;
6
7 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
8 import "@openzeppelin/contracts/token/ERC721/extensions/
  ERC721URIStorage.sol";
9 import "@openzeppelin/contracts/utils/Counters.sol";
10 import "@openzeppelin/contracts/access/Ownable.sol";
11
12 // Definizione dello smart contract
13 contract Contract is ERC721URIStorage, Ownable {
14
```

```

15 // Dichiarazione del contatore per gli ID
16 using Counters for Counters.Counter;
17 Counters.Counter private _tokenIds;
18
19 // Costruttore (nome del contratto, simbolo del contratto)
20 // Il nome e' quello che comparira' sul crypto wallet
21 constructor() ERC721("Esempio", "NFT") {}
22
23 // Funzione che permette di coniare un nuovo NFT da questo
    smart contract
24 // recipient: indirizzo che riceverà l'NFT coniato
25 // tokenURI: collegamento ad un file JSON che contiene i
    metadati dell'NFT coniato
26 function mintNFT(address recipient, string memory tokenURI)
    public onlyOwner returns (uint256) {
27     // Incrementa il contatore
28     _tokenIds.increment();
29     // Conia il nuovo NFT
30     uint256 newItemId = _tokenIds.current();
31     _mint(recipient, newItemId);
32     _setTokenURI(newItemId, tokenURI);
33     // Restituisce l'ID del nuovo NFT
34     return newItemId;
35 }
36 }

```

7. Collegare Alchemy e il crypto wallet al progetto

Dopo aver eseguito da riga di comando

```
npm install dotenv -save
```

creiamo un file di configurazione **.env** all'interno della cartella del progetto.

Esso deve contenere le seguenti informazioni

```

# APIKey HTTPS dell'app su Alchemy
API_URL = "https://polygon-mumbai.g.alchemy.com/v2/abcdefghijkl"

# Chiave privata del crypto wallet
PRIVATE_KEY = "0123456789"

```

8. Installare hardhat-ethers

Eseguendo da riga di comando

```
npm install --save-dev @nomiclabs/hardhat-ethers ethers@^5.0.0
```

si include hardhat-ethers nel proprio progetto, un plugin che rende più semplice interagire ed effettuare richieste alle blockchain EVM. Esso wrappa i metodi JSON-RPC standard, permettendo di interagire con la blockchain con dei metodi più user-friendly.

9. Aggiornare hardhat.config.js

Il file *hardhat.config.js* contiene un riassunto delle caratteristiche del contratto e dei plugin utilizzati da esso.

In particolare, va aggiornato come segue:

```
1 require('dotenv').config();
2 require("@nomiclabs/hardhat-ethers");
3
4 const {API_URL, PRIVATE_KEY} = process.env;
5
6 module.exports = {
7   solidity: "0.8.21",
8   defaultNetwork: "polygon-mainnet",
9   networks: {
10     hardhat: {},
11     polygon_mainnet: {
12       url: API_URL,
13       accounts: ['0x${PRIVATE_KEY}']
14     }
15   }
16 }
```

10. Scrivere lo script `deploy.js`

Questo script JavaScript serve per distribuire il contratto, ossia per caricarlo sulla blockchain scelta, nel nostro caso Polygon.

Ricordarsi di inserire il file ***deploy.js*** all'interno della cartella ***scripts***.

Qui di seguito il codice.

```
1  async function main() {
2
3      // Crea un "fabbricatore di istanze" dello smart contract
4      const MyNFT = await ethers.getContractFactory("Contract")
5
6      // Crea una nuova istanza dello smart contract
7      const myNFT = await MyNFT.deploy()
8      await myNFT.deployed()
9      console.log("Contratto distribuito all'indirizzo:", myNFT.
        address)
10 }
11
12 // Esecuzione di main e cattura errori
13 main()
14   .then(() => process.exit(0))
15   .catch((error) => {
16       console.error(error)
17       process.exit(1)
18   })
```

11. Distribuire il contratto eseguendo lo script

Eseguiamo da riga di comando

```
npx hardhat --network "nomeRete" run scripts/deploy.js
```

Nel nostro caso, quindi, il comando da eseguire è

