POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Analisi di Applicazioni Blockchain in Ambito Sanitario

Relatore
Prof. Danilo Bazzanella
Correlatore
Enrico D'acquisto

Candidato
Riccardo Bertolo

Anno Accademico 2022-2023

Alla mia Famiglia
A Eleonora
A tutti i miei Amici

Sommario

Una tecnologia dirompente è una qualsiasi innovazione che potrebbe sconvolgere il mercato esistente, cambiando radicalmente il modo in cui le aziende lavorano e le persone vivono.
Queste tecnologie rendono dei servizi facilmente accessibili e convenienti ai loro consumatori, riuscendo a soddisfarne una base molto ampia.
La Blockchain può essere considerata, come Internet al suo tempo, una tecnologia dirompente, in grado di alimentare un' era di invenzioni in nuove dimensioni.

L'azienda italiana *TurinTech*, dove la tesi è stata progettata, sta valutando la fattibilità di possibili implementazioni delle tecnologie Blockchain in diversi settori, tra cui un sistema per migliorare il settore sanitario, argomento di questo lavoro.
Lo scopo principale di questa tesi è la creazione di cartelle cliniche per i pazienti sotto forma di *non-fungible token NFT*, concedendo ai singoli individui di avere il completo controllo e di essere gli unici proprietari delle proprie informazioni cliniche.
Per far questo è stata definita un'organizzazione sanitaria di base, specificando anche altre diverse tipologie di token, partendo da quelli amministrativi e arrivando a token di medici e operatori.

L'infrastruttura alla base del progetto è la definizione di una blockchain nazionale, che gli enti e aziende possono sfruttare per beneficiare di tutte le principali caratteristiche offerte da questa tecnologia.

Abstract

A disruptive technology is any innovation that could disrupt the existing market, fundamentally changing the way companies work and people live.
These technologies make services easily accessible and convenient to their consumers, managing to satisfy a very broad base.
Blockchain can be considered, like the Internet in its time, a disruptive technology, capable of fueling an era of invention in new dimensions.

The Italian company *TurinTech*, where the thesis was designed, is evaluating the feasibility of possible implementations of Blockchain technologies in different sectors, including a system to improve the healthcare sector, the topic of this work.
The main purpose of this thesis is the creation of medical records for patients in the form of *non-fungible token NFT*, allowing individuals to have complete control and be the sole owners of their own clinical information.
To do this, a basic healthcare organization was defined, also specifying other different types of tokens, starting from administrative ones and arriving at tokens of doctors and operators.

The infrastructure underlying the project is the definition of a national blockchain, which organizations and companies can exploit to benefit from all the main features offered by this technology.

Ringraziamenti

Desidero esprimere la mia profonda gratitudine a tutti coloro che mi hanno accompagnato durante il percorso di realizzazione di questa tesi.

Un ringraziamento speciale va alla mia famiglia e alla mia ragazza per il completo sostegno durante questi anni di studio, senza il quale non sarei riuscito a raggiungere questo traguardo.

Vorrei inoltre ringraziare l'azienda TurinTech per avermi concesso l'opportunità di condurre la mia tesi presso di loro e il mio relatore, il Professore Danilo Bazzanella per i suggerimenti e le preziose indicazioni che mi hanno guidato durante questi mesi di ricerca.

Infine, desidero ringraziare i miei amici, che hanno condiviso con me gioie e sfide durante il percorso di studio: la loro presenza, affetto e incoraggiamento hanno reso questo cammino più piacevole e stimolante.

Indice

Elenco delle figure

Capitolo 1

Introduzione

In questi ultimi anni, la diffusione della nuova tecnologia Blockchain in diversi settori ha preso piede, cambiando significativamente il modo di lavorare di molte aziende che l'hanno adottata.
L'obiettivo di questa tesi è presentare come la tecnologia Blockchain potrebbe modificare e migliorare il settore della sanità nazionale. Lo studio è stato svolto presso *TurinTech* (12), azienda italiana che sta attentamente esaminando e investendo in questa innovativa tecnologia al fine di applicarla in vari settori di interesse.

Nel secondo capitolo, per spiegare i concetti successivi, è stato necessario definire dei cenni di crittografia su cui si basano tutte le blockchain esistenti, per garantire sicurezza e immutabilità dei dati archiviati all'interno della catena.
In particolare sono stati illustrati i principali crittosistemi a chiave pubblica, per poi arrivare a definire le funzioni Hash e alcuni algoritmi di firma digitale, concetti necessari per gli studi successivi.

Nel terzo capitolo viene presentata la storia della Blockchain, facendo un breve riassunto di tutte le scoperte e invenzioni che sono state fatte negli scorsi decenni e che hanno permesso la definizione del presente concetto di Blockchain.
Il primo vero utilizzo mondiale è stato con la nascita di Bitcoin (56) nel 2008, ma senza le basi gettate precedentemente, non sarebbe stato possibile. Concetti come *Blind Signatures*, *Timestamp* e *Smart Contracts* sono stati essenziali per lo

svilupparsi di questa nuova tecnologia.

Nel quarto capitolo è introdotto il concetto generale di Blockchain, definendone le caratteristiche e paragonandola alle altre tecnologie già presenti.
Sono anche elencate le principali tipologie, soffermandosi sulle differenze fondamentali e i diversi campi di utilizzo, in base agli scopi necessari.
E' anche definito un concetto molto importante all'interno di questa tecnologia, il metodo di consenso da parte dei nodi presenti nella rete, che può essere di diverse tipologie, che funzionano meglio in determinati contesti rispetto ad altri.

Nel quinto capitolo sono esaminate l'architettura e il funzionamento interno di una blockchain, prendendo in consideranzione, come riferimento, la blockchain di Bitcoin, partendo dai modelli matematici e crittografici che ne stanno alla base e tutti i possibili algoritmi per la generazione delle chiavi e indirizzi. Successivamente è analizzata nel dettaglio la struttura dei blocchi e delle transazioni che compongono la catena, concludendo con alcuni concetti organizzativi essenziali per garantire il corretto funzionamento dell'intero sistema.

Nel sesto capitolo sono introdotti nuovi concetti innovativi nel campo della Blockchain: *Smart Contract*, *Token* e *Decentralized Autonomous Organizations (DAO)*, con la creazione di *Ethereum*, che hanno permesso di passare dalla Blockchain 1.0 alla Blockchain 2.0.
Con queste invenzioni, l'utilizzo della Blockchain si è distaccato dallo scopo esclusivamente finanziario, espandendosi a tantissimi settori diversi.

Nel settimo capitolo, dopo una breve introduzione sui metodi di archiviazione usati dalla sanità odierna, è descritta una possibile idea per la creazione di una blockchain nel settore pubblico nazionale, enunciandone esclusivamente alcuni concetti di base, non essendo il principale obiettivo di questo elaborato.
A seguire è definito nel dettaglio il principale progetto, quindi la creazione di un sistema per migliorare il settore della sanità, con l'introduzione di diverse tipologie di token, principalmente i token delle cartelle cliniche per i pazienti.

Quindi, dopo una prima parte dove si sono analizzate le basi per la creazione di una DAO sulla blockchain nazionale, si sono definiti i contratti per generare i token di amministratori, operatori sanitari e pazienti. Inoltre, sono state affrontate alcune proposte di utilizzo pratico, descrivendone i metodi e i passaggi principali.

Capitolo 2

Cenni di Crittografia

La crittografia, attraverso tecniche di cifratura, ha lo scopo di rendere incomprensibile un messaggio a chi non è autorizzato a leggerlo.
L'utilizzo della crittografia nella Blockchain è legato a un fattore di privacy e anonimato, in quanto tutte le transazioni all'interno dei blocchi sono pubbliche, e quindi viene utilizzata per cifrarle.
Per procedere con la definizione dell'architettura e del funzionamento di una blockchain, è necessario definire alcuni concetti crittografici che ne stanno alla base.

2.1 Sistemi a chiave privata

Nei sistemi a chiave privata, o a chiavi simmetriche, ogni utente possiede una chiave, che viene usata sia per cifrare il messaggio, sia per decifrarlo (38).
Quindi, quando due utenti vogliono scambiarsi un messaggio, il mittente deve cifrare il testo in chiaro usando una chiave privata, inviare il cifrato al destinatario attraverso il canale pubblico e la chiave privata attraverso un canale privato. Il destinatario userà la chiave privata ottenuta per decifrare il messaggio e ottenere il testo in chiaro. Una volta che ogni coppia di utenti si è scambiata la chiave privata, può usare sempre la stessa, per cifrare e decifrare i messaggi che si inviano: in un sistema a chiave privata con n utenti, ogni coppia deve avere una chiave, quindi il

numero di chiavi necessarie sarà

$$\binom{n}{2} = \frac{n(n-1)}{2} \sim \frac{n^2}{2}.$$

2.2 Sistemi a chiave pubblica

I sistemi a chiave pubblica, o a chiavi asimmetriche, rispetto a quelli a chiave privata, hanno la differenza che ogni utente possiede due chiavi anzichè una sola, una da mantenere privata, l'altra invece da pubblicare. Quindi in un sistema con n utenti, il numero di chiavi necessarie sarà solo $2n$ (38).

Nel processo di invio di un messaggio, usando un sistema a chiave pubblica, il mittente cifra il messaggio da inviare con la chiave pubblica del destinatario; in questo modo può inviare il messaggio cifrato attraverso un canale pubblico. Una volta che il destinatario riceverà il messaggio, per decifrarlo, basterà applicare la sua chiave privata relativa alla pubblica con cui il mittente ha cifrato il messaggio.

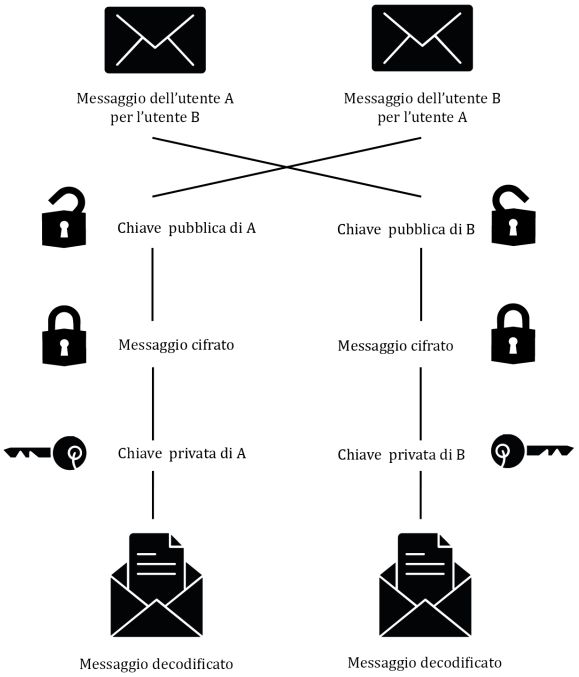

Figura 2.1: Schema di invio di messaggi tramite un sistema a chiave pubblica.

Il documento può essere anche firmato digitalmente, per verificarne l'identità e evitare manomissioni, rilevando una eventuale modifica non autorizzata durante l'invio sul canale pubblico.

Anche se il numero di chiavi nella crittografia asimmetrica è molto minore, essa non ha reso obsoleta la crittografia simmetrica e spesso possono essere utilizzate insieme, la prima per lo scambio di chiavi per poi utilizzare la seconda.

2.3 Sistema RSA

Il primo e più famoso dei sistemi a chiave pubblica è RSA, sviluppato nel 1978 da Ronald Rivest, Adi Shamir e Leonard Adleman, motivo del nome che deriva dalle iniziali dei suoi inventori.
Il procedimento che ne sta alla base è il prodotto di numeri primi; infatti, dati due numeri primi, è facile stabilire il loro prodotto, ma dato un certo numero, è molto difficile individuare i numeri primi che lo determinano una volta moltiplicati.
Questo garantisce la sicurezza alla base dei sistemi a chiave pubblica: infatti, conoscendo la chiave privata, è facile calcolare la chiave pubblica, ma conoscendo la pubblica non è computazionalmente fattibile risalire alla chiave privata (68).

L'algoritmo crittografico per definire la coppia di chiavi è il seguente:

- ogni utente sceglie due numeri primi grandi distinti p e q,
- calcola $N = p * q$ e $\phi(N) = N - p - q - 1$,
- sceglie $e \in Z^*_{\phi(N)}$ e calcola d tale che $e * d \equiv 1 \mod (\phi(N))$.

La coppia (N, e) costituisce la chiave pubblica dell'utente, mentre la coppia $(\phi(N), d)$ è la chiave privata.

Se quindi qualcuno vuole inviare un messaggio x ad un utente A, basterà prendere la chiave pubblica di A e calcolare $f_A(x) = x^e \mod (N)$.
Una volta che poi A ottiene $f_A(x)$, per decifrarlo, userà la sua chiave privata con la funzione di decifratura $f_A^{-1}(x) = x^d \mod (N)$.

$$f_A^{-1}(f_A(x)) = f_A^{-1}(x^e \mod (N)) = x^{e*d} \mod (N) = x^{1+k*\phi(N)} \mod (N) = \\ = x \mod (N).$$

In questo modo A otterrà il testo in chiaro x, sia se il messaggio x è coprimo con N, sia se non lo è, per il Teorema di Eulero generalizzato (38).

Il sistema, però, è a rischio quando il messaggio da inviare viene codificato in un numero x che non è coprimo con N, in quanto p o q dividono x e quindi calcolando il minimo comune divisore tra il messaggio x e N, si riesce a risalire alla chiave privata dell'utente. Questo però succede con una probabilità trascurabile se vengono scelti dei numeri p e q molto grandi, infatti i possibili numeri minori di N non coprimi con N sono $p + q - 1$ e quindi la probabilità di sceglierne uno è

$$\frac{p+q-1}{N} = \frac{1}{q} + \frac{1}{p} - \frac{1}{pq}.$$

2.4 Sistema di ElGamal

Il criptosistema di ElGamal è un altro sistema a chiave pubblica che però si basa sulla difficoltà di risolvere il problema del logaritmo discreto, quindi ricavare l'esponente k conoscendo g e g^k (37).
L'algoritmo crittografico per definire la coppia di chiavi è il seguente:

- per l'intero sistema si sceglie un primo grande p e un generatore g di Z_p^*,
- ogni utente quindi sceglierà la propria chiave privata come $x \in Z_{p-1}$ e renderà pubblica g^x.

Quando l'utente A vuole inviare un messaggio M all'utente B, che ha chiave privata y e chiave pubblica $b = g^y$, dovrà:

- A sceglie a caso $k \in Z_{p-1}$ e invia a B la coppia (g^k, Mb^k),
- B può usare la sua chiave privata y e calcolare $(g^k)^y = (g^y)^k = b^k$ e quindi calcolare l'inverso b^{-k},
- a questo punto moltiplica il secondo termine della coppia ricevuta da A per b^{-k} ricavando $M = Mb^k b^{-k}$.

2.5 ECC, Elliptic-curve cryptography

La crittografia basata su curve ellittiche, definite su campi finiti, è una crittografia a chiave pubblica (38).
Una curva ellittica è una curva algebrica, che con l'operazione somma di punti appartenenti alla curva è un gruppo abeliano, con elemento neutro il punto all'infinito O.

Un gruppo abeliano è un gruppo la cui operazione binaria interna gode della proprietà commutativa; infatti la somma di punti su una curva ellittica gode della proprietà commutativa.

In generale una curva ellittica si definisce su un campo finito F_p e ha la forma:

$$y^2 = x^3 + ax + b \tag{2.1}$$

e quindi tutti i calcoli interni alla curva ellittica, per esempio le somme di punti, devono essere fatti in aritmentica modulare.

Dati due punti appartenenti ad una curva ellittica $P = (x_1, y_1)$ e $Q = (x_2, y_2)$, si può definire $R = P + Q = Q + P = (x_3, y_3)$, sempre appartenente alla curva ellittica.

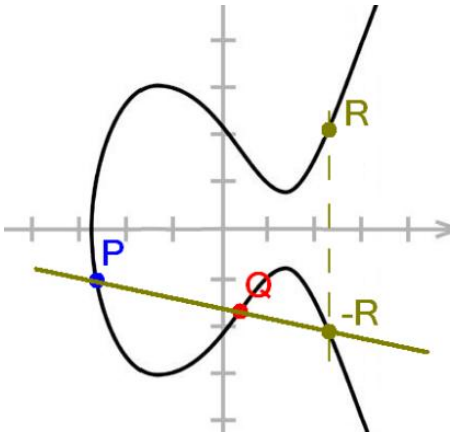

Figura 2.2: Somma dei punti P e Q di una curva ellittica.

Per calcolare graficamente la somma di due punti appartenenti alla curva ellittica, si considera la retta secante la curva nei punti da sommare, P e Q; essa incontrerà la curva in un terzo punto $(-R = (x_3, -y_3))$, che è il simmetrico, rispetto l'asse x, di $R = P + Q$.
Matematicamente $R = (x_3, y_3) = (\lambda^2 - x_1 - x_2, -(\lambda x_3 + y_1 - \lambda x_1))$, dove

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \frac{x_1^2 + x_1 x_2 + x_2^2 + a}{y_2 + y_1}.$$

Se si volesse calcolare la somma di un punto con se stesso, anche più volte, quindi calcolare un suo multiplo, il ragionamento sarebbe analogo. Per esempio, se si sta calcolando $2Q$, graficamente, si procederà tracciando la tangente alla curva nel punto Q, e analogamente alla somma di due punti distinti essa incontrerà la curva in un altro punto, simmetrico, rispetto l'asse x, di $2Q$.

Utilizzando il sito (10) si può mostrare la somma dei punti di una determinata curva ellittica.

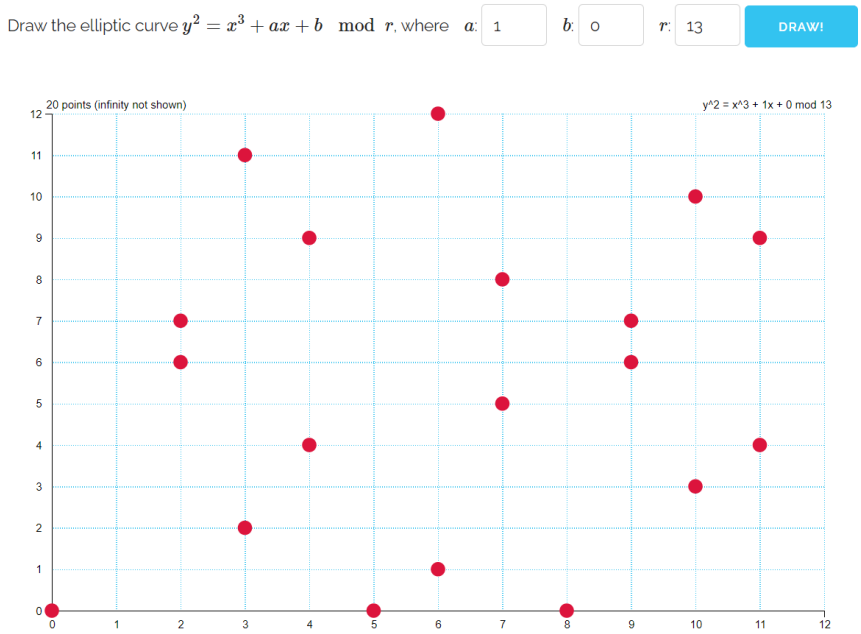

Figura 2.3: Esempio curva ellittica $y^2 = x^3 + x$ con $p = 13$.