```
npx hardhat --network polygon_mainnet run scripts/deploy.js
```


5.4 Verifica dello smart contract

Dopo aver caricato il contratto sulla blockchain, la fase successiva è la sua verifica. Questa fase consiste nel confrontare il codice sorgente originale del contratto con il bytecode compilato che viene effettivamente eseguito sulla blockchain.

Questo processo è molto importante perché il codice del contratto pubblicato potrebbe essere diverso da quello effettivamente eseguito sulla blockchain, a causa, ad esempio, di errori nella compilazione o di manipolazioni malevole.

Inoltre, verificare lo smart contract ci darà successivamente la possibilità di eseguire delle importanti operazioni su PolygonScan in modo molto più user-friendly.

I passi da seguire per verificare il contratto, quindi, sono i seguenti:

1. Installare hardhat-verify

Eseguiamo da riga di comando

```
npm install --save-dev @nomicfoundation/hardhat-verify
```

per installare hardhat-verify, un plugin che aiuta gli sviluppatori a verificare gli smart contract sulle blockchain EVM-compatibili.

2. Aggiornare .env

Aggiungiamo a *.env* una variabile contenente una API Key di PolygonScan.

Per ottenere una API Key si visiti la seguente pagina web: [20].

Il codice di *.env*, quindi, risulta essere il seguente:

```
# APIKey HTTPS dell'app su Alchemy
API_URL = "https://polygon-mumbai.g.alchemy.com/v2/abcdefghijkl"

# Chiave privata del crypto wallet
PRIVATE_KEY = "0123456789"

# APIKey di PolygonScan
API_KEY = "kjihgfedcba"
```

3. Aggiornare hardhat.config.js

Aggiorniamo *hardhat.config.js* con le ultime novità.

```
1 require('dotenv').config();
2 require("@nomiclabs/hardhat-ethers");
3 require("@nomicfoundation/hardhat-verify");
4
5 const {API_URL, PRIVATE_KEY, API_KEY} = process.env;
6
7 module.exports = {
8   solidity: "0.8.21",
9   defaultNetwork: "polygon-mainnet",
10  networks: {
11    hardhat: {},
12    polygon_mainnet: {
13      url: API_URL,
14      accounts: ['0x${PRIVATE_KEY}']
15    }
16  },
17  etherscan: {
18    apiKey: {
19      polygon_mainnet: API_KEY
20    }
21  }
22 }
```

4. Verificare il contratto

Eseguiamo da riga di comando

```
npx hardhat verify --network "nome rete" "indirizzo contratto"
```

Nel nostro caso, quindi, il comando da eseguire è

```
npx hardhat verify --network polygon_mainnet 0x123456789
```

5.5 Creazione degli NFT

Abbiamo creato, caricato sulla blockchain e verificato uno smart contract. La fase successiva prevede di utilizzare il contratto intelligente per mintare gli NFT secondo l'approccio definito nel Capitolo 4. Procediamo con l'analisi dei passi da seguire:

1. Decentralizzare l'immagine e i metadati dell'NFT

Per questo punto utilizzeremo Pinata [21], un sito che permette di caricare file sull'InterPlanetary File System (IPFS), una rete peer-to-peer che permette di archiviare e condividere dati in un file system distribuito. Questo sito servirà per decentralizzare alcune delle informazioni raccolte nel Documento di specifica per il cliente (vedi Sezione 5.1). In particolare:

- un file contenente l'immagine che rappresenterà l'NFT;
- un file in formato .json che includerà i metadati dell'NFT.

Chiaramente andrà caricata una coppia di questi file per ogni tipo di NFT da creare. Una volta eseguito l'upload, inoltre, verranno forniti da Pinata gli indirizzi IPFS nei quali i file sono stati memorizzati.

Qui sotto un esempio di file .json contenente i metadati. Notiamo che alla riga 13 è indicato l'indirizzo IPFS dell'immagine associata a questi metadati.

```
1 {
2   "attributes": [
3     {
4       "trait_type": "Color",
5       "value": "Red"
6     },
7     {
8       "trait_type": "Sleeves",
9       "value": "Long"
10    }
11  ],
12  "description": "Descrizione d'esempio",
13  "image": "ipfs://ABCDEFGHIIJK",
14  "name": "Esempio"
15 }
```

Per fare un po' d'ordine possiamo anche creare una cartella *images* e una cartella *metadata*, inserirle all'interno della cartella del nostro progetto ed utilizzarle per archiviare i file immagine e i file .json appena descritti.

2. Installare Alchemy Web3

Eseguiamo da riga di comando

```
npm install @alch/alchemy-web3
```

per installare Web3, un plugin con obiettivi molto simili ad hardhat-ethers.

3. Aggiornare .env

Per mintare gli NFT serve conoscere anche la chiave pubblica del crypto wallet, l'indirizzo dello smart contract sulla blockchain, l'ABI (interfaccia per interagire con il contratto) e l'indirizzo IPFS dei metadati dell'NFT.

Aggiorniamo quindi il file *.env* come segue:

```
# APIKey HTTPS dell'app su Alchemy
API_URL = "https://polygon-mumbai.g.alchemy.com/v2/abcdefghijkl"

# Chiave privata del crypto wallet
PRIVATE_KEY = "0123456789"

# APIKey di PolygonScan
API_KEY = "kjihgfedcba"

# Chiave pubblica del crypto wallet
PUBLIC_KEY = "0x9876543210"

# Indirizzo dello smart contract
CONTRACT_ADDRESS = "0xa1b2c3d4e5"

# Definizione dell'ABI (../artifacts/contracts/'
  nomeFileContratto'.sol/'nomeContratto'.json)
CONTRACT_ABI_PATH = "artifacts\contracts\Contratto.sol\Contract
  .json"

# Indirizzo IPFS dei metadati dell'NFT
IPFS = "ipfs://lmnopqrstuvwxyz"
```

4. Scrivere lo script mint-nft.js

Il prossimo passo prevede di scrivere uno script per mintare gli NFT a partire dallo smart contract definito nella Sezione 5.3.

Inseriamo questo codice nella cartella *scripts*.

In caso volessimo coniare NFT di tipo diverso, è necessario modificare l'indirizzo IPFS dei metadati dell'NFT all'interno del file *.env*.

```
1 // Carica le variabili di ambiente dal file .env
2 require("dotenv").config()
3 // Imposta il provider Web3
4 const {createAlchemyWeb3} = require("@alch/alchemy-web3")
5 const web3 = createAlchemyWeb3(process.env.API_URL)
6
7 // Imposta lo smart contract
8 const contract = require(process.env.CONTRACT_ABI_PATH)
9 // Definizione dell'indirizzo del contratto nella blockchain
10 const contractAddress = process.env.CONTRACT_ADDRESS
11 // Creazione di una nuova istanza del contratto
12 const nftContract = new web3.eth.Contract(contract.abi,
    contractAddress)
13
14 // Funzione per il conio di un nuovo NFT
15 async function mintNFT(tokenURI) {
16     // Ottieni l'ultimo nonce. Tenerne traccia serve per motivi
    // di sicurezza (evitare attacchi di riproduzione)
17     const nonce = await web3.eth.getTransactionCount(process.env.
        PUBLIC_KEY, "latest")
18
19     // Costruisci l'oggetto transazione
20     const tx = {
21         from: process.env.PUBLIC_KEY,
22         to: contractAddress,
23         nonce: nonce,
24         gas: 500000,
25         data: nftContract.methods.mintNFT(process.env.PUBLIC_KEY,
            tokenURI).encodeABI()
26     }
```

```

27 // Firma la transazione
28 const signPromise = web3.eth.accounts.signTransaction(tx,
    process.env.PRIVATE_KEY)
29
30 // Invia la transazione alla blockchain
31 signPromise
32   .then((signedTx) => {
33     // Restituisce l'hash della transazione
34     web3.eth.sendSignedTransaction(
35       signedTx.rawTransaction,
36       function (err, hash) {
37         if (!err) {
38           console.log("L'hash della transazione e'", hash)
39         } else {
40           console.log("Qualcosa e' andato storto con la tua
              transazione:", err)
41         }
42       }
43     )
44   })
45   .catch((err) => {
46     console.log("Promise fallita:", err)
47   })
48 }
49
50 // Minta l'NFT
51 mintNFT(process.env.IPFS)

```

5. Mintare l'NFT eseguendo lo script

Infine, eseguiamo da riga di comando

```
node scripts/mint-nft.js
```

per avviare lo script definito al punto precedente e mintare l'NFT.

5.6 Trasferimento degli NFT al cliente

La fase finale prevede di trasferire l’NFT appena creato al cliente. Così facendo, sulla blockchain saranno memorizzati in chiaro e ben visibili a tutti sia l’indirizzo di chi ha mintato l’NFT, ossia il nostro, sia l’indirizzo del primo proprietario, ossia il cliente. I passi da seguire, in particolare, sono i seguenti:

1. Aggiornare `.env`

Per trasferire l’NFT al cliente, serve chiaramente conoscere il suo indirizzo.

Aggiungiamo anche questa informazione al file `.env`.