+	∞	(0,0)	(2,6)	(2,7)	(3,2)	(3,11)	(4,4)	(4,9)	(5,0)	(6,1)	(6,12)	(7,5)	(7,8)	(8,0)	(9,6)	(9,7)	(10,3)	(10,10)	(11,4)	(11,9)
∞	∞	(0,0)	(2,6)	(2,7)	(3,2)	(3,11)	(4,4)	(4,9)	(5,0)	(6,1)	(6,12)	(7,5)	(7,8)	(8,0)	(9,6)	(9,7)	(10,3)	(10,10)	(11,4)	(11,9)
(0,0)	(0,0)	∞	(7,5)	(7,8)	(9,7)	(9,6)	(10,3)	(10,10)	(8,0)	(11,9)	(11,4)	(2,6)	(2,7)	(5,0)	(3,11)	(3,2)	(4,4)	(4,9)	(6,12)	(6,1)
(2,6)	(2,6)	(7,5)	(9,7)	∞	(11,4)	(7,8)	(8,0)	(6,1)	(10,10)	(9,6)	(4,4)	(3,2)	(0,0)	(4,9)	(2,7)	(6,12)	(5,0)	(11,9)	(10,3)	(3,11)
(2,7)	(2,7)	(7,8)	∞	(9,6)	(7,5)	(11,9)	(6,12)	(8,0)	(10,3)	(4,9)	(9,7)	(0,0)	(3,11)	(4,4)	(6,1)	(2,6)	(11,4)	(5,0)	(3,2)	(10,10)
(3,2)	(3,2)	(9,7)	(11,4)	(7,5)	(4,4)	∞	(10,10)	(3,11)	(6,1)	(7,8)	(5,0)	(6,12)	(2,6)	(11,9)	(0,0)	(10,3)	(4,9)	(9,6)	(8,0)	(2,7)
(3,11)	(3,11)	(9,6)	(7,8)	(11,9)	∞	(4,9)	(3,2)	(10,3)	(6,12)	(5,0)	(7,5)	(2,7)	(6,1)	(11,4)	(10,10)	(0,0)	(9,7)	(4,4)	(2,6)	(8,0)
(4,4)	(4,4)	(10,3)	(8,0)	(6,12)	(10,10)	(3,2)	(9,6)	∞	(7,8)	(2,6)	(6,1)	(5,0)	(11,4)	(2,7)	(9,7)	(4,9)	(3,11)	(0,0)	(11,9)	(7,5)
(4,9)	(4,9)	(10,10)	(6,1)	(8,0)	(3,11)	(10,3)	∞	(9,7)	(7,5)	(6,12)	(2,7)	(11,9)	(5,0)	(2,6)	(4,4)	(9,6)	(0,0)	(3,2)	(7,8)	(11,4)
(5,0)	(5,0)	(8,0)	(10,10)	(10,3)	(6,1)	(6,12)	(7,8)	(7,5)	∞	(3,2)	(3,11)	(4,9)	(4,4)	(0,0)	(11,4)	(11,9)	(2,7)	(2,6)	(9,6)	(9,7)
(6,1)	(6,1)	(11,9)	(9,6)	(4,9)	(7,8)	(5,0)	(2,6)	(6,12)	(3,2)	(4,4)	∞	(3,11)	(10,10)	(9,7)	(8,0)	(2,7)	(7,5)	(11,4)	(0,0)	(10,3)
(6,12)	(6,12)	(11,4)	(4,4)	(9,7)	(5,0)	(7,5)	(6,1)	(2,7)	(3,11)	∞	(4,9)	(10,3)	(3,2)	(9,6)	(2,6)	(8,0)	(11,9)	(7,8)	(10,10)	(0,0)
(7,5)	(7,5)	(2,6)	(3,2)	(0,0)	(6,12)	(2,7)	(5,0)	(11,9)	(4,9)	(3,11)	(10,3)	(9,7)	∞	(10,10)	(7,8)	(11,4)	(8,0)	(6,1)	(4,4)	(9,6)
(7,8)	(7,8)	(2,7)	(0,0)	(3,11)	(2,6)	(6,1)	(11,4)	(5,0)	(4,4)	(10,10)	(3,2)	∞	(9,6)	(10,3)	(11,9)	(7,5)	(6,12)	(8,0)	(9,7)	(4,9)
(8,0)	(8,0)	(5,0)	(4,9)	(4,4)	(11,9)	(11,4)	(2,7)	(2,6)	(0,0)	(9,7)	(9,6)	(10,10)	(10,3)	∞	(6,12)	(6,1)	(7,8)	(7,5)	(3,11)	(3,2)
(9,6)	(9,6)	(3,11)	(2,7)	(6,1)	(0,0)	(10,10)	(9,7)	(4,4)	(11,4)	(8,0)	(2,6)	(7,8)	(11,9)	(6,12)	(4,9)	∞	(3,2)	(10,3)	(7,5)	(5,0)
(9,7)	(9,7)	(3,2)	(6,12)	(2,6)	(10,3)	(0,0)	(4,9)	(9,6)	(11,9)	(2,7)	(8,0)	(11,4)	(7,5)	(6,1)	∞	(4,4)	(10,10)	(3,11)	(5,0)	(7,8)
(10,3)	(10,3)	(4,4)	(5,0)	(11,4)	(4,9)	(9,7)	(3,11)	(0,0)	(2,7)	(7,5)	(11,9)	(8,0)	(6,12)	(7,8)	(3,2)	(10,10)	(9,6)	∞	(6,1)	(2,6)
(10,10)	(10,10)	(4,9)	(11,9)	(5,0)	(9,6)	(4,4)	(0,0)	(3,2)	(2,6)	(11,4)	(7,8)	(6,1)	(8,0)	(7,5)	(10,3)	(3,11)	∞	(9,7)	(2,7)	(6,12)
(11,4)	(11,4)	(6,12)	(10,3)	(3,2)	(8,0)	(2,6)	(11,9)	(7,8)	(9,6)	(0,0)	(10,10)	(4,4)	(9,7)	(3,11)	(7,5)	(5,0)	(6,1)	(2,7)	(4,9)	∞
(11,9)	(11,9)	(6,1)	(3,11)	(10,10)	(2,7)	(8,0)	(7,5)	(11,4)	(9,7)	(10,3)	(0,0)	(9,6)	(4,9)	(3,2)	(5,0)	(7,8)	(2,6)	(6,12)	∞	(4,4)

Figura 2.4: Somma dei punti della curva in 2.3.

Sfruttando la crittografia basata sulle curve ellittiche, i sistemi già noti possono essere ridefiniti: come esempio si vede il sistema di ElGamal (38).
Si supponga di aver fissato una curva ellittica E, su un campo F_q con q grande e primo e sia noto un punto B della curva.
Ogni utente deve scegliere un numero casuale a, che rappresenterà la sua chiave privata, e lo moltiplichi per B, ottenendo la sua chiave pubblica aB.

Ora, se l'utente A vuole mandare un messaggio M a B, dovrà:

- A codifica M in un punto della curva P_M e sceglie un numero casuale k,
- A manda a B la coppia $(kB, P_M + k(a_B B))$,
- B moltiplica per a_B, la sua chiave privata, il primo termine della coppia ricevuta e sottrae al secondo, ottenendo P_M,
- B decodifica P_M, secondo il metodo definito e ottiene M.

2.6 Hash

Dato Σ un alfabeto e Σ^* l'insieme di tutte le parole (di lunghezza qualsiasi) ottenibili da Σ, definiamo funzione Hash, una funzione

$$h : \Sigma^* \to \Sigma^n, \tag{2.2}$$

con valore di n fissato.
Tipicamente nelle applicazioni $\Sigma = \{0,1\}$ e n=160,256,384 o 512.
L'immagine di x, $h(x)$ è detta Hash o *digest* e la funzione h, ovviamente, non potrà essere iniettiva, in quanto gli elementi presenti nel codominio sono un numero finito, rispetto agli elementi presenti nel dominio.
Un' importante proprietà che una funzione Hash deve avere è l'effetto a valanga, quindi cambiando anche solo un carattere nell' input, l'output dovrà cambiare radicalmente.

- SHA-256(Testo): ebb9e60cfec069e7f34394b51e0744429e8cbc3c0adc22485e8cb4632f912944,
- SHA-256(Testa): c5a7f22c5e8bed9776556c0521dd5940e44d965b5b6b66e76ff64b3bcb769925,
- SHA-256(Teste): 89f308210c7c7820bad0974f31e751bfa433d2066a93e808947c3188dedba6e3.

Inoltre deve essere priva di collisioni, quindi dato $a \in \Sigma$, deve essere computazionalmente infattibile trovare $b \in \Sigma$ diverso da a, tale per cui $h(a) = h(b)$.

SHA, Secure Hash Algorithm, è una famiglia di diverse funzioni crittografiche di hashing sviluppate a partire dal 1993 dalla National Security Agency (NSA) e pubblicate dal NIST come standard federale dal governo degli USA (38).

Il primo standard SHA definito è stato lo SHA-1, che produce un digest del messaggio di soli 160 bit, mentre gli SHA-224, SHA-256, SHA-384 e SHA-512, indicati generalmente con il nome SHA-2, producono un digest di lunghezza di bit uguale al valore definito nella loro sigla.

La sicurezza dello SHA-1, però, è stata invalidata negli anni e dato che gli SHA-2 hanno un algoritmo simile ad esso, anche se al momento non si sono ancora verificate violazioni, potrebbero essere compromessi nel futuro.
Nel 2012 il NIST ha proclamato come vincitore del contest per la creazione del nuovo SHA-3 un sottoinsieme della famiglia di funzioni crittografiche Keccak (5).

La famiglia di funzioni crittografiche SHA, però, non è l'unica ad esistere; infatti, per esempio, altri algoritmi di hashing sono i RIPEMD, alternativa europea agli algoritmi americani del tempo. Il primo algoritmo ideato fu il RIPEMD-128, che produce un output di 128 bit, e successivamente furono prodotte altre versioni con output più lunghi, per esempio, il più famoso è il RIPEMD-160 (13).

- RIPEMD-160(Testo): 3592108b26d93d20edd57ee0e47a0c950f4fa17f,
- RIPEMD-160(Testa): c14bf5378b7c4f3833dd6e516d342717c14d4b27,
- RIPEMD-160(Teste): acdd533879f2cd75df696ee8857c8668f1f32f5a.

Bitcoin, per esempio, per la creazione degli indirizzi, utilizza la funzione SHA256 e RIPEMD-160.

2.7 Firme Digitali

Una firma digitale, proprio come una firma nella vita reale, serve a garantire il mittente di un messaggio o comunque l'identità di chi compie una certa operazione.

I sistemi a chiave pubblica hanno come lato positivo il numero ridotto di chiavi e soprattutto evitano il problema dello scambio di chiavi, ma chiunque può cifrare i messaggi usando la chiave pubblica del destinatario, quindi deriva la necessità di autenticare il mittente del messaggio (38).

Uno schema generale per la firma digitale in un sistema a chiave pubblica può essere definito come segue.

- A vuole inviare un messaggio M a B, quindi tramite la funzione pubblica di B, f_B, gli invia $f_B(M)$. Questo però sarebbe in grado di inviarlo chiunque,

quindi, per firmarsi, invia anche la coppia $(f_B(f_A^{-1}(s_A||h(M))), s_A)$, dove s_A può essere un testo che contiene il nome di A.

- A è l'unico che può creare il primo termine della coppia, dato che è l'unico a conoscere f_A^{-1} e concatena anche $h(M)$, digest del messaggio M, per evitare che B si spacci per lui quando invia un altro messaggio ad un altro utente.
- B quindi applica $f_A f_B^{-1}$ al primo termine della coppia ricevuta e verifica che coincida con il secondo termine.

2.8 DSA, Digital Signature Algorithm

Il sistema di firma digitale di ElGamal raramente viene utilizzato nella pratica. Una variante molto usata è invece la firma digitale DSA, Digital Signature Algorithm, perché si serve di firme più corte ed è più efficiente (38).

L'algoritmo crittografico per definire la coppia di chiavi, a 1024 bit, è il seguente:

- si genera un primo p con $2^{1023} < p < 2^{1024}$,
- si cerca un primo q divisore di $p-1$ tale che $2^{159} < q < 2^{160}$,
- si cerca un α di ordine q (α genera un sottogruppo di Z_p^* con q elementi),
- si sceglie a caso un numero d tale che $0 < d < q$,
- si calcola $\beta = \alpha^d \mod (p)$,
- la chiave pubblica è $k_{pub} = (p, q, \alpha, \beta)$ e quella privata è $k_{priv} = d$.

Il protocollo di firma digitale tramite DSA, quindi, è il seguente:

- si sceglie un numero casuale k tale che $0 < k < q$,
- si calcola $r \equiv (\alpha^k \mod (p)) \mod (q)$,
- si calcola $s \equiv (h(M) + dr)k^{-1} \mod (q)$,
- la firma del messaggio M quindi è la coppia (r, s).

Per procedere alla verifica invece:

- si calcola $w = s^{-1} \mod (q) = k/(h(M) + dr) \mod (q)$,
- si calcola $u_1 = wh(M) \mod (q)$,
- si calcola $u_2 = wr \mod (q)$,
- si calcola $v = (\alpha^{u_1}\beta^{u_2} \mod (p)) \mod (q)$,
- la firma è autentica se v, così calcolato, coincide con $r \equiv (\alpha^k \mod (p)) \mod (q)$.

Per il sistema DSA esistono standard anche con un numero maggiore di bit, come NIST 800-57 raccomanda lunghezze di 2048 o 3072 bit, in ogni caso tutti multipli di 64 (20).

2.9 ECDSA, Elliptic Curve Digital Signature Algorithm

Sulle curve ellittiche si basa il ECDSA, Elliptic Curve Digital Signature Algorithm, algoritmo per la firma digitale, variante del DSA.
L'ECDSA si fonda sulla matematica dei gruppi ciclici delle curve ellittiche su campi finiti e sulla difficoltà del problema del logaritmo discreto delle curve ellittiche (17).
I parametri fissi e necessari per l'algoritmo sono: la curva ellittica usata, un punto G della curva, detto punto generatore, il numero n di punti della curva e una funzione hash h, usata nel processo. Ogni utente, inoltre, deve avere la propria chiave privata d, intero, e chiave pubblica $P = dG$, punto della curva (38).

Il protocollo di firma è il seguente:

- il mittente calcola il digest $h = h(M)$ del messaggio in chiaro M,
- sceglie a caso un valore $k \in Z_n$,
- calcola il punto della curva $kG = (x, y)$,
- fissa $r = x \mod (n)$ e calcola $s = (h + rd)/k \mod (n)$,

- la firma da allegare al messaggio è la coppia (r, s).

Di seguito si definisce l'algoritmo di verifica della firma.

- Calcolare l'inverso del secondo termine della firma, determinando $w = s^{-1} = k/(h + rd) \mod (n)$,
- calcolare $u = wh \mod (n)$ e $v = wr \mod (n)$,
- calcolare il punto $Q = uG + vP$, dove $P = dG$ è la chiave pubblica del firmatario,
- accettare la firma solo se la prima coordinata del punto Q, ridotta modulo n, coincide con r.

Lo schema di firma ECDSA permette inoltre di recuperare la chiave pubblica dal messaggio firmato e la firma.
Dalla firma (r, s), però, il processo può avere come risultato 0, 1 o 2 possibili punti della curva ellittica validi, quindi chiavi pubbliche, per la firma definita. Per ovviare a questa ambiguità, alcune implementazioni più complesse dell'ECDSA aggiungono un bit v alla firma durante il processo, ottenendo la forma (r, s, v); in questo modo la chiave pubblica può essere ripristinata con sicurezza (17).

Capitolo 3

Storia della Blockchain

Il concetto di Blockchain, sebbene sia stato formulato molto prima dell'era dell'informatica, risale al quindicesimo secolo, grazie ad una popolazione della Micronesia, un arcipelago situato a nord dell'Australia (53).
La gente utilizzava grandi pietre di calcare, chiamate Rai, come forma di moneta; di solito, il valore delle pietre era direttamente proporzionale alle loro dimensioni, ma poteva variare anche a seconda della difficoltà di estrazione o della lunghezza del processo di trasporto, considerando la loro elevata fragilità. Altri fattori, come la cava di provenienza o un rischio elevato durante l'estrazione, potevano influenzare il loro valore.
Poiché molte di queste pietre erano così voluminose da renderne impossibile il trasporto, la popolazione aveva bisogno di un metodo per tracciare gli scambi. A tal scopo, fu creato un registro, chiamato libro mastro, in cui venivano registrate tutte le transazioni riguardanti i passaggi di proprietà delle pietre.
Inoltre, ogni individuo possedeva una copia del libro, che doveva essere aggiornato ogni volta che avveniva un cambio di proprietà (33).

3.1 Blind Signatures 1982

Nel 1982, David Chaum, un crittografo e informatico americano, pubblicò l'articolo *Blind Signatures for Untraceable Payments*, dove introdusse il concetto di *blind signatures* in crittografia (35).

Il protocollo proposto aveva lo scopo di apporre la firma ad un documento, senza venire a conoscenza del contenuto, garantendo così la privacy e l'anonimato del mittente.

Il processo generale, per un generico documento m, può essere riassunto come segue.
Per prima cosa sono da considerare tre funzioni, che definiranno il sistema crittografico che sta alla base del processo.

- La funzione di firma s' e la sua inversa s pubblica, tali per cui $s(s'(m)) = m$. Anche conoscenso s, non si riuscirà a ricavare s'.
- La funzione del mittente c e la sua inversa c', conosciute esclusivamente dal mittente e tali che $c'(s'(c(m))) = s'(m)$.
  Ovviamente conoscendo $c(m)$ e la funzione s', non deve essere possibile ricavare m, altrimenti il marcatore riuscirebbe a ricavarsi il messaggio da firmare e non sarebbe più un sistema blid signatures.
- La funzione di controllo r, per verificare la validità della firma applicata.

Si può assumere che A scriva il messaggio m e che B debba firmarlo, senza doverne conoscere il contenuto.

- A scrive m, genera $r(m)$, che rende pubblico e genera $c(m)$ che invia a B per farlo firmare.
- B firma $c(m)$ e quindi manda a A $s'(c(m))$.
- A applica la funzione c' e ottiene il messaggio firmato $c'(s'(c(m))) = s'(m)$.
- Chiunque può verificare la firma applicando le funzioni r e s pubbliche, $r(s(s'(m))) = r(m)$ e confronta questo valore con quello pubblicato inizialmente da A.

3.2 DigiCash 1989

Nel 1989, sempre David Chaum sviluppò *DigiCash*, società pioniera nell'ambito dei pagamenti elettronici anonimi e della crittografia per la tutela della privacy

finanziaria.
L'azienda definì il concetto di denaro elettronico, creando una forma di moneta digitale, *eCash*, che permetteva agli utenti di fare pagamenti online in totale anonimato, sfruttando protocolli crittografici (35).

L'ostacolo principale dell'adozione di eCash, però, era la centralizzazione, in quanto si affidava alle banche. L'obbligo di aprire un conto corrente in una banca che accettasse depositi in eCash, quindi, era un vincolo importante, in quanto i fondi non avevano alcun valore se l'utente non riusciva a trovare una banca che accettasse tale metodo di pagamento (59).
eCash riuscì ad ottenere l'adozione in banche in alcuni Stati del mondo, come per esempio Germania e Svizzera, ma nel 1998 la società presentò istanza di fallimento (42).

Nonostante la chiusura dell'azienda, le idee e il lavoro sviluppato da David Chaum con DigiCash hanno continuato a esercitare una grande influenza nell'ambito della crittografia e dei sistemi di pagamento elettronici.

3.3 Timestamp 1991

Nel 1991, Stuart Haber e W. Scott Stornetta, due ricercatori nel campo della crittografia, pubblicarono *How to Time-Stamp a Digital Document*, un sistema che permetteva di marcare temporalmente i documenti digitali, con lo scopo quindi di certificare, in un preciso momento, il contenuto di essi.
Questo era fondamentale per preservarne l'integrità e l'autenticità nel corso del tempo, in quanto, in caso di modifiche, esse avrebbero alterato l'impronta crittografica del documento invalidando il timestap.
Nella pubblicazione, introdussero anche il concetto di "catena di timestap", sistema in cui i timestamp dei documenti vengono collegati per creare una sequenza cronologica. In questo modo, se un documento all'interno della catena venisse modificato, il collegamento tra i timestamp verrebbe interrotto, creando una discontinuitò nella catena, cosa che quindi incrementa maggiormente la sicurezza (45).

Ovviamente un protocollo di marcatura temporale deve possedere alcune caratteristiche fondamentali, tra cui:

- correttezza dal punto di vista temporale, quindi che sia impossibile marcare un documento con data e ora diverse da quelle effettive;
- immutabilità del timestamp, senza che la modifica sia individuabile;
- immutabilità di un documento marcato con timestamp, senza che la marcatura rimanga invariata.

Una soluzione iniziale semplice sarebbe che il marcatore, dopo aver apposto la firma sul documento, mantenesse una copia dello stesso. In caso di contestazioni sulla sua autenticità, basterebbe confrontare il documento originale con la copia conservata dal marcatore per verificarne l'integrità, senza compromettere la marcatura.
Tuttavia, questa soluzione comporta alcune problematiche, come la questione della privacy del documento e la gestione delle dimensioni dei documenti archiviati.

La loro proposta, per ovviare a questi problemi, è l'utilizzo di una funzione hash, definita nella Sezione 2.6, da parte del mittente, sul documento da firmare; in questo modo il marcatore non andrebbe più a conservare documenti di grandi dimensioni, in quanto la funzione hash ha un output di dimensione definita, qualunque sia l'input e si andrebbe anche ad eliminare il problema della privacy (45).

3.4 Merkle Tree 1992

Nel 1992, Bayer, Stuart Haber e W. Scott Stornetta pubblicarono un articolo, *Improving the Efficiency and Reliability of Digital Time-Stamping*, dove andavano a migliorare il sistema di timestamp definito precedentemente, considerando anche i Merkle Tree nello studio (inventati da Ralph Merkle nel 1979), così da aumentare l'efficienza, potendo racchiudere più certificati di documenti in un singolo blocco (30).

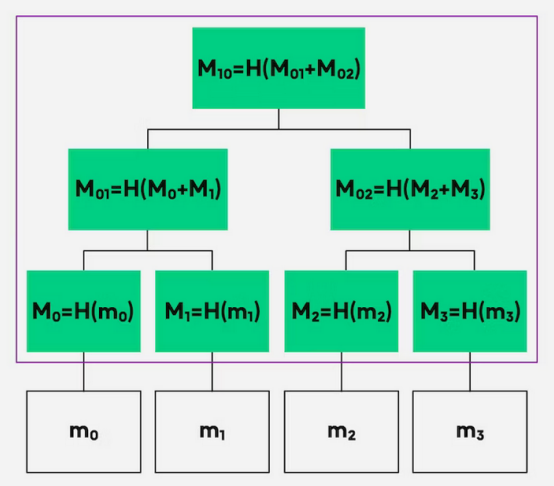

Figura 3.1: Esempio Merkle Tree con quattro messaggi (32).

In Figura 3.1 si può notare la struttura di un Merkle Tree, anche detto Albero di Hash, che parte da quattro messaggi. In questo modo la verifica finale, per grandi strutture di dati, è più sicura e più efficiente, in quanto basta controllare esclusivamente la hash alla radice dell'albero.

Si può vedere che, partendo dai messaggi in chiaro m0, m1, m2 e m3, si procede con un primo passo tramite una funzione hash, e successivamente segue un altro passo, sempre tramite funzione hash, ma delle hash concatenate del passaggio precedente. Infine, si conclude applicando nuovamente la funzione hash alle hash concantenate al passo precedente.
Cambiando anche solo un singolo valore all'interno di uno dei messaggi in chiaro, si otterranno funzione hash diverse durante tutti i passaggi successivi.

3.5 A Cypherpunk's Manifesto 1993

I Cypherpunk sono un gruppo "crypto-anarchico" che sostiene l'uso intensivo della crittografia come tassello fondamentale per un cambiamento sociale e politico.
Nel 1993, Eric Hughes, attivista e forte sostenitore della crittografia e della privacy digitale, pubblicò un manifesto, *A Cypherpunk's Manifesto*, dove vengono enunciate le principali ideologie del movimento cypherpunk (46).

"La privacy è una necessità per una società aperta nell'era dell'elettronica. Siccome ogni informazione può essere rivelata, dobbiamo assicurarci di rivelare il meno possibile."

"Noi Cypherpunks abbiamo come missione la costruzione di sistemi anonimi. Difendiamo la nostra privacy usando crittografia, sistemi di inoltro, firme digitali e denaro elettronico."

"Affinchè la privacy sia diffusa deve essere parte di un contratto sociale. La gente deve unirsi e mettere in piedi tali sistemi per il bene comune"

"Procediamo assieme, velocemente. Avanti."

3.6 Hashcash 1997

Hashcash è un sistema *Proof of Work* (PoW), proposto da Adam Back nel 1997 per limitare le email spam e attacchi di denial of service (29).

Un attacco Denial of Service (DoS) è un'azione con lo scopo di rallentare o bloccare le risorse di un sistema informatico, che eroga un determinato servizio.
In questo modo, la banda di comunicazione viene saturata completamente e i client non riescono a raggiungere le risorse erogate dal server sotto attacco (60).

Il termine Proof of Work, invece, si riferisce ad una prova di lavoro computazionale, quindi un compito che un partecipante della rete deve eseguire per ottenere l'accesso ad un determinato servizio.

Tornando a Hashcash, l'utente mittente, per inviare una email, deve prima creare un timbro, che successivamente allegherà alla email durante l'invio. Il timbro viene creato unendo al messaggio una stringa casuale e calcolando la hash SHA-1 a 160 bit della concatenazione. Vengono generate e concatenate nuove stringhe casuali finchè l'hash calcolata non inizierà con 20 zeri.
Questo processo può essere un po' dispendioso per chi deve inviare la email, motivo

per cui gli spammer vengono limitati, dovendo creare innumerevoli timbri per tutte le email spam che vogliono inviare, ma per chi riceve la email è immediato verificare se il timbro sia corretto o no (37).

Questo concetto sarà adottato successivamente da Bitcoin e molte altre criptovalute, per confermare le transazioni e aggiungere nuovi blocchi alla catena.

3.7 The Idea of Smart Contracts 1997

Nel 1997, Nick Szabo, un informatico di DigiCash, iniziò a definire una prima idea di Smart Contracts, ancora slegata dal contesto Blockchain, pubblicando l'articolo *The Idea of Smart Contracts* (65).
Uno smart contract è un protocollo informatico che agevola e garantisce l'esecuzione automatica di un accordo contrattuale.
Con l'avvento della Blockchain, un contratto è un programma eseguito sui nodi validatori, il cui risultato, solitamente corrispondente a una modifica dello stato della stessa blockchain, costituisce una transazione soggetta al consenso dei nodi validatori.
Questi smart contract mirano a garantire una maggiore sicurezza e velocità nell'esecuzione rispetto ai contratti tradizionali, riducendo drasticamente i costi di transazione (37).

3.8 B-Money 1998

Nel 1998, Wei Dai definì B-Money, una delle prime proposte di criptovaluta (36).
L'obiettivo principale era quello di costruire un sistema di contante elettronico anonimo e distribuito.
Similmente a come farà Bitcoin, l'utilizzo di Hashcash come Proof of Work era visto come possibile mezzo per generare denaro e le transizioni erano inviate a tutti i partecipanti della rete, che condividevano il compito di tenere la totalità dei conti.
Molte delle idee di Dai sono state riprese da Nakamoto per la creazione di Bitcoin e in seguito da altre criptovalute (37).

3.9 Reusable Proof of Work 2004

Nel 2004, Hal Finney ha ideato un protocollo che, similmente a B-Money, crea valuta utilizzando la Proof of Work basata su Hashcash, ma che dà anche la possibilità di trasferire in modo sicuro la criptovaluta da un utente all'altro (41).

In pratica, una volta generato un token Proof of Work (PoW), sotto forma di un pezzo di hashcash, esso può essere utilizzato per ottenere un token, denominato da Finney, Reusable Proof of Work (RPoW), di pari valore.
Questo token RPoW può quindi essere impiegato in qualsiasi modo similmente ad un token hashcash, però non è scartato dopo l'utilizzo e può essere scambiato dal destinatario per un nuovo token RPoW di pari valore (41).
Ogni token RPoW o PoW può essere usato solo una volta, ma poichè dà vita a uno nuovo è come se possa essere riutilizzato (da cui deriva il nome ReusablePoW): quindi un singolo token PoW costituisce la base per una catena di token RPoW (37).

3.10 Bit Gold 2005

Nel 2005 Nick Szabo pubblicò una proposta su una moneta digitale anonima e distribuita, *Bit gold*.
Come per i token RPoW di Finney, anche Bit gold genera moneta basandosi sulla Proof of Work di Hashcash. La proposta di Nick si fonda sul calcolo di una stringa di bits partendo da una *challenge string* pubblica, a cui ogni utente lega una stringa personale e il timestamp, per poi calcolare la hash finchè non risulta minore di un determinato target.
Il primo utente che riuscirà a ottenere una *proof of work string (timestamped)* con hash minore del target, la pubblicherà, insieme alla *challenge string* in un registro pubblico e distribuito, come prova di verifica.
A sua volta, la *proof of work string* appena calcolata verrà usata come *challenge string* al passo successivo; in questo modo si andrà a generare una catena di Proof of Work pubblica (66).

3.11 Bitcoin 2008

Nel 2008, una delle più grandi banche di investimento mondiali, la Lehman Brothers Holdings Inc., ha dichiarato fallimento con centinaia di miliardi di debiti.
Questo fatto ha aumentato i dubbi di molti sul sistema economico occidentale e, non a caso, l'idea di Bitcoin nacque proprio alla fine del 2008.

Un anonimo inventore, noto con lo pseudonimo di Satoshi Nakamoto, pubblicò verso la fine del 2008 l'articolo dal titolo *Bitcoin A Peer-to-Peer Electronic Cash System* (56), nel quale si definiva la creazione di una criptomoneta digitale, anonima e distribuita, gestita da una rete peer-to-peer di utenti.
Il 18 agosto 2008 è stato registrato il dominio "bitcoin.org" e a seguito del whitepaper di novembre 2008, il 3 gennaio del 2009 è stato creato il genesis block (il primo blocco della blockchain) di Bitcoin.
Da quel momento in poi, per molti anni, la valutazione di Bitcoin è salita molto lentamente, fino al 2017, quando il notevole incremento dell'utilizzo ha fatto crescere il suo valore (37).

Figura 3.2: Andamento prezzo Bitcoin negli anni (8).

Come si può vedere dalla Figura 3.2, il valore di un singolo bitcoin, dal 2009 ad oggi è passato da 0.00076 $ a valori superiori a 60 mila $, mentre al momento si aggira attorno a 29 mila $, anche se il prezzo è comunque molto volatile, in base all'interesse e all'utilizzo del momento.

3.12 Ethereum 2015

Ethereum è stato fondato nel 2015 da un gruppo di sviluppatori guidato da Vitalik Buterin. Non si tratta solo di una nuova criptomoneta, come Bitcoin, ma di un cambio radicale nel mondo della Blockchain; infatti con la sua creazione si è cominciato a parlare di Blockchain 2.0, rispetto alla Blockchain 1.0, che era usata esclusivamente con lo scopo di gestire transazioni sicure di monete digitali.

Ethereum è una piattaforma globale, open source basata su una blockchain, dove possono essere eseguite delle applicazioni decentralizzate (dApps), dette Smart Contract.
Come Bitcoin, ha una sua criptomoneta (ether - ETH), ma Ethereum è soprattutto lo strumento per gestire quello che è sostanzialmente un enorme computer globale, che non si può spegnere e le cui applicazioni una volta lanciate non possono essere bloccate (37).

Figura 3.3: Andamento prezzo Ethereum negli anni (8).

Come si può vedere dalla Figura 3.3, l'andamento del prezzo di Ethereum ha fatto movimenti simili a quello di Bitcoin, raggiungendo anche valori superiori a 4500 $ per un singolo ether, quando Bitcoin ha raggiunto i suoi massimi storici.

Capitolo 4

Introduzione al concetto di Blockchain

4.1 Definizione di Blockchain

La Blockchain è una sottofamiglia di tecnologie, che può essere considerata come un database che memorizza tutti i dati all'interno di blocchi legati tra di loro, formando una catena. Quando una nuova transazione deve essere aggiunta in un blocco, il mittente la invia all'interno della rete peer-to-peer verso tutti nodi della rete, che devono procedere alla verifica e alla validazione, affidandosi ad un meccanisimo di consenso comune a tutta la rete (28).

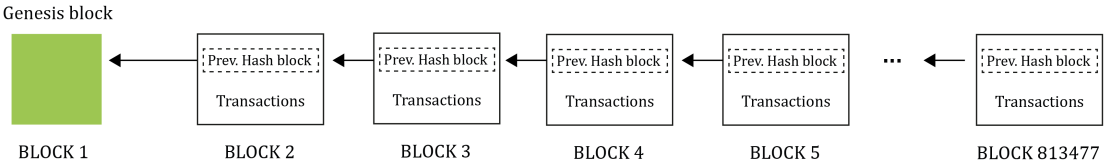

Figura 4.1: Schema generale di una blockchain.

La tecnologia Blockchain si basa su tre pilastri fondamentali:

- immutabilità,
- decentralizzazione,

- trasparenza.

Un blocco, una volta creato e aggiunto ad una blockchain, è immutabile, perchè è strettamente legato al blocco precedente, in quanto contiene la hash del suo header (concetti che verranno approfonditi in seguito). In questo modo, se si provasse a modificare una transazione all'interno di un blocco al passo temporale i, a cui sono già stati legati altri blocchi successivamente, si modificherebbe l'hash del suo header presente nel blocco al passo $i + 1$ e quindi tutte le hash di tutti i blocchi successivi del passo i. In questo modo, andando a modificare un blocco, si dovrebbe necessariamente modificare tutta la blockchain da quel punto in avanti.
Questo meccanismo garantisce l'integrità dei dati e quindi si può sempre far riferimento alle informazioni memorizzate perché si sa che non sono state modificate nel tempo. Un determinato ammontare di criptovaluta, per esempio, può essere scambiato da un indirizzo ad un altro, ma questa informazione non viene modificata in un blocco passato, ma inserita in un nuovo blocco che fa allungare la catena. Questo garantisce la possibilità di tracciare l'intera cronologia in modo affidabile (47).

Il concetto di decentralizzazione è molto importante nella tecnologia Blockchain, in quanto non è presente un'autorità centrale che coordina tutto, ma ogni partecipante è su un piano di parità con gli altri. Se un utente deve inviare denaro ad un altro, per esempio, non ha bisogno di una terza parte, ma basta che lo invii seguendo il protocollo della blockchain presente. Essendo decentralizzato, i dati non sono archiviati in un unico punto, ma distribuiti a tutti i nodi della rete, e quindi non possono essere hackerati.
Non essendoci un'entità centrale, inoltre, la rete può comunque continuare a funzionare anche se un nodo viene spento (47).

Il fatto che la blockchain di Bitcoin, per esempio, sia completamente trasparente a chiunque non vuol dire che un trasferimento sia facilmente riconducibile ad un individuo o ad un'azienda. Ogni individuo, per usare una blockchain, deve avere un proprio indirizzo, calcolato tramite protocolli crittografici che lo slegano totalmente dalla sua vera identità. Alcune aziende, però, possono rendere pubblici i loro indirizzi per aumentarne la loro credibilità (47).

4.2 Rete Peer-To-Peer P2P

In informatica, a livello applicativo, ci sono due principali architetture di rete: la *Cliet-Server* e la *Peer-To-Peer (P2P)*.

L'architettura client-server è un modello di rete che si basa sul concetto di server, dispositivi o software specializzati che forniscono servizi o dati ai client, che ne fanno richiesta.
Un'architettura peer-to-peer, invece, è una rete di utenti che comunicano direttamente tra di loro senza la necessità di un server centralizzato; infatti ogni nodo può agire sia da server sia da client (64).
La decentralizzazione, quindi, svolge un ruolo molto importante in questa tipologia di architettura, come anche la scalabilità, in quanto la rete può crescere facilmente aggiungendo nuovi nodi e si raggiunge un livello di efficienza superiore, riducendo il carico su singoli server centrali.

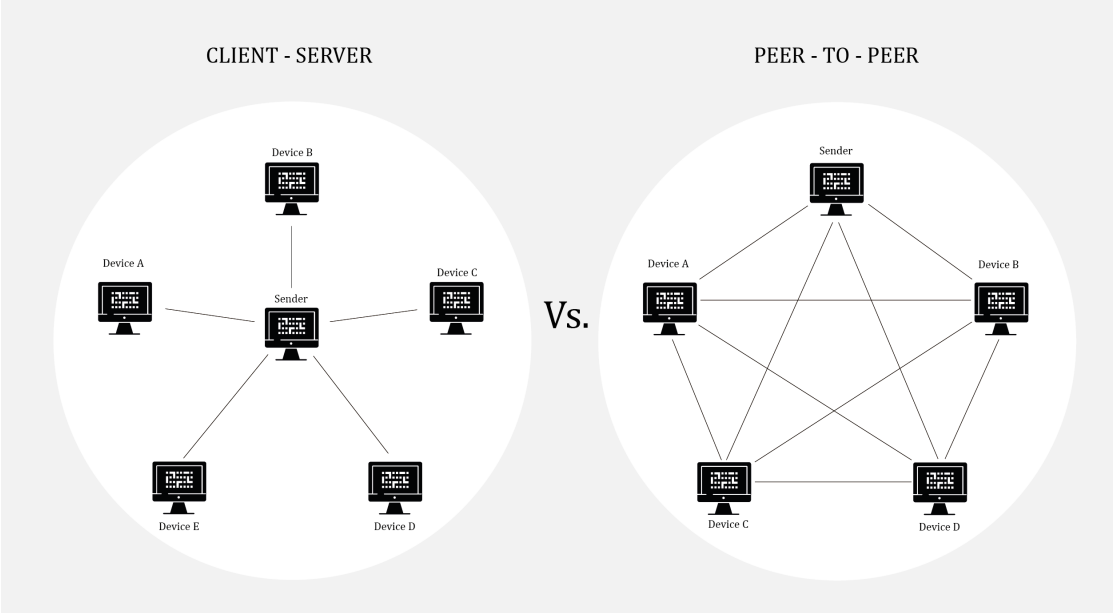

Figura 4.2: Confronto tra architettura client-server e peer-to-peer.

Questa tecnologia si basa sul concetto di decentralizzazione, e all'interno della Blockchain ciò è importantissimo, in quanto si elimina la necessità di un'autorità

centrale e in tutti i nodi è salvata una copia dell'intera catena e quindi possono partecipare al processo di validazione della transazioni e di creazione dei blocchi. Inoltre, tutti i nodi possono verificare se un validatore ha creato un blocco valido, semplicemente controllando la hash che ha trovato durante il processo di mining, come per esempio con Bitcoin.

4.3 Tipi di Blockchain

In base allo scopo di utilizzo di una infrastruttura Blockchain, alcune tipologie sono migliori di altre.

4.3.1 Distributed Ledger

Il termine *Distributed Ledger* si riferisce in sostanza a tutti i database distribuiti, che vengono condivisi e sincronizzati tra i nodi decentralizzati di una rete. Di conseguenza, tutte le blockchain sono tecnicamente dei distributed ledger; tuttavia, non si può affermare che tutti i distributed ledger siano delle blockchain, poiché, per esempio, i dati non sono sempre conservati in una catena di blocchi, ma possono essere salvati in differente maniera (49).

L'infrastruttura che consente l'implementazione e il funzionamento dei distributed ledger è detta *Distributed Ledger Technology (DLT)*, e rappresenta un modello emergente all'interno del campo della Blockchain. La DLT si basa su un sistema decentralizzato di registrazione e condivisione delle informazioni, in cui il registro è distribuito tra un insieme di nodi partecipanti che raggiungono un consenso sulla validità delle transazioni attraverso un determinato meccanismo (67).

La DLT, quindi, offre un'architettura resistente e sicura, che consente a ogni partecipante di possedere una copia identica del registro distribuito. Le blockchain sono un particolare tipo di DLT dove le transazioni vengono crittografate e inserite in blocchi sequenziali di dati, che vengono poi collegati tra loro in modo immutabile attraverso algoritmi di hash crittografici e secondo una sequenza temporale ben definita. Questo approccio garantisce l'integrità e l'immutabilità dei dati registrati,

rendendo estremamente difficile la manipolazione o la cancellazione retroattiva delle informazioni.

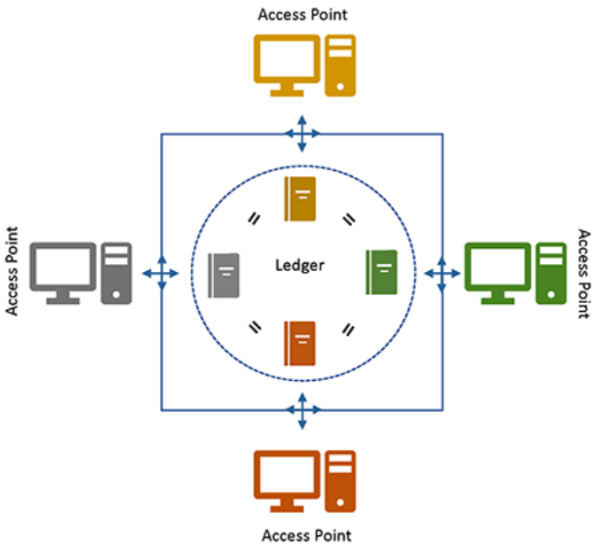

Figura 4.3: Rappresentazione Distributed Ledger Technology (51).

4.3.2 Public Blockchain

Le blockchain pubbliche sono reti pubbliche, proprio come la rete di Bitcoin, aperte a tutti gli utenti che desiderino effettuare transazioni o diventare nodi della rete. Inoltre, chiunque può vedere tutto quello che succede all'interno della blockchain. Questa tipologia è priva di autorizzazioni, quindi tutte le transazioni che vengono inviate sulla rete, se rispettano le regole di sintassi della blockchain, vengono aggiunte alla lista di transazioni da validare e successivamente inserite nei blocchi tramite un meccanismo di consenso crittografico.
La decentralizzazione è un punto molto importante per le blockchain pubbliche; infatti esse non richiedono un centro di autorità centrale che può confermare o negare transazioni specifiche e quindi tutte le transazioni vengono registrate su un unico database che è archiviato da tutti i nodi della rete.
Le transazioni presenti in un blocco validato e inserito all'interno della catena non possono essere annullate.

4.3.3 Private Blockchain

In alcuni ambiti, i sistemi decentralizzati presentano delle carenze e la completa trasparenza non è sempre auspicabile. In questi casi si possono sviluppare delle blockchain private, che sono aperte solo ai partecipanti che soddisfano i criteri di appartenenza alla rete, diversamente dalle pubbliche, dove chiunque può interagire.
In questa tipologia di blockchain i consensi vengono decisi esclusivamente da determinati membri con autorizzazione (autorità di consenso) e il costo computazionale per inserire un blocco all'interno della catena è molto più efficiente, in quanto c'è fiducia verso le autorità di consenso.
Queste blockchain sono adatte per casi d'uso in cui è necessario un grado di controllo e privacy molto alto, per esempio, all'interno di un'azienda; infatti, in alcuni casi, il livello di decentralizzazione è molto basso.