```
# APIKey HTTPS dell'app su Alchemy
API_URL = "https://polygon-mumbai.g.alchemy.com/v2/abcdefghijkl"

# Chiave privata del crypto wallet
PRIVATE_KEY = "0123456789"

# APIKey di PolygonScan
API_KEY = "kjihgfedcba"

# Chiave pubblica del crypto wallet
PUBLIC_KEY = "0x9876543210"

# Indirizzo dello smart contract
CONTRACT_ADDRESS = "0xa1b2c3d4e5"

# Definizione dell'ABI
(../artifacts/contracts/'nomeFileContratto'.sol/'nomeContratto'
'.json)
CONTRACT_ABI_PATH = "artifacts\contracts\Contratto.sol\Contract
.json"

# Indirizzo IPFS dei metadati dell'NFT
IPFS = "ipfs://lmnopqrstuvwxyz"

# Chiave pubblica del destinatario
RECEIVER_KEY = "0x5432109876"
```

2. Scrivere lo script transferNFT.js

Scriviamo ora uno script per trasferire NFT da un mittente ad un destinatario.

Nel nostro caso, quindi, trasferiremo NFT al cliente.

```
1 // Carica le variabili di ambiente dal file .env
2 require("dotenv").config()
3
4 // Imposta il provider Web3
5 const { createAlchemyWeb3 } = require("@alch/alchemy-web3")
6 const web3 = createAlchemyWeb3(process.env.API_URL)
7
8 // Imposta gli account mittente e destinatario
9 const accountFrom = process.env.PUBLIC_KEY;
10 const accountTo = process.env.RECEIVER_KEY;
11
12 // Imposta lo smart contract e l'ID dell'NFT da trasferire
13 const nftContractAddress = process.env.CONTRACT_ADDRESS;
14 const contract = require(process.env.CONTRACT_ABI_PATH)
15 const nftContract = new web3.eth.Contract(contract.abi,
16     nftContractAddress);
17
18 // Ottieni la chiave privata dell'account mittente
19 const privateKey = Buffer.from(process.env.PRIVATE_KEY, 'hex');
20
21 // Costruisci la transazione per il trasferimento dell'NFT
22 const transferTx = nftContract.methods.safeTransferFrom(
23     accountFrom, accountTo, nftId);
24
25 // Ottieni il nonce dell'account mittente
26 web3.eth.getTransactionCount(accountFrom, (err, nonce) => {
27     if (err) {
28         console.error('Errore nel recuperare il nonce:', err);
29         return;
30     }
31
32
```



```

33 // Costruisci l'oggetto transazione
34 const txObject = {
35     nonce: nonce,
36     gasLimit: web3.utils.toHex(500000),
37     gasPrice: web3.utils.toHex(web3.utils.toWei('30', 'gwei
38     ')),
39     to: nftContractAddress,
40     data: encodedTx
41 };
42
43 // Firma la transazione
44 const signPromise = web3.eth.accounts.signTransaction(
45     txObject, process.env.PRIVATE_KEY)
46
47 // Invia la transazione alla blockchain
48 signPromise
49 .then((signedTx) => {
50     // Restituisce l'hash della transazione
51     web3.eth.sendSignedTransaction(
52         signedTx.rawTransaction,
53         function (err, hash) {
54             if (!err) {
55                 console.log("L'hash della transazione e'", hash)
56             } else {
57                 console.log("Qualcosa e' andato storto con la tua
58                 transazione:", err)
59             }
60         }
61     )
62 })
63 .catch((err) => {
64     console.log("Promise fallita:", err)
65 })
66 });

```

Si noti che lo script presente nelle pagine precedenti trasferisce solo l’NFT rappresentato dal tokenID definito alla riga 16.

Per trasferire una serie di NFT consecutivi tra loro, come è ragionevole che sia, è sufficiente modificare leggermente il codice utilizzando un contatore che viene incrementato ad ogni trasferimento.

3. Trasferire l’NFT eseguendo lo script

Infine, eseguiamo da riga di comando