4.3.4 Hybrid Blockchain

Le blockchain ibride sono blockchain che derivano le loro caratteristiche da entrambi i tipi precedenti, cercando di ottenere tutti i vantaggi possibili in base allo scopo finale.
Possono essere formulati sistemi diversi; per esempio, la verifica dei dati delle transazioni può avvenire come per le blockchain pubbliche, ma poi i dati vengono archiviati in modo privato, accessibile solo a chi ne è autorizzato.
In ogni caso, i dati delle transazioni salvati all'interno della catena non possono essere modificati, per definizione del sistema, ma possono essere mantenuti privati totalmente o parzialmente.

Queste blockchain ibride, inoltre, sono più veloci e scalabili rispetto alle blockchain pubbliche; e, a differenza delle private, sono più facili da controllare.
La loro flessibilità, in aggiunta, permette una personalizzazione interna, in base alle esigenze del sistema (25).

4.3.5 Permissionless e Permissioned

Una *blockchain premissionless* non ha dei permessi specifici, quindi tutti possono diventare nodi della rete e partecipare al processo di validazione e inserimento dei blocchi. Bitcoin e Ethereum sono due esempi di blockchain permissionless, e solitamente questa tipologia è strettamente legata con le blockchain pubbliche.
Nelle *blockchain premissioned*, invece, l'ingresso ai nodi entranti è limitato e quindi gli stessi nodi possono partecipare alla rete solo in seguito alla concessione di un permesso da parte degli amministratori, cosa molto usata all'interno delle blockchain private.

Possono esistere diverse configurazioni delle tipologie di blockchain appena enunciate: per esempio, una blockchain pubblica permissioned permette un accesso libero alle transazioni e ai dati, ma la creazione dei blocchi è consentita ad un numero ristretto di nodi. In base alle esigenze per cui una blockchain è stata creata, avrà delle caratteristiche differenti.

4.4 Tipologie di Consenso

I meccanismi di consenso si riferiscono all'insieme completo di protocolli che consentono ad una rete di nodi di constatare la correttezza di una blockchain (43).

Il consenso è fondamentale per garantire l'integrità e la sicurezza di una blockchain, evitando l'inserimento di transazioni fraudolente o errate, che, una volta inserite in un blocco, non potrebbero più essere annullate.
Come già anticipato, ogni blocco nella catena contiene l'hash del blocco precedente, quindi, se si modificasse un determinato blocco passato, sarebbero da modificare tutti gli hash dei blocchi successivi. Questo fatto, essendo l'intera catena distribuita, rende estremamente difficile modificare fraudolentemente un blocco già inserito: è il motivo per cui il processo di consenso di inserimento di un blocco è il momento più delicato e oggetto di possibili attacchi.

Esistono diversi protocolli di consenso, ma quelli più usati dalle blockchain principali sono la Proof of Work (PoW) e la Proof of Stake (PoS).

4.4.1 Proof of Work

La Proof of Work è un sistema per scoraggiare attacchi denial of service e altri abusi di servizio, come spam sulla rete, e prevede l'imposizione di alcuni lavori al richiedente del servizio.
Una caratteristica fondamentale di questi schemi è la loro asimmetria: il lavoro deve essere moderatamente complesso dal lato richiedente (fattibile per piccoli numeri, ma ingestibile in grandi quantità) e veloce da verificare per il fornitore del servizio (37).
Il sistema HashCash è un metodo di consenso Proof of Work usato da Bitcoin.

4.4.2 Proof of Stake

Il protocollo di consenso Proof of Stake, invece, segue il principio di far inserire i blocchi tenendo conto dell'ammontare di criptomonete possedute dal validator (validatore di blocchi) e messe in staking, con l'idea che chi possiede grosse quantità di valuta ha tutto l'interesse a far funzionare il sistema (37).
Ovviamente non si possono far inserire i blocchi sempre dall'utente con maggior criptovaluta, altrimenti tutti i blocchi sarebbero inseriti da lui; quindi ci sono alcune modalità per decidere chi andrà ad inserire i blocchi successivi, per esempio, randomicità pesata o anzianità.

Nel primo caso, in base alla quantità di moneta che si possiede, si ha una determinata probabilità di inserire un blocco: più moneta si possiede, più la probabilità è alta.
In caso di modalità basata su anzianità, tutti i nodi hanno un determinato valore di anzianità, che dipende da quante e da quando si posseggono le monete; ciò determina la probabilità di inserire un blocco. Una volta che un nodo inserisce un blocco, il suo valore di anzianità decade a zero e riprende a crescere se continua a tenere in staking le sue monete.

4.4.3 Altri metodi di consenso

Altri metodi di consenso meno comuni possono essere:

- *DPoS, Delegated Proof of Stake*: non è l'intera rete a occuparsi dell'inserimento dei nuovi blocchi, ma questo compito viene delegato a un gruppo ristretto di utenti.
- *PoI, Proof of Importance*: versione modificata della PoS, ma per inserire i blocchi non conta solo la quantità di moneta posseduta, bensì anche la quantità di moneta scambiata.
- *PoB, Proof of Burn*: maggiore è il numero di criptomoneta bruciata da un utente in favore del sistema, maggiore è la sua mining power e, di conseguenza, più alte sono le probabilità di essere scelto come validatore del blocco successivo.
- *Proof of Space* detta anche *Proof of Capacity* o *Proof of Storage*: è un protocollo di consenso che si basa sulla quantità di memoria sul disco che i miners allocano. Maggiore quantità di memoria dà maggiore probabilità di inserire il blocco successivo.
- *PoA, Proof of Authority*: protocollo di consenso usato principalmente nelle blockchain private, in quanto i validatori mettono la loro vera identità e reputazione per garantire la trasparenza dei blocchi aggiunti

Capitolo 5

Architettura e Funzionamento della Blockchain

Per studiare l'architettura e il funzionamento interno di una blockchain si può prendere in considerazione come esempio lo studio della blockchain di Bitcoin.

5.1 Codifiche Bitcoin

All'interno della blockchain di Bitcoin, basandosi su un sistema a chiave pubblica, ogni utente possiede una coppia di chiavi, una privata e una pubblica. Attraverso la chiave pubblica viene poi generato il relativo indirizzo, che si può comunicare per farsi inviare dei pagamenti.
Prima di procedere con la definizione del protocollo per la generazione delle chiavi, sono da definire alcuni cenni di crittografia, su cui si basa questa blockchain.

5.1.1 Curva secp256k1 di Bitcoin

Bitcoin utilizza un sistema a chiave pubblica basato sulle curve ellittiche con l'algoritmo ECDSA, precisamente la curva usata è la secp256k1, che ha parametri $a = 0$, $b = 7$, e come numero primo p per il campo finito F_p:

$$\text{p} = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1.$$

Per generare una coppia di chiavi (pubblica e privata), partendo da una curva ellittica, serve inoltre un punto della curva, detto punto generatore G, attraverso il quale si andrà a calcolare la chiave pubblica, usando la chiave privata (14).

Nella curva di Bitcoin, il generatore, nella sua forma compressa, quindi con solo la componente x, è

G = 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798,

mentre, in forma non compressa,

G = 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08 FFB10D4B8.

Data la chiave privata d (una sequenza di 256 bit) e il generatore G, la chiave pubblica P viene quindi calcolata come

$$P = dG \mod (p). \tag{5.1}$$

Il calcolo della chiave pubblica è unidirezionale, in quanto, trovare P, conoscendo d e G è immediato, ma partendo da P e G, è un calcolo computazionalmente infattibile ottenere la chiave privata d.
La chiave pubblica non è un intero, ma un punto della curva, quindi coordinate che verificano l'equazione della curva secp256k1 (37).

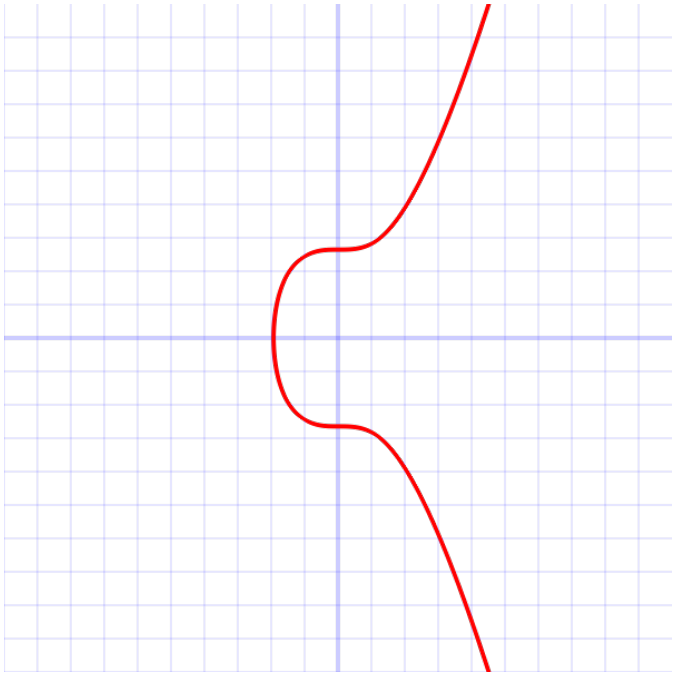

Figura 5.1: Curva di Bitcoin Secp256k1 (14).

In Figura 5.1 si può vedere una rappresentazione della secp256k1 sui numeri reali, in realtà, però, essendo la curva definità su un campo finito F_p, la sua rappresentazione è simile a Figura 2.3, quindi un insieme di punti sparsi (14).
Per la secp256k1 il numero di punti della curva è

1157920892373161954235709850086879078528375642790749043826051631415181614943337.

5.1.2 Generazione chiavi private

Una chiave privata su Bitcoin è una sequenza di 256 bit.
Questa formulazione, però, non è facile da gestire per gli utenti, in quanto è una sequenza casuale di 256 zeri e uni, quindi si può passare alla sua formulazione in decimale o esadecimale.
La codifica più utilizzata, però, viene chiamata Wallet Import Format (WIF) e rappresenta la chiave privata dopo una codifica tramite algoritmo Base58, codifica alfanumerica che evita alcuni classici errori come "O" "0" O "l" e "I".

Gli step necessari per ottenere una chiave codificata tramite WIF sono i seguenti (37).

- Si converte la chiave generata di 256 bit in formato esadecimale, ottenendo una chiave di 64 cifre.
- Si concatena alla chiave privata esadecimale il prefisso esadecimale 0x80, ottenendo la extended private key.
- Si calcola due volte l'hash di questa quantità tramite SHA-256.
  Le prime 8 cifre esadecimali rappresentano il checksum, che viene concatenato come suffisso alla chiave privata.
- La quantità ottenuta viene codificata tramite Base58.

Il numero di chiavi private che si possono generare è, quindi, 2^{256}, circa il numero di atomi presenti nell'universo conosciuto. Se un utente dovesse generare la stessa identica sequenza di 256 bit di un altro, avrebbe la possibilità di spostare i bitcoin legati a quella chiave. La probabilità che questo accada, però, è veramente molto vicina allo zero.

5.1.3 Generazione chiavi pubbliche

Come anticipato nella Sezione 5.1.1, per ottenere la chiave pubblica $P = (Px, Py)$, partendo dalla chiave privata, basta moltiplicarla per il punto generatore della curva di Bitcoin secp256k1.

Come per le chiavi private, anche le chiavi pubbliche possono essere codificate in molti modi differenti (37).

- All'inizio, si utilizzavano chiavi pubbliche non compresse, cioè venivano salvate entrambe le coordinate del punto della curva. Sulla blockchain, veniva salvata come 04||Px||Py (04 è un prefisso esadecimale).
- Dopo un po' di tempo, si è cominciato a memorizzare soltanto la coordinata Px del punto, insieme ad un byte extra per indicare la parità o disparità di y.

In particolare, un punto viene salvato come 02||Px se la coordinata y è pari, mentre viene salvato come 03||Px se la coordinata y è dispari.

5.1.4 Generazione indirizzi

Anche per quanto riguarda gli indirizzi, ci sono più formati che possono essere definiti.

Per ottenere un address che inizia con la cifra 1 (P2PKH address) o con la cifra 3 (P2SH address) su Bitcoin si procede nel seguente modo:

- Si hasha la chiave pubblica (per un P2PKH address) o lo script (per un P2SH address) in successione, prima con SHA-256 e poi con RIPEMD-160.
- Si concatena il prefisso 0x00 (per un P2PKH address) o 0x05 (per un P2SH address) al digest appena ottenuto.
- Si hasha due volte la sequenza ottenuta con la funzione SHA-256. Le prime 8 cifre esadecimali, rappresentano il checksum, che viene concatenato come suffisso alla sequenza ottenuta dopo il secondo step.
- Quest'ultima sequenza viene codificata in Base58.

Per il formato SegWit address, che inizia con la sequenza bc1, si procede come:

- Si hasha la chiave pubblica in successione, prima con SHA-256 e poi con RIPEMD-160. Il digest che si ottiene è il witness program.
- Il witness program viene poi codificato in bech32.

5.1.5 Esempio processo di generazione di un P2PKH indirizzo Bitcoin

Il processo di generazione di un P2PKH indirizzo Bitcoin, quindi, prevede vari step da seguire, partendo dalla generazione della chiave privata (15).

Si genera una chiave privata da 256 bit e si converte in formato esadecimale

18e14a7b6a307f426a94f8114701e7c8e774e7f9a47e2c2035db29a206321725.

Successivamente si calcola la relativa chiave pubblica in forma compressa, quindi solo coordinata x di 32 bytes e 1 bytes iniziale per la parità della coordinata y

0250863ad64a87ae8a2fe83c1af1a8403cb53f53e486d8511dad8a04887e5b2352.

Si procede ai passaggi di hashing, prima tramite SHA-256, poi con RIPEMD-160 e si concatena il prefisso 0x00, ottenendo il *extended RIPEMD-160*

00f54a5851e9372b87810a8e60cdd2e7cfd80b6e31.

Si eseguono nuovamente due passaggi di SHA-256

c7f18fe8fcbed6396741e58ad259b5cb16b7fd7f041904147ba1dcffabf747fd,

e si prendono i primi 4 byte, quindi le prime 8 cifre, come checksum: c7f18fe8.
A questo punto si concatena il checksum ottenuto come suffisso al *extended RIPEMD-160*

00f54a5851e9372b87810a8e60cdd2e7cfd80b6e31c7f18fe8,

e si ottiene l'indirizzo codificando il tutto tramite Base58

1PMycacnJaSqwwJqjawXBErnLsZ7RkXUAs.

5.2 Block

Come già anticipato, Bitcoin registra tutte le transazioni tramite una blockchain pubblica, condivisa e gestita direttamente dai nodi di una rete peer-to-peer.
Ogni blocco è suddiviso in due parti principali: *header* e *body*. Il body racchiude tutte le transazioni del blocco, mentre nell'header sono presenti campi di gestione del blocco stesso.

5.2.1 Header Block

La header di un blocco contiene tutte le informazioni di gestione relative al blocco stesso (37).

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The proof-of-work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the proof-of-work algorithm

Figura 5.2: Header blocco di Bitcoin (62).

Il campo *Version* si riferisce esclusivamente alla versione del software/protocollo usato al momento.
La *Previous Block Hash* è la hash SHA-256 dell'header del blocco precedente, mentre la *Merkle Root* è l'hash SHA-256 di tutti gli hash di tutte le transazioni presenti del body del blocco.
Il *Timestamp* è il valore approssimato dell'istante di creazione del blocco.
La *Difficulty Target* è il valore corrente del target. Con valori di bits più alti, sarà più facile trovare una hash per convalidare un blocco, dato che è più facile trovare un valore inferiore al target.
Infine *Nonce* è un valore che viene inserito in modo casuale fino a che la funzione di hash (eseguendo due volte di seguito SHA-256) del blocco risulterà inferiore al target.

5.2.2 Nodes

I nodi sono i pilastri della rete di Bitcoin, come per qualsiasi altra blockchain. Svolgono il ruolo di monitorare continuamente il buon andamento della blockchain e validano le nuove transazioni, che devono essere inserite in un nuovo blocco da aggiungere alla catena principale.
Quando un utente invia alla rete una nuova transazione, essa entra a far parte di un pool di transazioni da validare, da cui i nodi attingono per creare nuovi blocchi provvisori su cui far partire il processo di prova di lavoro (*Proof of Work (PoW)*)

per validare il blocco e inserirlo nella catena.

All'interno della rete di nodi della blockchain di Bitcoin ci sono due tipologie di nodi, i *Full Nodes* e i *Light Nodes* (37).
La differenza è nella memoria di archiviazione dei dati dell'intera blockchain, e pertanto anche dal lato della verifica. Ogni full node memorizza l'intera blockchain (quindi tutte le transazioni degli utenti dall'inizio della storia di Bitcoin), e gestisce gli aspetti del protocollo, compresa la verifica delle le transazioni che vengono proposte alla rete.

Un light node, invece, non memorizza l'intera blockchain, ma solo gli header di tutti i blocchi (80 byte invece di oltre 1 MB, risparmiando oltre 99.99% di spazio) e ha bisogno di una minima potenza di calcolo.
Nonostante queste limitazioni, può comunque avere un ruolo nella rete: può inviare le transazioni alla rete e verificarne la correttezza formale. Non è invece in grado di verificare se chi vuole eseguire il pagamento abbia effettivamente ricevuto a sua volta, su quell'indirizzo, abbastanza bitcoin, senza averli già spesi in passato; per far questo serve l'intera blockchain.

Il light node non è in grado quindi di verificare autonomamente se una transazione è corretta, ma è in grado di verificare che è in un blocco, dando per scontato che, se è in un blocco, significa che è stata accettata dalla rete e quindi è corretta. Tutto questo può essere fatto trasferendo pochissimi byte tra il full node e il light node, perchè data la struttura del Merkle Tree, le hash intermedie sono dell'ordine del logaritmo del numero delle transazioni di un blocco.

5.3 Transazioni

Le transazioni sono una componente fondamentale nella blockchain di Bitcoin: esse sono operazioni che permettono il trasferimento di valore tra gli utenti della rete. Una volta che una transazione viene creata, seguendo una precisa sintassi, viene poi propagata sulla rete, validata dai nodi e infine aggiunta alla catena principale di blocchi di Bitcoin.

5.3.1 Transaction Outputs (UTXO)

Le componenti principali di una transazione sono i *transaction outputs*, denominati come UTXO. Essi rappresentano una quantità indivisibile di bitcoin che si trovano su un indirizzo e che non è stata ancora spesa dal proprietario. L'insieme di tutti i UTXO presenti in un dato momento è definito come *set UTXO* e, tutte le volte che viene fatta una transazione, segue un cambiamento di stato nel set UTXO, in quanto gli input della transazione vengono eliminati, mentre gli output diventano dei nuovi UTXO. Ovviamente il saldo di un portafoglio è la somma di tutti gli UTXO che possono essere spesi da quel determinato wallet (37).

Il valore contenuto in un UTXO non è fisso, ma un valore in multipli di satoshi, e non è divisibile, quindi va speso interamente durante una transazione.
Per esempio, l'utente A ha un portafoglio con saldo 0.5 BTC, che comprendono due UTXO ottenuti da transazioni passate, uno da 0.1 BTC e uno da 0.4 BTC, e vuole inviare 0.3 BTC all'utente B.
L'unica possibilità è creare una transazione che ha come input 0.4 BTC UTXO, che verranno spezzati e 0.3 andranno all'utente B, mentre 0.1 torneranno al suo indirizzo (un po' meno, essendoci le fee da pagare ai nodi della rete).
Durante questa transazione, l' UTXO di input viene eliminato e si generano i due nuovi UTXO riferiti agli output della transazione, in questo modo il modello UTXO funge da meccanismo del protocollo per tenere traccia di dove si trovano le monete in un dato momento (31).

5.3.2 Script

Un input di una transazione identifica un determinato UTXO, che viene speso grazie ad una prova di possesso, chiamata *unlocking script*, quindi il portafoglio dove sono contenuti i BTC da trasferire crea un input *locking script* sbloccabile tramite il relativo unlocking script, per ogni UTXO utilizzato. Gli script contenuti negli inputs e outputs di ogni transazione sono scritti in un linguaggio di scripting, chiamato *Script* (37).
Questo linguaggio si basa sull'uso delle *pile*, quindi le due principali operazioni sono: *push*, che posiziona un oggetto in cima alla pila e *pop*, che rimuove l'oggetto

che c'è in cima alla pila, mentre gli operatori sono indicati come UP_X, dove X rappresenta l'opreazione da svolgere sugli elementi in cima alla pila.

Supponiamo, per esempio, di possedere un UTXO con relativo locking script

2 OP_ADD 9 OP_EQUAL.

Il contenuto dell'unlocking script, x, per sbloccare questo locking script, viene posizionato per primo nella pila vuota e successivamente vengono inseriti, da sinistra a destra, gli elementi presenti nel locking script.
Ad un certo punto la pila sarà

x 2 OP_ADD,

quindi è da svolgere l'operazione OP_ADD, che richiede di sommare gli elementi presenti nella pila, ottenendo un singolo elemento: x+2.
A questo punto si prosegue aggiungendo gli altri elementi del locking script e infine nella pila si avrà

x+2 9 OP_EQUAL,

che restituisce *True* se x+2=9, quindi il valore da utilizzare come unlocking script è 7.