```
node scripts/transferNFT.js
```

per avviare lo script definito al punto precedente e trasferire uno o più NFT al cliente.

Idee di utilizzo

Nel Capitolo 5 abbiamo descritto un metodo concreto che permette di creare una serie di NFT a partire da un documento di specifica steso con i requisiti del cliente. La situazione finale, quindi, è che il cliente possiede nel proprio crypto wallet un NFT per ognuno dei beni che vuole proteggere, come da approccio definito nel Capitolo 4. A questo punto si può discutere di diverse idee e scelte per completare il sistema.

6.1 Come associare l’NFT all’oggetto fisico

Una prima scelta da effettuare riguarda come creare un legame indissolubile tra l’NFT rappresentante il bene e l’oggetto stesso.

Un approccio molto semplice e funzionale è quello di creare un QR code e apporlo sull’oggetto fisico. Esso sarà un link ad una pagina che permette di consultare tutta la storia dell’NFT (es. PolygonScan), tra cui chi ha creato lo smart contract da cui è stato mintato e chi è l’attuale proprietario. Queste informazioni, infatti, sono sufficienti per permettere a chiunque di riconoscere se l’oggetto su cui è presente il QR code è autentico oppure no. Infatti, anche se un truffatore riuscisse a copiare il QR code e lo ponesse su un oggetto fasullo, non riuscirebbe a fornire delle prove attendibili che dimostrino il suo essere proprietario dell’oggetto.

Inoltre, il QR code fornisce protezione anche in caso il truffatore provi a creare degli NFT falsi e ad associarli ad oggetti fasulli. In questo caso, infatti, è sufficiente controllare che l’indirizzo di chi ha mintato l’NFT corrisponda a quello di chi è stato incaricato di farlo, ossia noi. Per questo punto, però, è necessario che chiunque conosca il nostro indirizzo: un modo molto semplice per raggiungere questo scopo è aggiungere al sito del cliente una sezione apposita in cui, oltre alle varie istruzioni su come identificare possibili truffe, sia scritto in chiaro il nostro indirizzo pubblico.

Per una maggiore sicurezza, infine, si potrebbe utilizzare un sistema di doppia autenticazione, ad esempio associando al QR code anche un chip integrato all'interno del bene da proteggere, chiaramente nei casi in cui è possibile farlo.

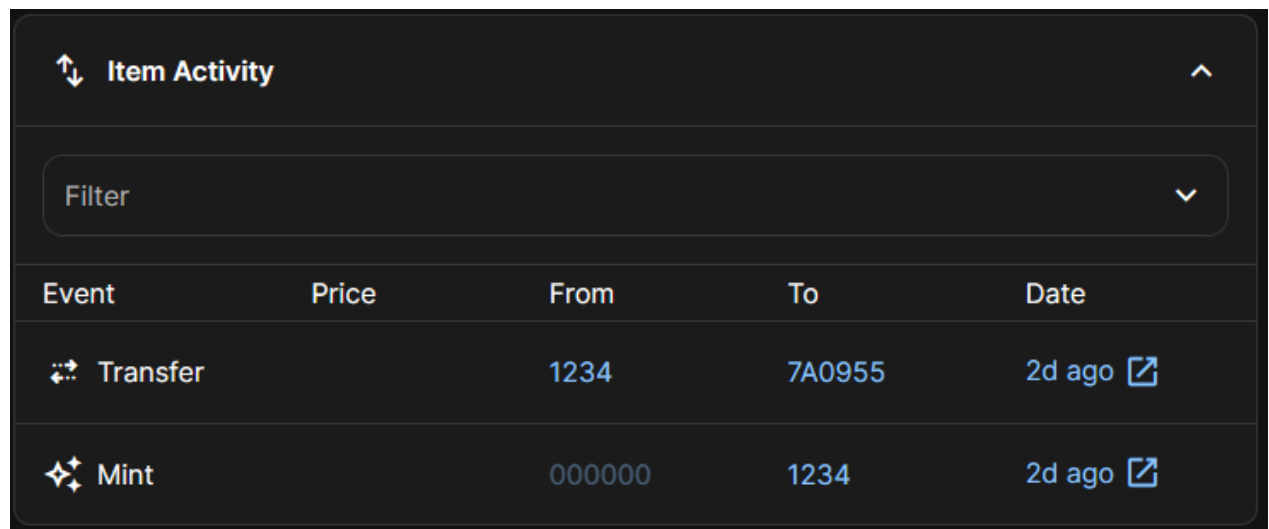
6.2 Come presentare l'NFT

Assumiamo di aver scelto il sistema di associazione tramite QR code. Una volta inquadrato il code che pagina viene aperta?

Per evitare di aprire la pagina PolygonScan che per alcuni utenti potrebbe essere considerato poco user-friendly, un'idea potrebbe essere di utilizzare uno store virtuale (es. OpenSea [22]).

Questi siti, oltre a visualizzare in modo molto comodo sia l'immagine sia i metadati dell'NFT, forniscono anche alcuni strumenti come la possibilità di aggiungere una descrizione per aiutare l'utente a comprendere meglio le caratteristiche del bene.

Infine, presentano anche una comoda tabella riassuntiva delle operazioni passate riguardanti l'NFT (vedi Figura 11), tra cui chi ha mintato l'NFT e tutti i proprietari, compreso quello attuale.



The image shows a screenshot of a web interface titled "Item Activity". It features a table with five columns: "Event", "Price", "From", "To", and "Date". There are two rows of data: a "Transfer" event and a "Mint" event. Each row includes a small icon to the left of the event name and a link icon to the right of the date. A "Filter" dropdown is located above the table.





Event	Price	From	To	Date
 Transfer		1234	7A0955	2d ago 
 Mint		000000	1234	2d ago 

Figura 11: Tabella delle operazioni sull'NFT [22]

6.3 Come vendere il bene

Assumendo di avere ogni prodotto fisico con un QR code associato, fornirò ora un'idea per un possibile sistema finale per vendere un bene materiale, dalla vendita iniziale in negozio o online fino ad una possibile rivendita a terzi.

Immaginiamo di avere un negozio fisico di una nota marca di prodotti di lusso e supponiamo che un cliente entri nel negozio con l'intenzione di acquistare una borsetta con un QR code associato.

6.3.1 Primo acquisto

Per mascherare i problemi dell'approccio 1:1 descritti nella Sezione 4.2, è ragionevole creare gli NFT dinamicamente anziché a priori staticamente. In particolare, quando un cliente acquista una borsetta con QR code dal negozio fisico, l'NFT che protegge quella borsetta non sarà ancora stato creato. Perciò la prima volta che viene inquadrato il QR code verrà aperto un form. In questo sarà necessario inserire il proprio indirizzo pubblico insieme ad una OTP generata al momento dell'acquisto.

Una volta svolta questa operazione, sarà eseguito automaticamente uno script che minterà l'NFT a nostro nome, che siamo stati incaricati di farlo, e successivamente lo trasferirà al cliente che lo ha acquistato, che sarà il primo proprietario effettivo.

6.3.2 Controlli vari e acquisti successivi al primo

Dopo la creazione e il trasferimento dell'NFT al primo proprietario, ogni qualvolta si inquadrerà il QR code verrà aperto OpenSea (come definito nella Sezione 6.2) che permetterà di svolgere tutti i controlli del caso.

Si noti che, in caso di acquisti successivi al primo, è fondamentale che il vecchio proprietario del bene materiale si occupi anche di trasferire l'NFT al nuovo proprietario, pena il non corretto funzionamento di tutto il sistema! Questa operazione può essere svolta, ad esempio, tramite un crypto wallet oppure andando a creare una pagina apposita che, dopo un'opportuna fase di autenticazione, esegua lo script della Sezione 5.6.

Conclusioni

In questo lavoro di tesi abbiamo terminato lo sviluppo di un sistema per garantire protezione ad un qualsiasi bene materiale.

Dopo aver chiarito l'obiettivo, siamo partiti definendo una serie di concetti preliminari necessari ai fini del nostro progetto, tra cui alcuni strumenti utilizzati concretamente.

Abbiamo proseguito scegliendo la blockchain su cui sviluppare effettivamente il sistema e, dopo aver compreso che avremmo utilizzato gli NFT, ci siamo occupati di selezionare l'approccio per la loro creazione che più si addiceva al nostro scopo.

Successivamente ci siamo occupati di definire un metodo vero e proprio che permettesse di creare una serie di NFT sulla blockchain scelta e secondo l'approccio deciso. In particolare, siamo partiti da un documento di specifica contenente tutte le informazioni necessarie per il nostro obiettivo fino ad ottenere una serie di NFT di proprietà del cliente, uno per ogni oggetto da proteggere.

Infine abbiamo visto alcuni spunti su come usare questi NFT in un possibile sistema concreto, concentrandoci su come associare l'NFT all'oggetto fisico, su come presentare l'NFT e su un possibile modo finale per vendere i beni e utilizzare la verifica tramite NFT per proteggerli.

Ringraziamenti