5.3.3 Pay-to-Public-Key-Hash (P2PKH)

La maggior parte delle transazioni consuma outputs bloccati tramite un *Pay-to-Public-Key-Hash (P2PKH)* script, che per essere sbloccato necessita di inviare la chiave pubblica e la relativa firma digitale creata a partire dalla chiave privata.
Il locking script ha la seguente scrittura:

OP_DUP OP_HASH160 <Address> OP_EQUALVERIFY OP_CHECKSIG.

Il contenuto dell'unlocking script deve essere della forma

Signature PublicKey.

A questo punto nella pila si avrà l'unlocking script e il primo operatore è OP_DUP, che duplica l'elemento in cima alla pila, ottenendo

Signature PublicKey PublicKey,

quindi tramite OP_HASH160, verranno applicate in sequenza le funzioni hash SHA256 e RIPEMD160, che applicate a una chiave pubblica, ritornano l'indirizzo,

Signature PublicKey Address.

Proseguendo con il locking script, aggiungiamo alla pila un secondo Address e con OP_EQUALVERIFY si verifica se i due Address presenti in cima alla pila sono uguali; in caso positivo, vengono tolti dalla pila.

Signature PublicKey Address Address OP_EQUALVERIFY => Signature PublicKey.

Tramite OP_CHECKSIG, infine, si verifica che la firma inserita nella pila sia valida, usando come input la chiave pubblica fornita e in caso positivo restituisce *True*.

In generale, nel linguaggio Script di Bitcoin, è richiesta una firma digitale tramite ECDSA, ogni volta che in uno script è presente un operatore come OP_CHECKSIG, OP_CHECKSIGVERIFY, OP_CHECKMULTISIG e OP_CHECKMULTISIGVERIFY. Gli ultimi due operatori si riferiscono a sistemi multisignature, dove è richiesta più di una chiave pubblica per sbloccare il locking script.

5.3.4 Pay-to-Script-Hash (P2SH)

Un'altra tipologia di script è il *Pay-to-Script-Hash (P2SH)* script, dove è necessario fornire il corretto script, il cui hash è uguale a quello contenuto nel locking script, per sbloccare il contenuto di un UTXO.

OP_HASH160 <script digest> OP_EQUAL.

5.3.5 Pay-to-Witness-Public-Key-Hash (P2WPKH)

Con l'aggiornamento del software di Bitcoin, *Segregated Witness (SegWit)*, implementato nel 2017, è diventato possibile definire un altro formato di address: i SegWit address, indirizzi che iniziano con la sequenza bc1.
Una transazione *Pay-to-Witness-Public-Key-Hash (P2WPKH)*, sotto molti aspetti,

è simile ad una transazione P2PKH e il locking script presenta soltanto due campi, un numero, che sarebbe la *witness version* e una stringa esadecimale, chiamata *witness program*, l'hash della chiave pubblica, quindi lo stesso address presente nel locking script di una normale transazione P2PKH.
Quindi, l'unlocking script per sbloccarne il contenuto è completamente vuoto, perché i dati che servivano per sbloccare un locking script di una P2PKH: Signature e la PublicKey sono ora spostati nel campo witness (61).

Questo aggiornamento ha offerto parecchi benefici alla scalabilità, sicurezza e performance di Bitcoin. Infatti, la firma digitale occupava una grande quantità di spazio all'interno di una transazione e, spostando questo campo altrove, si ha un notevole guadagno, migliorando la scalabilità.

5.3.6 Pay-to-Witness-Script-Hash (P2WSH)

Il *Pay-to-Witness-Script-Hash (P2WSH)*, analogamente al P2WPKH rispetto al P2PKH, è una versione simile al P2SH, che semplifica il locking script, utilizzando meno operatori, cosa che alleggerisce anche il rispettivo unlocking script.

5.4 Creazione di un blocco

Per la creazione di un nuovo blocco all'interno della blockchain, i nodi procedono con la definizione di un blocco provvisorio, riempendo il body con transazioni che devono essere confermate e poi procedono con il calcolo della hash, dell'intero blocco, modificando volta per volta il valore *Nonce* all'interno dell'header, per ottenere hash diverse.
Una volta che un nodo ottiene una hash sotto il *target*, definito sempre all'interno dell'header, lo comunica agli altri nodi della rete, che verificano "la prova di lavoro" PoW.
Se il blocco passa questo controllo da parte degli altri nodi della rete, viene inserito nella blockchain e le transazioni interne confermate.
Essendo la rete asincrona, due nodi potrebbero comunicare un blocco valido approssimativamente nello stesso momento, generando una biforcazione nella catena.

5.5 Fork

E' possibile che diversi nodi concludano la validazione di un blocco approssimativamente nello stesso istante.
In questo caso non si possono inserire entrambi i nuovi blocchi nella catena, perchè inserito uno, l'altro non è più un blocco valido, non contenendo l'hash dell'header dell'ultimo blocco inserito.
Non si può accettare semplicemente chi arriva una frazione di secondo prima, a causa dell' asincronia della rete. Se due blocchi sono minati a una certa distanza temporale (il secondo è minato quando tutta la rete ha già ricevuto il primo), il secondo verrà rifiutato come blocco non valido (perchè non contiene la hash dell'ultimo blocco, ma del penultimo), ma se i due blocchi sono minati in tempi piu ravvicinati, è un problema (37).
In tal caso si crea una biforcazione (*fork*) della catena.

Quando si crea una fork, entrambi i blocchi sono considerati momentaneamente come possibili nuovi blocchi e i nodi sono invitati a continuare a minare su uno qualsiasi dei due rami.
Appena in una delle due biforcazioni viene validato ed aggiunto un nuovo blocco, i nodi tendono a spostarsi sulla biforcazione più lunga, che quindi ha maggiore probabilità di diventare quella definitiva.

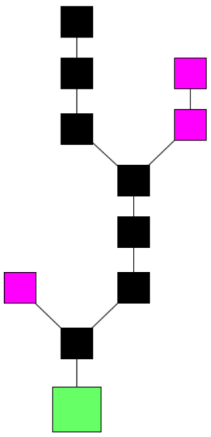

Figura 5.3: Schema di una blockchain con genesis block (verde), catena principale (nero) e blocchi abbandonati (viola).

Siccome le transazioni contenute nei blocchi abbandonati non saranno mai validate, i nodi non prenderanno la ricompensa per aver minato tali blocchi, nè le relative fees.
Per questo i nodi tendono fortemente a spostarsi sulla biforcazione piu lunga e le biforcazioni tendono a risolversi velocemente.
Le transazioni presenti in un blocco nella biforcazione che viene annullata tornano nel pool di transazioni da validare, se non sono già state inserite in un blocco dell'altro ramo della biforcazione.

Nella storia di Bitcoin non c'è mai stata una biforcazione più lunga di 6 blocchi e quindi mai una transazione con 6 conferme è stata annullata (il numero di conferme è il numero di blocchi successivi aggiunti alla catena).

5.5.1 Soft fork e Hard fork

Oltre alle fork momentanee generate automaticamente quando due blocchi vengono minati approssimativamente nello stesso momento, esistono altri tipi di biforcazioni, ma volontarie.

Le *Soft Fork* non sono delle vere e proprie biforcazioni della catena, ma semplicemente il risultato di aggiornamenti del protocollo preesistente (34).
Gli utenti che continuano a mantenere il vecchio software possono operare regolarmente e quindi le transazioni eseguite con il nuovo protocollo devono essere riconosciute come valida anche dagli utenti che non hanno aggiornato il software.

Le *Hard Fork*, invece, sono delle modifiche sostanziali al protocollo e quindi i nuovi blocchi non sono compatibili con il vecchio protocollo. Questo porta una vera e propria biforcazione, dove si hanno quindi due distinte blockchain, che continuano a svilupparsi autonomamente e che condividono però tutti i blocchi fino al momento della fork (37).

Bitcoin, per esempio, nella sua storia ha avuto varie hard fork, ma la maggior parte ha dato origine a una nuova blockchain e a una nuova comunità che è stata vitale

solo per poco tempo.
Bitcoin Cash è stata la prima hard fork significativa di Bitcoin fatta nel 2017.

Figura 5.4: Varie fork di Bitcoin (50).

5.6 Compensi nodi e le Fee

I nodi che riescono a risolvere la PoW e inserire un nuovo blocco nella catena ricevono le *fee* contenute in ciascuna transazione del blocco estratto e il *block reward* (37).
Le *fee* sono gli importi che chi scrive una transazione è disposto a pagare per far accettare la propria operazione.
Più sono alte le fee inserite da chi scrive una transazione, e più è probabile che la transazione venga inserita velocemente nella blockchain, quindi sono liberamente scritte al momento della compilazione della transazione.

Un altro scopo delle fee, oltre a compensare i nodi, è quello di rendere resistente la rete ad un attacco Danial of Service.

Se le fee non esistessero, chiunque potrebbe inviare alla rete di nodi numerose transazioni valide tra due indirizzi di sua proprietà, intasando il circuito e impedendo ad altri di fare transazioni.

La *block reward*, invece, è l'ammontare di bitcoin che vengono creati duranti l'inserimento di un nuovo blocco.
La *coinbase transaction* è la prima transazione che i nodi inseriscono in ciascun blocco che provano a minare e consente loro di indicare un suo indirizzo a cui inviare i compensi, nel caso riuscisse a risolvere la PoW prima degli altri e inserire il blocco nella catena.

La block reward ogni 210000 blocchi generati, quindi più o meno ogni quattro anni, si dimezza, processo conosciuto con il nome di *Halving*. Nel gennaio del 2009 era di 50 BTC, mentre ora è di 6.25 BTC, quindi ogni 10 minuti, approssimativamente (tempo di creazione di un nuovo blocco con il target del momento), vengono generati 6.25 nuovi bitcoin, il prossimo Halving sarà nel 2024 e abbasserà la block reward a 3.125 BTC.
La block reward, quindi, decresce esponenzialmente e il numero totale di bitcoin previsto è di 21 milioni: l'ultimo si prevede verrà estratto nel 2140, e da questa data in poi il compenso per l'inserimento di un blocco sarà dato esclusivamente dalla somma delle fee delle transazioni inserite. Il numero totale di bitcoin, quindi, segue l'andamento di una serie geometrica

$$210000 * 50 * \sum_{n=0}^{+\infty} \left(\frac{1}{2}\right)^n = 210000 * 50 * \frac{1}{1-\frac{1}{2}} = 21000000.$$

5.7 Proof of Work

Il protocollo di Proof of Work di Bitcoin, come anticipato, segue la procedura definita da HashCash. Il problema da risolvere sfrutta le proprietà delle funzioni Hash, che, grazie alla loro unidirezionalità, rende la risoluzione del problema molto difficile, ma allo stesso tempo facilmente verificabile se la soluzione proposta è corretta.

Per risolvere la PoW di Bitcoin, un nodo deve trovare il valore di *nonce*, parametro nell'header del blocco, che, inserito in esso, ne fa risultare la Hash più piccola di un valore target.
Il target ha dimensione 256 bits, la stessa dimensione dell'output della funsione SHA-256 e più il target è piccolo (inizia con tanti zeri), più sarà difficile trovare una hash più piccola di esso, e quindi più sarà difficile inserire il blocco in blockchain.
I passi che i nodi fanno durante il processo di mining sono i seguenti.

- Scelgono un valore x per nonce.
- Procedono facendo due volte la funzione di hashing SHA-256 sull'intero blocco, con il valore x nella sezione nonce dell'header.
- Verificano che il risultato della doppia hash sia inferiore al target definito nell'header del blocco.
- Se è verificata, comunicano il blocco alla rete che verificano la PoW e danno il consenso, se invece la hash trovata è maggiore del target, modificano x, per esempio possono procedere con $x + 1$ e ripetono tutta la procedura.

Il protocollo di Bitcoin prevede un aggiustamento periodico del valore target all'interno dei blocchi, per mantenere una velocità ottimale media di aggiunta.
Ogni 2016 blocchi, il protocollo aggiusta il valore target presente negli header dei blocchi, in quanto, se il tempo medio di inserimento dei 2016 blocchi precedenti è inferiore a 10 minuti, il protocollo aumenterà un po' la difficoltà di risoluzione abbassando il valore del target, mentre se il tempo medio di inserimento è superiore a 10 minuti, faciliterà l'inserimento dei prossimo 2016 blocchi, aumentando il valore target (26).

New_Target = Old_Target * (Actual Time of Last 2016 Blocks / 20160 minutes)

Dove appunto "20160 minutes" sarebbero i minuti se tutti i 2016 blocchi fossero minati precisamente ogni 10 minuti.

Capitolo 6

Blockchain 2.0

La blockchain introdotta da Bitcoin, con lo scopo di creare esclusivamente un sistema monetario digitale che andasse a sostituire quello fisico degli Stati, può essere considerata come primo utilizzo della blockchain e quindi definito con il termine *Blockchain 1.0*.
Nel 2015, con la nascita di Ethereum, si è cominciato a parlare di *Blockchain 2.0*, che si riferisce ad un nuovo modo d'impiego della Blockchain, in quanto la criptomoneta di Ethereum può essere certamente utilizzata come Bitcoin per le transazioni economiche, ma è soprattutto uno strumento per gestire quello che è sostanzialmente un "enorme computer globale", dove le applicazioni, una volta lanciate, non possono più essere bloccate. In questo modo, il mondo della Blockchain si è esteso a settori diversi da solo quello finanziario.

La piattaforma Ethereum, quindi, fornisce una macchina virtuale decentralizzata, la Ethereum Virtual Machine (EVM), che esegue script utilizzando una rete di nodi pubblici. E' un ambiente di programmazione molto avanzato, con la proprietà di essere Turing completa, quindi ha lo stesso potere computazionale di una macchina di Turing universale (37).

Due concetti importanti introdotti con la Blockchain 2.0 sono gli smart contract e i non-fungible token (NFT).
Il concetto di NFT deriva dal fatto che ogni token deve essere distinguibile da

tutti gli altri, opposto dei FT, fungible token, come per esempio 1 bitcoin, che è indistinguibile da un altro bitcoin.
Un NFT può essere associato a qualsiasi cosa, per esempio una canzone, un'immagine, un quadro o anche una proprietà virtuale. Inoltre può essere venduto o scambiato direttamente tra indirizzi in blockchain (58).

6.1 Smart Contract

Il principio degli *Smart Contract* è analogo a quello che sta alla base delle macchine dispensatrici di bevande e merendine: in corrispondenza di un pagamento monetario, automaticamente viene elargito un bene di consumo (23).
Nick Szabo, nel 1997, introdusse una prima idea degli smart contract, come anticipato nella Sezione 3.7, ma il concetto era ancora slegato dalla tecnologia Blockchain e l'obiettivo era quello di eseguire e velocizzare la negoziazione digitale.

Con la nascita di Ethereum, però, la tecnologia smart contract si è sviluppata ulteriormente ed è stata portata nei sistemi Blockchain (58).
Gli smart contract sono quindi dei software basati sulla blockchain, analoghi ai contratti legali nel mondo reale, ma costituiti da un codice crittografico che serve ad automatizzare l'esecuzione in modo che utenti sconosciuti e partecipanti decentralizzati possano essere immediatamente certi dell'esito, senza la necessità di intermediari e senza perdite di tempo (24).

Gli smart contract sono un tipo di account Ethereum, ovvero hanno un saldo e possono inviare transazioni in rete, ma non sono controllati da un utente, bensì sono distribuiti in rete ed eseguiti come programmato. Al verificarsi di alcuni eventi, essi si comportano in base a come sono definiti a livello di programmazione.
Alcune funzioni interne ad uno smart contract, secondo come è stato programmato, possono essere attivate dal creatore del contratto, l'*owner*, ma una volta che il contratto è stato distribuito sulla rete, se, per esempio, ci si è dimenticati di creare una funzione che sposti il saldo del contratto ad un altro indirizzo Ethereum, esso rimarrà bloccato per sempre all'interno del contratto.

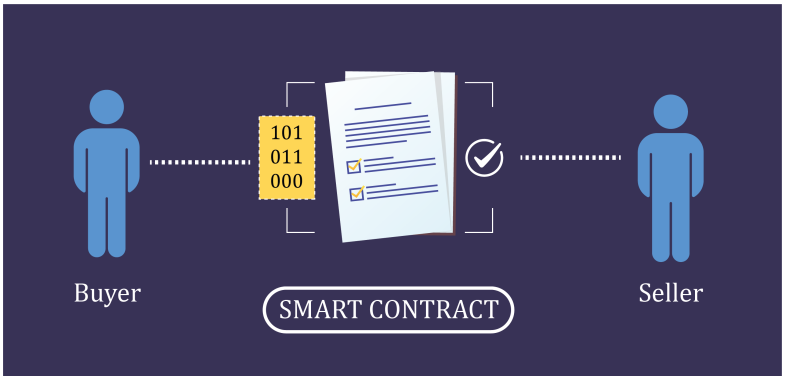

Figura 6.1: Schema smart contract.

Come un normale indirizzo Ethereum, anche gli indirizzi che si riferiscono ad uno smart contract sono pubblici e tutte le interazioni da parte degli utenti o le funzioni attivate dall'owner del contratto possono essere visualizzate e verificate da un qualsiasi utente, attraverso dei siti che permettono l'esplorazione dell'intera blockchain, per esempio Etherscan (3).

6.1.1 Scrittura

Uno smart contract è un contratto sotto forma di codice, che deve essere compilato dalla Ethereum Virtual Machine (EVM). Questa macchina virtuale esegue un formato speciale di codice, chiamata *bytecode EVM*, che però è molto ingombrante e difficile da leggere per i programmatori, i quali preferiscono sviluppare contratti usando un linguaggio ad alto livello e un compilatore per convertirli in bytecode (27).

Per la stesura di smart contract, quindi, si è scelto di creare dei linguaggi di programmazione dichiarativi e non imperativi, in quanto nei primi è possibile dichiarare la natura del risultato atteso, senza effetti collaterali e non ci sono modifiche allo stato al di fuori di una funzione e non si deve definire l'intera serie di procedure che combinano la logica e il flusso di un programma. In questo modo, è più facile capire come si comporta un programma, perché ogni parte è separata e può essere compresa isolatamente, cosa molto importante, perché nei smart contract, ogni errore costa denaro per essere ricompilato sulla rete (27).

In linguaggio più usato è *Solidity*, che però è imperativo, perché la maggior parte dei programmatori resiste al cambiamente e preferisce usare un linguaggio simile a Java-Script, C++, e Java, però ci sono anche altri linguaggi con cui si possono fare smart contract.

- *LLL*, primo linguaggio per scrivere smart contract su Ethereum, linguaggio dichiarativo, ma ora poco usato.

- *Serpent*, linguaggio con sintassi simile a Python, linguaggio imperativo.

- *Vyper*, anch'esso linguaggio con sintassi molto simile a quella di Python, ma usato più di frequente insieme a Solidity.

Il concetto di smart contract, oltre a comprendere il codice del programma, che diventa la logica contrattuale, racchiude anche i messaggi che gli utenti inviano al programma, che rappresentano gli eventi che fanno attivare il contratto (22).
Come già anticipato, la distribuzione di uno smart contract è tecnicamente una transazione sulla rete, quindi occorre pagare delle fee, proprio come quando si fa un semplice trasferimento di ETH, ma, essendo la dimensione della transazione molto più elevata, si avranno dei costi assai più elevati. Affinché la EVM possa interpretare e memorizzare uno smart contract, questo deve però essere compilato prima della distribuzione.

6.1.2 Remix

Remix (9) è un *integrated development environment (IDE)*. Questa tipologia di software è utilizzata dagli sviluppatori per la creazione di applicazioni aggregando tutti gli strumenti di cui si ha bisogno in un unico ambiente.
Attraverso Remix si può scrivere il codice sorgente di smart contracts usando utili funzionalità per agevolarne la stesura, e tramite determinate interfacce si può procedere alle fasi di compilazione, debug e distribuzione sulla blockchain di Ethereum.

6.2 Token

Un token (gettone) rappresenta un valore o un diritto ed è nella sostanza come una moneta virtuale con un utilizzo specifico (37).
La parola token viene usata per significati diversi:

- *Utility token*: danno diritto all'utilizzo di alcune features su una piattaforma,oppure a benefici o servizi premium;

- *Security token*: danno diritto di partecipazione al profitto della piattaforma che li ha emessi;

- *Payment token*: criptomonete.

Il concetto di token su blockchain esisteva già prima della nascita di Ethereum; per esempio, i bitcoin sono essi stessi dei token, e molte altre piattaforme basate su token sono state sviluppate negli anni a seguire.
L'introduzione, però, del primo standard token su Ethereum ha portato all'esplosione del loro utilizzo in questo ambito (27).

6.3 Ethereum Request for Comments

Gli *Ethereum Request for Comments (ERC)* sono standard tecnici proposti per Ethereum, disponibili online nei *Ethereum Improvement Proposals (EIP)* (2). Gli EIP includono anche le specifiche del protocollo di base, quindi gli standard per la piattaforma Ethereum, le API client e gli standard contrattuali.
Ci sono diversi Ethereum Request for Comments che sono stati proposti nel tempo, ognuno dei quali definisce uno standard o una convenzione a livello di applicazione per Ethereum inclusi, ma non tutti limitati esclusivamente per la creazione di *Token*.