Un sentito ringraziamento al Dott. Andrea Visconti per l'opportunità fornitami e per la passione e l'impegno dimostrato sia nelle lezioni del suo splendido corso di Crittografia I sia nel rapporto con i tirocinanti e con gli studenti tutti.

Ringrazio poi la mia famiglia, in particolare i miei genitori Andrea e Roberta, per avermi supportato in questo lungo cammino di poco più di 3 anni, sostenendomi nei momenti di difficoltà e di sconforto e manifestando gioia e affetto per ogni scalino superato.

Ringrazio infine tutti i miei amici e colleghi, in particolare Andrea e Nicholas, per aver portato spensieratezza nelle mie giornate, consigliandomi sempre saggiamente quando mi sono trovato di fronte a delle scelte importanti e aiutandomi a portare il peso di certe situazioni spiacevoli, permettendomi di terminare questo viaggio con il sorriso.

Bibliografia

- [1] *On the cryptography of Distributed Ledger Technology - 1. Hash Functions*
URL: <https://www.finriskalert.it/wp-content/uploads/visconti.pdf> (visitato il 14/03/2024)
- [2] *Blockchain Merkle Trees*
URL: <https://www.geeksforgeeks.org/blockchain-merkle-trees/>
(visitato il 14/03/2024)
- [3] *Hash Pointers and Data Structures*. URL: https://www.zhaohuabing.com/post/2018-05-12-cryptocurrency_week1_hash_pointer_and_data_structures/ (visitato il 14/03/2024)
- [4] *What is blockchain?*
URL: <https://opensea.io/learn/blockchain/what-is-blockchain>
(visitato il 14/03/2024)
- [5] *What is a smart contract?* URL: <https://opensea.io/learn/blockchain/what-is-a-smart-contract> (visitato il 15/03/2024)
- [6] *What is an NFT?*
URL: <https://opensea.io/learn/nft/what-are-nfts>
(visitato il 16/03/2024)
- [7] Adeniyi E. A., Falola P. B., Maashi M. S., Aljebreen M., Bharany S.
Secure sensitive data sharing using RSA and ElGamal cryptographic algorithms with hash functions (consultato il 17/03/2024)
- [8] *What is a crypto wallet?*
URL: <https://opensea.io/learn/web3/what-is-crypto-wallet>
(visitato il 17/03/2024)

- [9] *What is Etherscan?*
URL: <https://opensea.io/learn/blockchain/what-is-etherscan>
(visitato il 18/03/2024)
- [10] *Etherscan*
URL: <https://etherscan.io/> (utilizzato il 07/10/2023)
- [11] *PolygonScan*
URL: <https://polygonscan.com/> (utilizzato il 15/10/2023)
- [12] *What are EVM Compatible Blockchains? A Guide to the Ethereum Virtual Machine.* URL: <https://blog.thirdweb.com/evm-compatible-blockchains-and-ethereum-virtual-machine/>
(visitato il 10/03/2024)
- [13] *Update Report: Energy Efficiency and Carbon Footprint of the Polygon Blockchain.* URL: <https://carbon-ratings.com/dl/polygon-update-2022>
(scaricato il 12/10/2023)
- [14] *Polygon for the Planet.* URL: <https://polygon.technology/sustainability> (visitato il 12/10/2023)
- [15] *Certificato dell'opera "Between the Sky and Sea"*
URL: <https://www.pinterest.it/pin/344806915203383539/>
(visitato il 05/04/2024)
- [16] *Dress from a Dream: Gold.* URL: <https://unxd.com/drops/dolce-gabbana-collezione-genesi/dress-from-a-dream-gold>
(visitato il 02/12/2023)
- [17] *Alchemy*
URL: <https://www.alchemy.com/> (utilizzato il 07/10/2023)
- [18] *MetaMask*
URL: <https://metamask.io/> (utilizzato il 04/10/2023)

- [19] *Polygon Mumbai Faucet*
URL: <https://www.alchemy.com/faucets/polygon-mumbai>
(utilizzato il 14/10/2023)
- [20] *PolygonScan Api Key Generator*
URL: <https://polygonscan.com/myapikey> (utilizzato il 09/12/2023)
- [21] *Pinata*
URL: <https://www.pinata.cloud/> (utilizzato il 09/10/2023)
- [22] *OpenSea*
URL: <https://opensea.io/learn> (utilizzato il 15/12/2023)