Un contratto che segue uno standard ERC, quindi, è uno smart contract con una struttura dati prestabilita, per facilitare l'implementazione da parte degli sviluppatori e l'implementazione sulla blockchain di Ethereum.

6.3.1 ERC-20 e Fungible Token

Nel 2015, Vitalik Buterin e Fabian Vogelsteller definirono il *ERC-20*, che consente l'implementazione di un'API standard per i token all'interno di smart contracts, specificando l'interfaccia standard per la creazione dei *fungible token (FT)*.
Essendo token ERC-20 significa che le diverse unità sono intercambiabili e non hanno proprietà uniche: appunto da ciò deriva il nome di token fungibili (27).

L'interfaccia comprende una serie di funzioni e eventi che devono essere definite in ogni implementazione, per esempio, in Solidity, una specifica dell'interfaccia ERC-20 appare nel seguente modo (27).

```
contract ERC20 {
    function totalSupply() constant
            returns (uint theTotalSupply);
     function balanceOf(address _owner)
            constant returns (uint balance);
     function transfer(address _to, uint _value)
            returns (bool success);
     function transferFrom(address _from, address _to, uint _value)
            returns (bool success);
     function approve(address _spender, uint _value)
            returns (bool success);
     function allowance(address _owner, address _spender)
            constant returns (uint remaining);
     event Transfer(address indexed _from, address indexed _to,
            uint _value);
     event Approval(address indexed _owner, address indexed _spender,
            uint _value);
}
```

- *totalSupply*: ritorna il numero totale di unità del token che esistono in questo momento.
- *balanceOf*: dato un address, ritorna il bilancio dei token di questo address.
- *transfer*: dato un address e una quantità, trasferisce a quell'address la quantità di token stabilita.

- *transferFrom*: come transfer, ma si specifica anche l'address mittente.
- *approve*: dato un address, verifica che possa inviare token, in base alla fornitura restante.
- *allowance*: dato l'address del proprietario e di chi spende, restituisce l'importo rimanente che chi spende è autorizzato a ritirare dal proprietario.
- *Transfer*: evento registrato dopo una chiamata della funzione transfer o transferFrom, se riuscita.
- *Approval*: evento registrato dopo una chiamata della funzione approve, se riuscita.

6.3.2 ERC-721 e Non-Fungible Token

Nel 2018, William Entriken, Dieter Shirley, Jacob Evans e Nastassia Sachs proposero il *ERC-721*, che consente l'implementazione di un'API standard per i token all'interno di smart contracts, specificando l'interfaccia standard per la creazione dei *non-fungible token (NFT)* (39).
Questa tipologia di token sono, appunto, non fungibili, quindi non possono essere replicati, in quanto unici. Facendo il paragone con il mondo reale, possono essere considerati come terreni o opere d'arte, mentre un fungible token può essere comparato alla lattina di una bevanda, identica a qualsiasi altra lattina, perfettamente replicabile, dato che non si vedrebbe la differenza.

Gli NFT cercano quindi di replicare nel mondo del digitale ciò che sono i beni infungibili nel mondo reale; possono, per esempio, essere applicati a delle opere artistiche di natura digitale, attestandone l'autenticità e la proprietà (23).
Questo tipo di token trova la sua utilità su piattaforme che offrono oggetti collezionabili, chiavi di accesso, biglietti della lotteria, posti numerati per concerti o eventi sportivi ecc. (39).

La differenza fondamentale nella struttura dell'interfaccia di un token ERC-20 e un ERC-721, è che ERC-20 presenta una mappatura dove un address è la chiave primaria, che tiene traccia dei saldi che appartengono a ciascun address, mentre

ERC-721 tiene traccia di ciascun ID di ogni token e di chi lo possiede, quindi l'ID del token è la chiave primaria.

L'interfaccia ERC-721 comprende una serie di funzioni e eventi, alcune analoghe nello standard ERC-20 (27).

```
interface ERC721 /* is ERC165 */ {
    event Transfer(address indexed _from, address indexed _to,
            uint256 _deedId);
    event Approval(address indexed _owner, address indexed _approved,
            uint256 _deedId);
    event ApprovalForAll(address indexed _owner, address indexed
    _operator,
            bool _approved);
    function balanceOf(address _owner) external view
            returns (uint256 _balance);
    function ownerOf(uint256 _deedId) external view
            returns (address _owner);
    function transfer(address _to, uint256 _deedId) external payable;
    function transferFrom(address _from, address _to, uint256 _deedId
    )
            external payable;
    function approve(address _approved, uint256 _deedId) external
    payable;
    function setApprovalForAll(address _operateor, boolean _approved)
     payable;
    function supportsInterface(bytes4 interfaceID) external view
            returns (bool);
}
```

- *Transfer*: evento registrato dopo una chiamata della funzione transfer o transferFrom, se riuscita.
- *Approval*: evento registrato dopo una chiamata della funzione approve, se riuscita.
- *ApprovalForAll*: evento registrato dopo una chiamata della funzione setApprovalForAll, se riuscita.

- *balanceOf*: dato un address, ritorna il bilancio dei token di questo address.
- *ownerOf*: dato un ID univoco di un token, ritorna l'address che lo possiede.
- *transfer*: dato un address e l'ID di un token, trasferisce a quell'address il token.
- *transferFrom*: come transfer, ma si specifica anche l'address mittente.
- *setApprovalForAll*: dato un address, abilita o disabilita l'approvazione per la gestione da parte di esso.
- *supportsInterface*: interroga se un contratto implementa un'interfaccia.

Uno smart contract, basato su ERC-20 o ERC-721, per esempio, può essere fatto da un qualsiasi utente, e distribuito sulla blockchain di Ethereum pagando una fee, in quanto è visto come una transazione. A questo punto, il creatore diventerebbe l'owner del contratto e può facilmente dimostrare l'esistenza e la proprietà di risorse digitali sotto forma di video, immagini, arti, in base al progetto da lui creato. Il creatore, inoltre, secondo come è programmato il contratto, guadagna delle royalties, da lui settate, ogni volta che avviene un commercio di un bene della sua collezione su qualsiasi piattaforma NFT.

6.3.3 ERC-1155 e Multi Token

Successivamente, sempre nel 2018, Witek Radomski, Andrew Cooke, Philippe Castonguay, James Therien e Eric Binet proposero il *ERC-155*, che consente l'implementazione di un'API standard per i token all'interno di smart contracts, specificando l'interfaccia standard per contratti che gestiscono più tipologie di token. Un singolo contratto può quindi includere qualsiasi combinazione di fungible token e non-fungible token (2).

Questo standard, più generale dei primi due descritti, risolve alcune criticità presenti, per esempio semplifica i precedenti standard, che prevedevano la scrittura di uno smart contract per ogni token. Esso permette la scrittura di un unico contratto contenente i dati di configurazione di tutti i token, andando così a semplificare e anche ridurre il numero di transazione necessarie per scambiarli tra diversi utenti.

6.4 Dynamic Non-fungible token (dNFT)

Ogni NFT contiene un token identifier, un valore univoco che identifica l'NFT, e l'indirizzo del contratto dal quale è stato generato.
Inoltre un NFT può presentare alcuni metadata, che descrivono in dettaglio il contenuto del token; questi dati vengono solitamente archiviati off-chain come file JSON.
Tutti i parametri appena citati, per un NFT, non si possono modificare: esso può essere trasferito da un indirizzo ad un altro, ma i vari metadata dell'NFT non potranno mai essere cambiati (18).

Questa è proprio la differenza tra gli NFT e i dNFT, *Dynamic Non-fungible token*. Il codice che identifica l'NFT e il contratto da cui è stato generato non potranno comunque essere modificati in futuro; i metadata, invece, tramite opportune funzioni nello smart contract di distribuzione, potranno subire delle variazioni, in base a quanto codificato.
Inoltre, è possibile aggiornare anche manualmente i metadata dei dNFT o programmarli in modo che vengano aggiornati automaticamente quando si verificano determinati eventi: ciò significa che i dati che attivano le modifiche possono essere on-chain o offline.

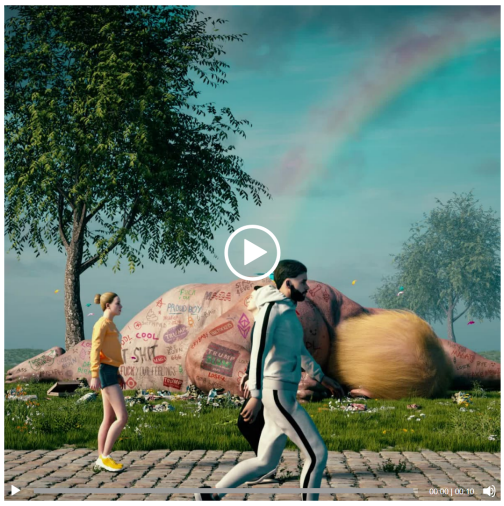

Figura 6.2: *"Crossroad"* con vittoria di Joe Biden (7).

Uno dei primi esempi di NFT dinamico è stato creato da Beeple.
In vista delle elezioni presidenziali degli Stati Uniti nel 2020, Beeple ha realizzato un NFT, *"Crossroad"*, che si sarebbe modificato in base all'esito delle elezioni.
Con la vittoria di Joe Biden, l'NFT generato è stato quello presente in Figura 6.2, dove si vede Donald John Trump sullo sfondo, disteso a terra.

6.5 Wallet

I wallets (portafogli) sono dispositivi, software o hardware, utilizzati dagli utenti per gestire e memorizzare non solo le criptovalute che possiedono, ma anche tutti i loro token, di qualsiasi tipo.
Lo scopo principale di un wallet è quello di conservare in modo sicuro le chiavi private per gestire i propri possedimenti salvati su blockchain. Questo perché è sbagliato dire che un individuo possiede un bitcoin, per esempio, o un determinato NFT, in quanto essi vivono e restano sulla blockchain specifica; è invece giusto definire che l'individuo possiede le chiavi private necessarie per la gestione dei suoi possedimenti. Sono suoi in quanto è l'unico in grado di gestirli, se è l'unico a conoscere le specifiche chiavi private.

Esistono diversi tipi di wallet, che possono essere suddivisi in varie categorie (37).

- *Desktop wallet*: software installati sul proprio computer desktop, in generale con dubbia sicurezza, perché dipende dall'attaccabilità del proprio PC.
- *Mobile Wallet*: molto simili alla versione desktop, ma installati sugli smartphone; il livello di sicurezza rimane dubbio, in quanto dipende dall'attaccabilità del proprio cellulare.
- *Web Wallet*: servizi accessibili tramite un browser web, senza dover installare nulla sui propri dispositivi. Le informazioni relative al wallet, però, come le chiavi private, sono memorizzate su server di proprietà di terzi, quindi la sicurezza potrebbe non essere assoluta.
- *Hardware Wallet*: dispositivi fisici che memorizzano offline tutti i propri dati; possono essere collegati al PC mediante porte USB e sono considerati molto sicuri.

Inoltre, un'altra distinzione che si può fare riguarda la custodia delle chiavi dei propri possedimenti sul wallet.
I *Custodian wallet* sono servizi di wallet che detengono per il cliente la custodia e la sicurezza delle chiavi, ma in questo modo è come se il cliente non possegga veramente i propri assets.
I *Non Custodian wallet*, invece, sono servizi di wallet in cui la custodia delle chiavi è a cura dell'utente, per avere una maggior sicurezza.

6.6 Transazioni in Ethereum

Le transazioni sulla rete Ethereum hanno delle diversità rispetto a quelle descritte su Bitcoin.

Le informazioni principali che l'oggetto di una transazione (39) contiene sono

```
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
  gasLimit: "21000",
  maxFeePerGas: "300",
  maxPriorityFeePerGas: "10",
  nonce: "0",
  value: "10000000000"
}
```

I campi *from* e *to* indicano, rispettivamente, l'indirizzo del mittente e del destinatario. I successivi tre campi si riferiscono al gas utilizzato nella transazione: *gasLimit* specifica il limite massimo che può essere consumato durante l'esecuzione; *maxFeePerGas* è la commissione massima per unità di gas che si desidera pagare; *maxPriorityFeePerGas*, invece, è la tariffa massima prioritaria disposta a pagare (utile per accelerare l'inclusione della transazione in tempi brevi, in momenti di congestione della rete).
Il campo *nonce* è solo un indicatore del numero di transazioni inviate dall'indirizzo mittente e *value* è l'ammontare di Ether inviati (39).

L'oggetto di una transazione appena descritto dovrà essere successivamente firmato dal mittente, utilizzando la propria chiave privata, per autenticarlo e trasmettere la transazione alla rete.

Esistono diverse tipologie di transazione sulla rete Ethereum:

- *Transazioni Regolari*, tra normali address di due utenti.

- *Distribuzione di un contratto*, transazione senza l'address nel campo *to*, in quanto si sta distribuendo un contratto sulla rete ed il campo dati è usato per il codice dello smart contract.

- *Esecuzione di un contratto*, transazione che interagisce con un contratto già distribuito, quindi nel campo *to* sarà presente l'address dello smart contract.

6.7 Blockchain 3.0

La tecnologia Blockchain, quindi, fu inizialmente proposta per creare denaro digitale, criptovalute, come Bitcoin, e le relative applicazioni in questo ambito sono spesso etichettate come *Blockchain 1.0*. Uno dei aspetti più importanti di questa prima versione della Blockchain era la decentralizzazione di trasferimenti finanziari peer-to-peer sicuri e trasparenti tra entità che non si conoscevano, quindi era tutto basato sulla decentralizzazione della moneta e la creazione del denaro digitale (54).

Con gli anni, sviluppando molte applicazioni più complesse e dirompenti, nacque la *Blockchain 2.0*, che, come definita all'interno del capitolo, porta questa tecnologia su altri fronti, con la creazione di smart contracts, organizzazioni autonome decentralizzate (DAO), token e molto altro, andando oltre le semplici transazioni finanziarie.

Attualmente, si sta assistendo all'era della *Blockchain 3.0*, a cui si possono collegare tutte le applicazioni di questa tecnologia al di fuori delle sole criptovalute e smart contracts. Ci sono casi applicativi in cui la Blockchain può essere utile o addirittura avere un ruolo dirompente, ma questo accade solo in certe situazioni.
Ha senso prendere in considerazione l'utilizzo di una soluzione basata su blockchain se l'azienda in questione ha almeno una di queste necessità:

- *Ecosistema economico*: gestire transazioni/relazioni tra un alto numero di imprese e organizzazioni.

- *Eterogeneità*: imprese e organizzazioni coinvolte di tipologia molto varia.

- *Mancanza di fiducia*: le realtà coinvolte non si conoscono e non hanno rapporti consolidati.

- *Disponibilità a condividere i dati*: accordo sulla necessità di condividere dati e informazioni in modo strutturale.
- *Alto livello di digitalizzazione.*
- *Trasparenza e immutabilità*: all'ecosistema in questione devono essere necessarie, o almeno utili, l'immutabilità dei dati e la loro trasparenza.
- *Governance*: serve la disponibilità a condividere regole, comportamenti, valori, processi.

Questa evoluzione continua a sviluppare il panorama tecnologico e crea nuove opportunità in settori molto diversi tra di loro: Finanza decentralizzata (DeFi), Assicurazioni, Agroalimentare, Turismo, Sanità, Giochi digitali, e molto altro.

Capitolo 7

Applicazione in ambito sanitario

I paragrafi che seguiranno avranno come oggetto una proposta di applicazione della tecnologia Blockchain in ambito sanitario, dopo aver descritto alcune possibili soluzioni per la creazione di una blockchain nazionale.

7.1 Sanità odierna

Il sistema sanitario odierno presenta alcune problematiche che devono essere risolte con soluzioni innovative e sostenibili.
La crescente necessità di risorse finanziarie, dovuta all'aumento dei costi delle cure, l'accesso limitato ai servizi medici e la gestione dei dati dei pazienti sono solo alcuni dei punti da migliorare.

Molte delle informazioni dei pazienti sono ancora memorizzate in modo centralizzato negli ospedali tramite forme tradizionali di archiviazione, con le relative problematiche, per esempio il singolo punto di guasto o la vulnerabilità alla manipolazione. Un database centralizzato espone il sistema a maggiori attacchi informatici che minacciano la privacy e la sicurezza dei dati dei pazienti. Inoltre, in questi sistemi, i dati possono perdersi in modo permanente quando vengono eliminati dal database di appartenenza.

Un altro problema è la riproducibilità dei dati, per ottenere altri giudizi da medici e ospedali esterni, o semplicemente per la condivisione, avendo standard e formati diversi.
Una malattia comune, la maggior parte delle volte, è trattata nello stesso modo per pazienti diversi. Il metodo in alcuni casi è controproducente, ma necessario, non avendo l'intera cartella clinica dei pazienti in un unico posto, per definire un trattamento più adatto al singolo individuo.

Il paziente, inoltre, nel sistema odierno, non ha il possesso dei propri dati, che sono proprietà dell'ospedale e quindi non sa come vengono gestiti e utilizzati, con la possibilità di avere dei replicati in ospedali diversi, per una diversa formattazione dei dati stessi.

L'applicazione della Blockchain in questo campo, con tutte le sue proprietà, offre un potenziale significativo per affrontare alcune delle problematiche presenti. Con un'architettura decentralizzata e la crittografia che ne sta alla base, migliora la gestione dei dati, la sicurezza e la privacy delle informazioni e l'efficienza dei processi. I dati dei pazienti potrebbero essere gestiti interamente in un unico record di proprietà del paziente, che contiene tutti i servizi sanitari passati e presenti, come allergie, vaccini, medicinali, interventi ecc. In questo modo si avrebbe un accesso immediato per l'utilizzo, sempre e ovunque.

7.2 Blockchain nel settore pubblico

L'applicazione della Blockchain nel settore pubblico è un processo sicuramente molto complesso, sia dal punto di vista tecnico che normativo, ma con una corretta implementazione può portare a grandi miglioramenti nell'efficienza, trasparenza e sicurezza delle operazioni statali e governative.

Nel settore pubblico, ogni decisione sulle regole della blockchain deve essere allineata con gli obiettivi politici, i valori pubblici, il quadro istituzionale e le aspettative della società.

L'attenzione principale di una blockchain di questo tipo è principalmente la registrazione, condivisione e la sicurezza dei dati, oltre che l'automazione dei processi tramite smart contract; quindi non è necessaria l'introduzione di una criptomoneta, aspetto assolutamente facoltativo in un contesto governativo.

La proposta potrebbe ricadere su una *blockchain privata permissioned*, più flessibile di una blockchain pubblica, con creazione dei blocchi consentita ad un numero ristretto di nodi.
Inoltre, organizzazioni diverse necessitano di esigenze differenti: in alcuni casi la privacy delle transazioni e la sicurezza dell'infrastruttura potrebbero essere il ruolo principale, mentre per altre la trasparenza e l'immutabilità sono la massima motivazione.
Un ente specifico o un'azienda, quindi, può beneficiare dell'utilizzo della tecnologia di base della blockchain definita; tramite *applicazioni decentralizzate (DApp)* e smart contract, è possibile creare *decentralized autonomous organizations (DAO)*, dove le regole sono definite nei contratti che le compongono.
Alcuni smart contract, per esempio, potrebbero essere scritti per mantenere determinate informazioni private o accessibili a un numero ristretto di partecipanti, mentre altre potrebbero essere rese pubbliche per questioni di trasparenza (40).

Un'azienda può quindi decidere di basare la sua organizzazione su una blockchain, appoggiandosi alla blockchain nazionale o richiedere un servizio da parte di, per esempio, Hyperledger Fabric, framework open source per la creazione di blockchain permissioned per applicazioni aziendali (4).
Nel progetto che si andrà a definire in questo elaborato, si considererà l'implementazione sulla blockchain nazionale, ma non esistendo, ed essendo comunque al di fuori del presente studio, non si possono conoscere nel dettaglio gli standard e i protocolli che la governeranno. Per questo motivo gli smart contract che si andranno a definire, per semplicità, saranno fatti usando il linguaggio di programmazione Solidity, seguendo le codifiche della blockchain di Ethereum.
Si definiscono solo alcuni concetti di importanza.

- **Scalabilità**: Una blockchain nazionale deve essere in grado di gestire un volume molto elevato di transazioni in modo efficiente.

Esistono varie strategie per gestire la scalabilità di una blockchain di questo tipo: una di esse è l'utilizzo di algoritmi di consenso ottimizzati. La *Proof of Authority PoA* può essere una scelta appropriata per migliorare nell'ambito contestuale.
L'efficienza di questo algoritmo è molto elevata, perché i nodi sono autorizzati e di fiducia, quindi possono confermare le transazioni rapidamente senza la necessità di risolvere complicati problemi crittografici, come nel caso di Bitcoin con la Proof of Work.
Per risolvere la scalabilità, basterà che l'autorità centrale nomini nuovi nodi all'occorrenza, espandendo la capacità della rete in modo flessibile.

- **Sicurezza e Privacy**: L'utilizzo di sistemi crittografici che si basano su ECC e ECDSA è un'opzione solida per definire la sicurezza della blockchain.
  La scelta della curva ellittica è un punto cruciale: chiavi private almeno a 256 bit, parametri non grossi e altre accortezze migliorano la sicurezza e la generazione delle chiavi. L'utilizzo di curve standard, magari anche quella di Bitcoin, potrebbe essere una soluzione possibile. In ogni caso la gestione delle chiavi può essere simile a quella di Bitcoin, quindi si genera una chiave privata da cui vengono generati: chiave pubblica, indirizzo e seed phrase.
  Alcune altre curve standard, per esempio, sono: Secp256r1, Secp384r1, entrambe raccomandate dal NIST per la crittografia, rispettivamente con chiavi di 256 e 384 bit; Secp521r1, con chiavi a 521 bit, maggiore sicurezza ma risorse computazionali più elevate.

7.3 Proposta Blockchain nazionale

La blockchain nazionale avrà determinate caratteristiche, per permettere a tutti gli enti pubblici che lo richiedono, appartenenti ad ambiti diversi tra di loro, di migliorare la loro situazione presente con questa tecnologia.

La scelta di una *blockchain permissioned* è la miglior soluzione, in quanto non sarebbe opportuno permettere a qualsiasi utente di inserire i blocchi e validare le transazioni, come invece succede con le blockchain permissionless. Lo stato concederà i permessi a dei nodi specifici, che, per esempio, potrebbero essere almeno due

per regione, situati in determinati punti di interesse.
Ogni nodo autorizzato, quindi, avrebbe una copia dell'intera blockchain e potrebbe partecipare al processo di validazione e inserimento delle transazioni nei blocchi da aggiungere alla catena.
Questo è anche un punto a favore per quanto riguarda la decentralizzazione, a differenza dell'uso di una struttura privata per un singolo ente, che, sfrutterebbe la tecnologia alla base e la crittografia, ma sarebbe praticamente centralizzato.
Per definire i nodi che saranno autorizzati a inserire i blocchi e validare le transazioni della blockchain, lo Stato può decidere alcuni luoghi di interesse che saranno configurati in base al software e alle specifiche della blockchain creata.
Se ci fossero dei problemi, in futuro, con determinati nodi, l'algoritmo alla base della blockchain può escluderli, non essendo tutti fondamentali al fine di gestire l'intera infrastruttura.

La scelta di una *blockchain privata* rispetto ad una pubblica, invece, è motivo di sicurezza e privacy per gli enti che poi la utilizzeranno; inoltre le blockchain private offrono maggiore flessibilità e controllo quando si tratta di gestire la visibilità dei dati.
Le organizzazioni possono fissare le regole e le autorizzazioni per l'accesso ai loro dati, quindi decidere di condividere tali dati in modo selettivo e pubblico. E' importante però che la blockchain sia definita già inizialmente per permettere questa flessibilità e le autorizzazioni devono essere specificate in modo adeguato all'inizio del processo di creazione di ogni DAO, per garantire che in futuro sia possibile cambiare la visibilità dei dati senza problemi.
In questi termini è come se la blockchain definita fosse ibrida, rispetto a privata.

Ogni blockchain deve avere il proprio metodo di consenso per l'inserimento dei blocchi nella catena. Con la caratteristica di essere permissioned, l'efficienza è molto alta rispetto alle blockchain permissionless, in quanto i nodi autorizzati sono scelti dallo Stato e quindi sicuri. In questo modo non è necessario che i nodi svolgano lavori computazionalmente dispendiosi, per provare la loro lealtà, ma basta che verifichino che le transazioni da inserire in un blocco siano corrette.
Il primo metodo presentato potrebbe essere la *Proof of Authority PoA*, ma anche

altre tipologie di consenso potrebbero essere attuate.

Per distribuire gli address personali agli individui che vogliono usufruire dei servizi erogati dagli enti che si appoggiano alla blockchain nazionale, la modalità proposta è tramite il *Sistema Pubblico di Identità Digitale SPID*. Ogni persona, attraverso le personali credenziali SPID, potrà accedere ad un'area riservata per la generazione del proprio indirizzo, con la relativa chiave privata e pubblica.
In questo modo, ogni individuo avrebbe un unico address legato alla propria persona, al fine di evitare che un utente crei molteplici indirizzi, come è possibile, per esempio, sulla blockchain di Bitcoin o Ethereum.

In questo progetto si è considerata l'implementazione di una blockchain nazionale nel settore pubblico, ma il concetto potrebbe essere esteso, con delle modifiche, a livello europeo o mondiale: i nodi amministratori verrebbero distribuiti in ogni Stato partecipante, con una maggioranza di nodi presenti negli Stati più popolosi.

7.4 Proposta progetto sanità

Il progetto che si andrà a studiare in questo elaborato è relativo alla sanità pubblica, con scopo principale la creazione di una infrastruttura basata sulla tecnologia Blockchain. In particolare, uno degli obiettivi principali è quello di creare un sistema di token per le cartelle cliniche dei pazienti in formato dNFT.
La differenza tra dNFT e NFT risiede nella modifica futura: in generale gli NFT non permettono modifiche successive, essendo statici, mentre i dNFT, essendo dinamici, consentendo modifiche future. In questo modo, ogni volta che un individuo effettua, per esempio, una visita, vedrà il proprio token aggiornarsi.

Ogni utente, quindi, avrà la propria cartella clinica in formato di token dNFT: all'interno ci saranno una sezione riservata al proprio profilo personale e altre con tutte le informazioni riconducibili a lui nel campo della sanità, per esempio malattie diagnosticate, farmaci assunti, visite, allergie, vaccini, suggerimenti del medico, immagini, scansioni effettuate e altro ancora. Si avrebbero tutte le informazioni rilevanti raccolte in un unico luogo, pronte per l'utilizzo e per massimizzare la

comodità e l'accessibilità.
Tutti questi dati, definiti come metadati, racchiusi in un unico token, hanno una dimensione non banale, motivo per cui la maggior parte di essi viene mantenuta off-chain, mentre in blockchain vengono inserite le hash di tali dati, per garantire scalabilità ed efficienza. Un'altra possibilità potrebbe essere quella di definire un sottoinsieme di metadati da mantenere on-chain e un altro, con dimensioni maggiori, da archiviare off-chain, per non appesantire molto le transazioni e quindi i blocchi della catena.

In questo modo, l'utente avrebbe la completa proprietà della propria cartella clinica, con la possibilità di condividere in modo anonimo le sue informazioni con aziende terze e centri di ricerca. Inoltre, a discrezione dell'utente stesso, ci sarebbe la possibilità di settare e guadagnare delle royalties per la condivisione dei propri dati.
L'accesso alle cartelle cliniche off-chain, salvate con una procedura simile a quella IPFS, può essere fatto tramite un indirizzamento basato sul contenuto, quindi i costi della larghezza di banda possono essere ridotti, velocizzato il download dei record e un grande volume di dati può essere distribuito senza duplicazione, risparmiando notevole spazio di archiviazione. Inoltre, la hash di un file non può essere modificata senza modificare il file stesso: può essere considerato come un meccanismo di archiviazione immutabile (55).

Per definire brevemente il funzionamento di IPFS, *InterPlanetary File System*, si deve pensare ad un sistema peer-to-peer per distribuire contenuti su Internet in modo decentralizzato.
Ogni file caricato, dopo averne calcolata la hash, viene spezzato in blocchi di dati più piccoli, anch'essi identificati da un hash crittografico unico, che rappresenta il loro contenuto. Questi blocchi sono poi distribuiti sui nodi IPFS della rete del sistema.
Successivamente, quando un file è richiesto da un utente specificando il suo hash, il sistema ricerca i blocchi che lo compongono nei nodi più vicini, ed essendo il sistema peer-to-peer, si ha un'ottima velocità di condivisione.

7.4.1 Definizione organizzativa

Per prima cosa, è da definire l'organizzazione alla base del collegamento della sanità pubblica con la blockchain nazionale.
L'implementazione di una Decentralized Autonomous Organizations, per rappresentare l'intero settore sanitario, è il primo punto da considerare. Essa svolgerà un ruolo fondamentale nella gestione di tutte le operazioni.
I responsabili degli ospedali o delle istituzioni sanitarie che aderiranno all'iniziativa saranno i membri chiave, amministratori della DAO, che rappresenteranno gli interessi delle rispettive istituzioni e potranno partecipare ai processi decisionali dell'organizzazione.
Ognuno di essi riceverà un NFT che riprodurrà la sua identità nella DAO e attraverso il quale potrà partecipare a tutti i processi organizzativi.

A seguire si mostreranno alcuni esempi di smart contract, definiti tramite Solidity, con gli standard di Ethereum, utilizzando l'IDE Remix 6.1.2.
I contratti relativi alla creazione di token avranno come base i contratti standard di *OpenZeppelin* (6), mentre per la documentazione delle parole chiave del linguaggio Solidity si fa riferimento al relativo sito principale (11).

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.21;

// Import: Importare standard token ERC1155
import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";

contract SmartDAOtoken is ERC1155 {

    address public owner; // Owner del contratto, definito
    inizialmente dal costruttore
    // Uri dove saranno salvati i metadata per i relativi tipi di
    token
    string public uriToken1 = "https://ipfs.io/ipfs/hospitalsToken/";
    string public uriToken2 = "https://ipfs.io/ipfs/
    emergencyRoomToken/";
    string public uriToken3 = "https://ipfs.io/ipfs/
    medicalOfficeToken/";
```

```
14
    // Mapping: Nome di uno specifico token
16  mapping(uint256 => string) public tokenNames;

18  // Mapping: Supply corrente di uno specifico token
    mapping(uint256 => uint256) private _currentSupply;
20
    // Mapping: URI di uno specifico token
22  mapping(uint256 => string) private tokenURIs;

24  // Modificatore: per definire l'identificatore onlyOwner
    modifier onlyOwner() {
26      require(msg.sender == owner, "Solo l'owner puo' chiamare
    questa funzione.");
        _;
28  }

30  // Costruttore: Inizializza il contratto
    constructor() ERC1155("") {
32      owner = msg.sender;

34      _currentSupply[1] = 0;
        _currentSupply[2] = 0;
36      _currentSupply[3] = 0;

38      tokenNames[1] = "HospitalsToken";
        tokenNames[2] = "EmergencyRoomToken";
40      tokenNames[3] = "MedicalOfficeToken";
    }
42
    // Funzione: Mint di un token ad un address
44  // Nel nostro caso specifico amount sara' sempre 1, quindi sopra
    lo si omette e nel corpo si fissa a 1
    function mintTokenDAO(address _address, uint256 tokenId) external
    onlyOwner {
46      require(tokenId >= 1 && tokenId <= 3, "Token ID deve essere
    tra 1 e 3.");
        require(balanceOf(_address, 1) != 1 &&
48              balanceOf(_address, 2) != 1 &&
                balanceOf(_address, 3) != 1,
```

```
50                "Non e' possibile possedere piu' di un token.");
        _mint(_address, tokenId, 1, "");
52          _currentSupply[tokenId] += 1;
        tokenURIs[tokenId] = getUri(tokenId);
54      }

56      // Funzione: Ritorna Uri di un token dato l'ID
        function getUri(uint256 tokenId) public view returns (string
    memory) {
58          require(tokenId >= 1 && tokenId <= 3, "Token ID deve essere
    tra 1 e 3.");
            if (tokenId == 1) {
60              return uriToken1;
            } else if (tokenId == 2) {
62              return uriToken2;
            } else if (tokenId == 3) {
64              return uriToken3;
            } else {
66              return "Error";
            }
68      }

70      // Funzione: Rimuovere token da un address
        // Per bloccare il suo accesso alla DAO
72      function removeTokenFromAddress(address _address, uint256 tokenId
    ) public onlyOwner {
            require(balanceOf(_address, tokenId) == 1, "Token non
    appartiene all'indirizzo.");
74          // safeTransferFrom(_address, address(this), tokenId, 1, "");
        // Trasferisci il token al contratto
            _burn(_address, tokenId, 1);
76      }

78      // Funzione: Trasferire il controllo del contratto
        function transferOwnership(address newOwner) public onlyOwner {
80          require(newOwner != address(0), "Inserire un address corretto
    .");
            owner = newOwner;
82      }
```

```
}
```

Con questo contratto si sono definiti tre tipi di token ERC-1155 che saranno consegnati agli indirizzi delle diverse organizzazioni sanitarie.
Al momento si è pensato di inserire esclusivamente *HospitalsToken*, *EmergencyToken* e *MedicalOfficeToken*, rispettivamente token per appartenere alla DAO di ospedali, pronto soccorso e studi medici, ma si potrebbero aggiungere altre varianti. Con le specifiche fissate, il loro peso sulle decisioni della DAO è uguale, ma può essere modificato in base all'importanza del token.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.21;

// Import: Importare il contratto dei token della DAO
import "./SmartDAOtoken.sol";

contract SmartDAO {

    address public owner; // Owner del contratto, definito
    inizialmente dal costruttore
    SmartDAOtoken public tokenDAOContract; // Variabile contratto per
    l'uso delle funzioni interne
    Proposal[] public proposals; // Elenco di tutte le proposte
    uint durationProposal = 5 days; // Durata della finestra di tempo
    per votare una proposta

    // Componenti di una proposta da fare alla DAO
    struct Proposal {
        uint256 proposalId; // ID proposta
        address proposer; // Address di chi fa la proposta
        string description; // Descrizione della proposta
        uint256 votesYes; // Numero di voti a favore della proposta
        uint256 votesNo; // Numero di voti contro la proposta
        uint256 startTime; // Inizio finestra temporale per votare
        uint256 endTime; // Fine finestra temporale per votare
        address[] voters; // Elenco di votanti
```

```
24          bool executed; // Inizialmente false, se poi la proposta sara
    ' eseguita diventera' true
    }
26
    // Mapping: Per la verifica che un address puo' avere una sola
    proposta attiva alla volta
28  mapping(address => bool) public activeProposals;

30  // Evento: Proposta creata
    event ProposalCreated(uint256 indexed proposalId, address indexed
     proposer, string description);
32
    // Evento: Proposta votata
34  event ProposalVoted(uint256 indexed proposalId, address indexed
    voter);

36  // Evento: Proposta eseguita
    event ProposalExecuted(uint256 indexed proposalId, address
    indexed proposer);
38
    // Modificatore: per definire l'identificatore onlyOwner
40  modifier onlyOwner() {
        require(msg.sender == owner, "Solo l'owner puo' chiamare
    questa funzione.");
42      _;
    }
44
    // Modificatore: per definire i possessori dei token della DAO
46  modifier onlyDAOTokenHolder() {
        require(tokenDAOContract.balanceOf(msg.sender, 1) == 1 ||
48              tokenDAOContract.balanceOf(msg.sender, 2) == 1 ||
                tokenDAOContract.balanceOf(msg.sender, 3) == 1,
50              "Solo i possessori dei token della DAO possono
    chiamare questa funzione.");
        // "balanceOf" e' una funzione presente nello standard
    ERC1155, non scritta esplicitamente in SmartDAOtoken
52      // "== 1" in quanto, nel nostro caso, ogni partecipante avra'
     un solo ed unico token
        _;
54  }
```

```
56      // Costruttore: Inizializza la DAO
        constructor(address _tokenDAOContract) {
58          tokenDAOContract = SmartDAOtoken(_tokenDAOContract);
            owner = msg.sender;
60      }

62      // Funzione: Creare Proposta
        function createProposal(string memory _description) external
    onlyDAOTokenHolder {
64          require(!activeProposals[msg.sender], "Presente gia' una
    proposta attiva per questo address.");

66          activeProposals[msg.sender] = true; // Attivazione di una
    proposta per l'address presente
            Proposal memory newProposal = Proposal({
68              proposalId: proposals.length,
                proposer: msg.sender,
70              description: _description,
                startTime: block.timestamp, // Timestamp del blocco
    presente
72              endTime: block.timestamp + durationProposal, // Una
    proposta puo' essere votata per una definita durata
                votesYes: 0,
74              votesNo: 0,
                voters: new address[](0),
76              executed: false
            });
78          proposals.push(newProposal); // Aggiunta della nuova proposta
     nell'elenco proposte

80          emit ProposalCreated(newProposal.proposalId, msg.sender,
    _description); // Evento di proposta creata
        }
82
        // Funzione: Votare Proposta
84      function vote(uint256 _proposalId, bool _vote) external
    onlyDAOTokenHolder {
            Proposal storage proposal = proposals[_proposalId]; // Per
    una questione di leggibilita'
```

```
86
        require(block.timestamp >= proposal.startTime && block.
    timestamp <= proposal.endTime, "Proposta non attiva.");
88      require(!alreadyVote(proposal.voters,msg.sender), "Proposta
    gia' votata.");
        require(!proposal.executed, "Proposta gia' eseguita.");
90
       if (_vote) {
92          proposal.votesYes ++;
        } else {
94          proposal.votesNo ++;
        }
96
        proposal.voters.push(msg.sender); // Aggiunta del votante
    alla lista votanti della proposta
98      emit ProposalVoted(_proposalId, msg.sender); // Evento di
    proposta votata
     }
100
     // Funzione: Controllo se un address ha gia' votato
102  function alreadyVote(address[] memory addresses, address _address
    ) private pure returns (bool) {
        for (uint i = 0; i < addresses.length; i++) {
104         if (addresses[i] == _address) {
                return true;
106         }
        }
108     return false;
     }
110
     // Funzione: Eseguire Proposta
112  function executeProposal(uint256 _proposalId) external
    onlyDAOTokenHolder  {
        Proposal storage proposal = proposals[_proposalId]; // Per
    una questione di leggibilita'
114
        require(block.timestamp > proposal.endTime, "Finestra di
    votazione ancora attiva.");
116     require(proposal.votesYes > proposal.votesNo, "La proposta
    non ha ricevuto abbastanza voti positivi.");
```

```
        require(!proposal.executed, "Proposta gia' eseguita.");
118
        proposal.executed = true; // Proposta eseguita
120     activeProposals[proposal.proposer] = false; // Il proposer
    non ha piu' proposte attive
        emit ProposalExecuted(_proposalId, proposal.proposer); //
    Evento di proposta eseguita
122     }

124     // Funzione: Trasferire il controllo del contratto
        function transferOwnership(address newOwner) public onlyOwner {
126     require(newOwner != address(0), "Inserire un address corretto
    .");
        owner = newOwner;
128     }

130 }
```

Con questo contratto, invece, si sono definite l'organizzazione e la governance alla base della DAO del sistema sanitario.
Ogni utente che possiede un token amministrativo della DAO può partecipare ai processi di creazione, votazione ed esecuzione di proposte. In questo modo le decisioni vengono prese considerando tutti gli enti che appartengono all'organizzazione, dando la possibilità di proporre ulteriori migliorie anche a partecipanti minori della rete.

La creazione di questi contratti e la distribuzione dei token per appartenere alla DAO è compito dello Stato, che attualmente, insieme alle regioni, tutela la salute e gestisce la sanità. Successivamente, la DAO sarà amministrata internamente dai partecipanti.

Ogni ospedale, clinica, ecc. avrà un NFT, per rappresentarlo all'interno della DAO e, uno dei compiti sarà quello di creare i token per medici e operatori sanitari che lavorano nella loro struttura.
Se un utente inizia a lavorare in una determinata struttura, il suo NFT viene creato e consegnato, mentre se cambia luogo lavorativo, e quindi ha già un token

personale, viene solo modificato un determinato metadata interno, che si riferisce al luogo di lavoro (modifica che dovrà apportare il nuovo ospedale).
Questi NFT, in base all'utente, rappresenteranno la loro identità digitale; di conseguenza, dispongono di differenti livelli di funzionalità, in base al ruolo ricoperto nella struttura. Ad esempio, un infermiere potrebbe avere un accesso meno esteso nello smart contract e quindi effettuare meno funzioni rispetto ad un medico specializzato.
Di seguito si riporta un esempio di contratto per la creazione di token a servizio degli operatori sanitari.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.21;

// Import: Importare standard token ERC721
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
// Import: Importare il contratto dei token della DAO
import "./SmartDAOtoken.sol";

contract SmartDOCTtoken is ERC721 {

    address public owner; // Owner del contratto, definito
    inizialmente dal costruttore
    SmartDAOtoken public tokenDAOContract; // Variabile contratto per
     l'uso delle funzioni interne
    uint256 private tokenIdCounter; // Contatore dei token mintati

    // Mapping: Lega ID univoco dell'NFT con i propri metadata
    mapping(uint256 => string) public metadataToken;

    // Struct: Tiene traccia delle modifiche dei metadata dei token
    struct MetadataChange {
        string metadata;
        uint256 timestamp;
        address changer;
    }
    // Mapping: Lega ID univoco dell'NFT alla struttura
    mapping(uint256 => MetadataChange[]) public metadataHistory;
```

```
26
    // Modificatore: per definire l'identificatore onlyOwner
28  modifier onlyOwner() {
        require(msg.sender == owner, "Solo l'owner puo' chiamare
    questa funzione.");
30      _;
    }
32
    // Modificatore: per definire i possessori dei token della DAO
34  modifier onlyDAOTokenHolder() {
        require(tokenDAOContract.balanceOf(msg.sender, 1) == 1 ||
36              tokenDAOContract.balanceOf(msg.sender, 2) == 1 ||
                tokenDAOContract.balanceOf(msg.sender, 3) == 1,
38              "Solo i possessori dei token della DAO possono
    chiamare questa funzione.");
        // "balanceOf" e' una funzione presente nello standard
    ERC1155, non scritta esplicitamente in SmartDAOtoken
40      // "== 1" in quanto, nel nostro caso, ogni partecipante avra'
     un solo ed unico token
        _;
42  }

44  // Costruttore: Inizializza il contratto
    constructor(address _tokenDAOContract) ERC721("","") {
46      tokenDAOContract = SmartDAOtoken(_tokenDAOContract);
        owner = msg.sender;
48      tokenIdCounter = 1;
    }
50
    // Funzione: Modificare i metadata del token tokenId con i nuovi
    metadata
52  // Tiene traccia anche di quando e' stata fatta la modifica e da
    chi (address), inoltre in questo modo abbiamo uno storico dei
    Metadata di un token
    function updateMetadata(uint256 tokenId, string memory
    newMetadata) external onlyDAOTokenHolder {
54      require(keccak256(abi.encodePacked(getMetadata(tokenId))) !=
    keccak256(abi.encodePacked("Rimosso")), "Impossibile modificare i
    metadata di un token rimosso.");
        address changer = msg.sender;
```

```
56          metadataHistory[tokenId].push(MetadataChange(newMetadata,
    block.timestamp, changer));
            metadataToken[tokenId] = newMetadata;
58      }

60      // Funzione: Ritorna i metadata precedenti di un nft (la versione
         index)
        // Per esempio se questo nft ha avuto 3 modifiche dei metadata,
    metadataHistory contiene 3 elementi e con index=1 ci dara' la
    versione 2 dei metadata
62      function getPreviousMetadata(uint256 tokenId, uint256 index)
    public view returns (string memory) {
            require(index < metadataHistory[tokenId].length, "Indice
    inserito invalido.");
64          return (metadataHistory[tokenId][index].metadata);
        }
66

        // Funzione: Ritorna gli ultimi metadata di un token, dato il
    tokenId
68      function getMetadata(uint256 tokenId) public view returns (string
         memory) {
            return metadataToken[tokenId];
70      }

72      // Funzione: Ritorna il timestamp dell'ultima modifica o
    creazione (se non sono ancora state fatte modifiche)
        function getLastMetadataTimestamp(uint256 tokenId) public view
    returns (uint256) {
74          return metadataHistory[tokenId][metadataHistory[tokenId].
    length - 1].timestamp;
        }
76

        // Funzione: Ritorna l'address dell'ultima modifica o creazione (
    se non sono ancora state fatte modifiche)
78      function getLastMetadataChanger(uint256 tokenId) public view
    returns (address) {
            return metadataHistory[tokenId][metadataHistory[tokenId].
    length - 1].changer;
80      }
```

```
82      // Funzione: Mint di un token ad un address
        function mintTokenDoct(address _address, string memory _metadata)
        external onlyDAOTokenHolder {
84          require(balanceOf(_address) != 1, "Non e' possibile possedere
        piu' di un token.");
            uint256 tokenId = tokenIdCounter;
86          _mint(_address, tokenId);
            tokenIdCounter = tokenIdCounter + 1;
88
            metadataToken[tokenId] = _metadata;
90          metadataHistory[tokenId].push(MetadataChange(_metadata, block
        .timestamp, msg.sender));
        }
92
        // Funzione: Rimuovere token da un address
94      function removeTokenFromAddress(address _address, uint256 tokenId
        ) external onlyDAOTokenHolder {
            require(balanceOf(_address) == 1, "Inserire un address
        corretto.");
96          _transfer(_address, address(this), tokenId); // Trasferisci
        il token al contratto
            metadataHistory[tokenId].push(MetadataChange("Rimosso", block
        .timestamp, msg.sender));
98          metadataToken[tokenId] = "Rimosso";
        }
100
        // Funzione: Trasferire il controllo del contratto
102     function transferOwnership(address newOwner) public onlyOwner {
            require(newOwner != address(0), "Inserire un address corretto
        .");
104         owner = newOwner;
        }
106
}
```

I token degli operatori sanitari possono essere creati e modificati esclusivamente dagli amministratori della DAO, quindi dalle strutture dove prestano servizio.
Le informazioni di ciascun operatore sono salvate in strutture dati on-chain e

off-chain. Salvare tutte le informazioni sulla catena risulterebbe gravoso dal punto di vista della dimensione delle transazioni e dei blocchi, pertanto la maggior parte sarà inserita in modalità simile a IPFS e in blockchain verrà inserita soltanto la hash di tali dati.
Nel contratto sono presenti molte funzioni per gestire e visualizzare i metadata dei token degli operatori; in particolare è data molta importanza al tracciamento delle modifiche che vengono apportare da parte degli amministratori della DAO.

I medici, a loro volta, avranno il compito di creare la cartella clinica alla prima visita del paziente e la possibilità di modificarla nelle visite successive.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.21;

// Import: Importare standard token ERC721
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
// Import: Importare il contratto dei token della DAO
import "./SmartDAOtoken.sol";
// Import: Importare il contratto dei token degli operatori sanitari
import "./SmartDOCTtoken.sol";

contract SmartPATIENTtoken is ERC721 {

    address public owner; // Owner del contratto, definito
    inizialmente dal costruttore
    SmartDOCTtoken public tokenDOCTContract; // Variabile contratto
    per l'uso delle funzioni interne
    SmartDAOtoken public tokenDAOContract; // Variabile contratto per
     l'uso delle funzioni interne
    uint256 private tokenIdCounter; // Contatore dei token mintati

    // Mapping: Lega ID univoco dell'NFT con i propri metadata
    mapping(uint256 => string) public metadataToken;

    // Struct: Tiene traccia delle modifiche dei metadata dei token
    struct MetadataChange {
        string metadata;
```

```
24          uint256 timestamp;
            address changer;
26      }
        // Mapping: Lega ID univoco dell'NFT alla struttura
28      mapping(uint256 => MetadataChange[]) public metadataHistory;

30      // Modificatore: per definire l'identificatore onlyOwner
        modifier onlyOwner() {
32          require(msg.sender == owner, "Solo l'owner puo' chiamare
    questa funzione.");
            _;
34      }

36      // Modificatore: per definire i possessori dei token degli
    operatori sanitari
        modifier onlyDOCTTokenHolder() {
38          require(tokenDOCTContract.balanceOf(msg.sender) == 1, "Solo i
     possessori di token di opereatori sanitari possono chiamare
    questa funzione.");
            // "balanceOf" e' una funzione presente nello standard ERC721
    , non scritta esplicitamente in SmartDOCTtoken
40          // "== 1" in quanto, nel nostro caso, ogni operatore avra' un
     solo ed unico token
            _;
42      }

44      // Modificatore: per definire i possessori dei token della DAO
        modifier onlyDAOTokenHolder() {
46          require(tokenDAOContract.balanceOf(msg.sender, 1) == 1 ||
                    tokenDAOContract.balanceOf(msg.sender, 2) == 1 ||
48                  tokenDAOContract.balanceOf(msg.sender, 3) == 1,
                    "Solo i possessori dei token della DAO possono
    chiamare questa funzione.");
50          // "balanceOf" e' una funzione presente nello standard
    ERC1155, non scritta esplicitamente in SmartDAOtoken
            // "== 1" in quanto, nel nostro caso, ogni partecipante avra'
     un solo ed unico token
52          _;
        }
54
```

```
    // Costruttore: Inizializza il contratto
56  constructor(address _tokenDAOContract, address _tokenDOCTContract
    ) ERC721("","") {
        tokenDAOContract = SmartDAOtoken(_tokenDAOContract);
58      tokenDOCTContract = SmartDOCTtoken(_tokenDOCTContract);
        owner = msg.sender;
60      tokenIdCounter = 1;
    }
62
    // Funzione: Modificare i metadata del token tokenId con i nuovi
    metadata
64  // Tiene traccia anche di quando e' stata fatta la modifica e da
    chi (address), inoltre in questo modo abbiamo uno storico dei
    Metadata di un token
    function updateMetadata(uint256 tokenId, string memory
    newMetadata) external onlyDOCTTokenHolder {
66      require(keccak256(abi.encodePacked(getMetadata(tokenId))) !=
    keccak256(abi.encodePacked("Rimosso")), "Impossibile modificare i
    metadata di un token rimosso.");
        address changer = msg.sender;
68      metadataHistory[tokenId].push(MetadataChange(newMetadata,
    block.timestamp, changer));
        metadataToken[tokenId] = newMetadata;
70  }

72  // Funzione: Ritorna i metadata precedenti di un nft (la versione
     index)
    // Per esempio se questo nft ha avuto 3 modifiche dei metadata,
    metadataHistory contiene 3 elementi e con index=1 ci dara' la
    versione 2 dei metadata
74  function getPreviousMetadata(uint256 tokenId, uint256 index)
    public view returns (string memory) {
        require(index < metadataHistory[tokenId].length, "Indice
    inserito invalido.");
76      return (metadataHistory[tokenId][index].metadata);
    }
78
    // Funzione: Ritorna gli ultimi metadata di un token, dato il
    tokenId
```

```
80      function getMetadata(uint256 tokenId) public view returns (string
        memory) {
            return metadataToken[tokenId];
82      }

84      // Funzione: Ritorna l'address dell'ultima modifica o creazione (
        se non sono ancora state fatte modifiche)
        function getLastMetadataTimestamp(uint256 tokenId) public view
        returns (uint256) {
86          return metadataHistory[tokenId][metadataHistory[tokenId].
        length - 1].timestamp;
        }
88
        // Funzione: Ritorna l'address dell'ultima modifica o creazione (
        se non sono ancora state fatte modifiche)
90      function getLastMetadataChanger(uint256 tokenId) public view
        returns (address) {
            return metadataHistory[tokenId][metadataHistory[tokenId].
        length - 1].changer;
92      }

94      // Funzione: Mint di un token ad un address
        function mintTokenPatient(address _address, string memory
        _metadata) external onlyDOCTTokenHolder {
96          require(balanceOf(_address) != 1, "Non e' possibile possedere
         piu' di un token.");
            uint256 tokenId = tokenIdCounter;
98          _mint(_address, tokenId);
            tokenIdCounter = tokenIdCounter + 1;
100
            metadataToken[tokenId] = _metadata;
102         metadataHistory[tokenId].push(MetadataChange(_metadata, block
        .timestamp, msg.sender));
        }
104
        // Funzione: Rimuovere token da un address
106     function removeTokenFromAddress(address _address, uint256 tokenId
        ) external onlyDAOTokenHolder {
            require(balanceOf(_address) == 1, "Inserire un address
        corretto.");
```

```
108         _transfer(_address, address(this), tokenId); // Trasferisci
    il token al contratto
            metadataHistory[tokenId].push(MetadataChange("Rimosso", block
    .timestamp, msg.sender));
110         metadataToken[tokenId] = "Rimosso";
        }
112
        // Funzione: Trasferire il controllo del contratto
114     function transferOwnership(address newOwner) public onlyOwner {
            require(newOwner != address(0), "Inserire un address corretto
    .");
116         owner = newOwner;
        }
118
}
```

In questo contratto si è analizzato un esempio di funzioni per la creazione dei token per le cartelle cliniche dei pazienti. Essi possono essere creati esclusivamente da operatori sanitari che possiedono il token creato dagli amministratori della DAO. Tutte le informazioni dei pazienti sono salvate in strutture dati on-chain e off-chain. Come per i dati dei medici, salvare tutte le informazioni sulla catena risulterebbe gravoso dal punto di vista della dimensionalità, quindi la maggior parte è salvata off-chain.

Nel contratto, oltre alle funzioni di gestione e creazione dei token, ne sono presenti altre per tracciare le modifiche ai metadata dei pazienti, che, successivamente, saranno gestite anche esternamente, tramite un contratto che controlla la governance tra paziente e medico.

7.4.2 Registrazione e autenticazione

I medici e ogni operatore sanitario abilitato ad effettuare visite e compilare referti saranno registrati dall'ospedale di competenza, quindi dagli amministratori della DAO, tramite i contratti definiti precedentemente.
I pazienti, invece, dopo aver ottenuto il proprio indirizzo sulla blockchain nazionale, avranno la possibilità di ricevere il token relativo alla propria cartella clinica durante la prima visita presso un medico abilitato.

Prima della visita stessa, ogni paziente deve procedere ad una fase di registrazione attraverso un'applicazione. Dovranno essere inseriti alcuni dati personali: nome, cognome, data di nascita, luogo di nascita, residenza, email, telefono e una password per i futuri accessi.

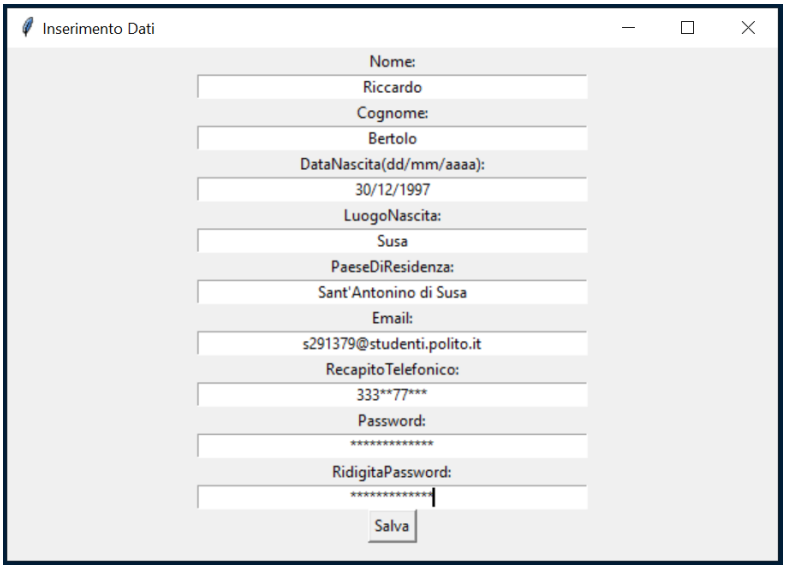

Figura 7.1: Inserimento dati da parte del paziente.

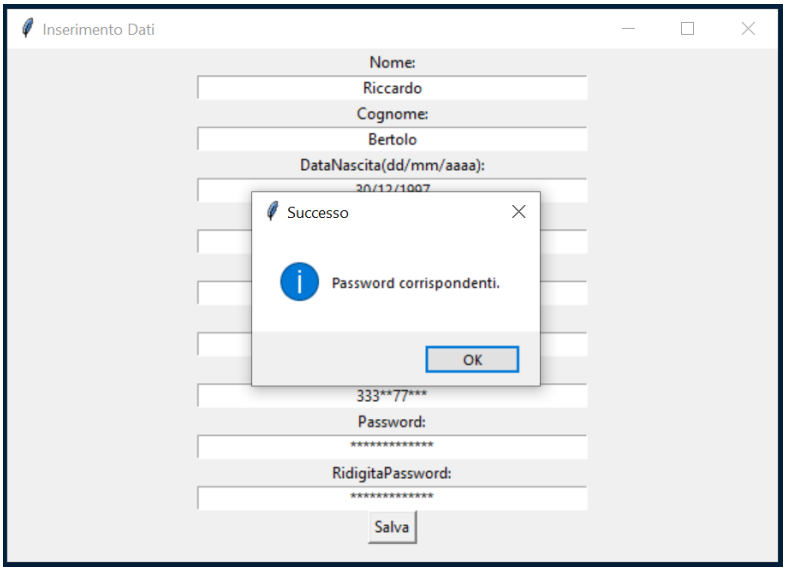

Figura 7.2: Conferma password inserite corrispondenti.

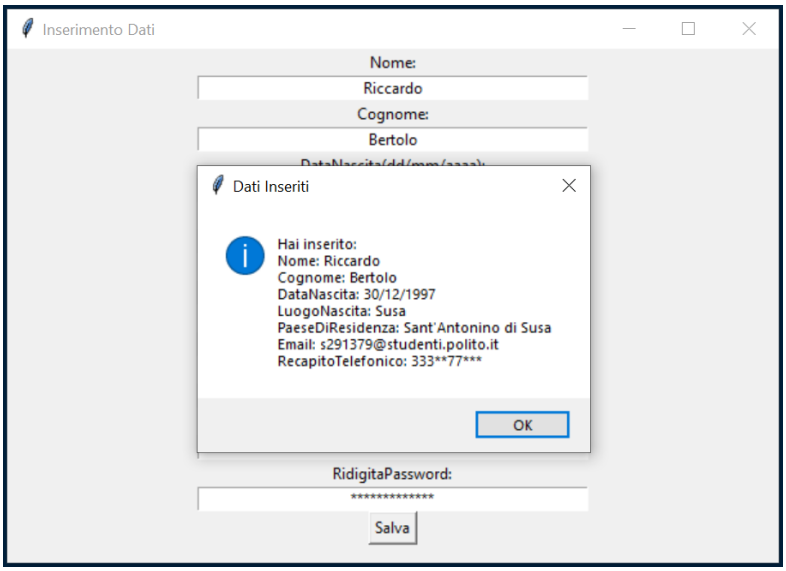

Figura 7.3: Riepilogo dati inseriti.

Questa prima registrazione non sarà ancora valida e quindi salvata esclusivamente sul dispositivo dell'utente, finché non verrà verificata e attivata da un operatore sanitario. I dati forniti prima dell'attivazione saranno completamente modificabili. Questo processo di attivazione evita anche la creazione di molteplici token da parte

di malintezionati, che andrebbero ad intasare la rete.
Per attivare la registrazione, durante la prima visita, basterà che il medico scansioni il codice QR che compare dopo che l'utente ha inserito i propri dati nell'applicazione.

Figura 7.4: Codice QR da mostrare al medico per l'attivazione.

Una volta scansionato il QR, il medico visualizzerà i dati inseriti dal paziente in formato JSON, formato usato solitamente per conservare i metadata dei token all'interno della blockchain di Ethereum.
Dopo aver verificato che i dati sono corretti, tramite un documento di identità, egli avrà la possibilità di mintare (creare e inserire in blockchain) sull'address del paziente, l'NFT relativo alla cartella clinica, tramite la funzione *mintTokenPatient*, presente all'interno del contratto definito precedentemente.
Inizialmente il token avrà metadata che conterranno esclusivamente i dati inseriti dal paziente in fase di registrazione, ma successivamente si potranno inserire tutti i dati sanitari.
Inoltre il sistema, prima di permettere al medico di mintare l'NFT per il paziente, verificherà che non ne sia già stato creato uno, in precedenza, con gli stessi dati personali (nome, cognome, data di nascita e luogo di nascita).

Dopo aver scannerizzato il QR i dati presenti saranno visualizzati nel seguente formato.

```
{"title":"NFT Sanitario",
    "description":"Descrizione NFT generica",
    "year":"2023",
    "category":"Sanita' pubblica",
    "attributes":[
    {"type":"Name",
        "value":"Riccardo"},
    {"type":"Surname",
        "value":"Bertolo"},
    {"type":"DataNascita",
        "value":"30/12/1997"},
    {"type":"LuogoNascita",
        "value":"Susa"},
    {"type":"Residenza",
        "value":"Sant'Antonino di Susa"},
    {"type":"Email",
        "value":"s291379@studenti.polito.it"},
    {"type":"Telefono",
        "value":"333**77***"}]}
```

L'applicazione alla base di questi passaggi consente all'utente la gestione esclusivamente dei token del servizio sanitario nazionale. Se il paziente, all'interno del proprio indirizzo, possiede altri token di enti diversi che si appoggiano alla blockchain nazionale, essi non saranno visualizzati dalla piattaforma.

In breve, una volta installata l'applicazione, si dovranno compilare i campi definiti precedentemente impostando una nuova password per quel dispositivo.
Successivamente si dovrà importare il proprio wallet relativo alla blockchain nazionale, inserendo la chiave privata corrispondente. Se il sistema, a questo indirizzo, rileva l'assenza del token relativo alla cartella clinica del servizio sanitario nazionale, presenta il codice QR definito in precedenza, che il paziente dovrà mostrare alla

prima visita per ottenere l'NFT.
In caso l'address contenga già il token relativo alla cartella clinica, si potrà procedere alle sezioni di gestione del token, avendo già ottenuto la proprietà di esso.

Il processo di creazione della chiave privata e successiva generazione della chiave pubblica e indirizzo, tramite algoritmo ECC, per ogni utente sulla blockchain nazionale, sarà gestito direttamente dallo Stato ed esula dal presente studio, ma, come anticipato, potrebbe essere eseguito tramite SPID.

Le interfacce precedenti e la generazione del codice QR del paziente dopo la registrazione sono state create usando il linguaggio Python con il seguente codice.

```
import qrcode
import tkinter as tk
from PIL import Image, ImageTk
from tkinter import messagebox
import json

def inserimento_dati_genera_qr():

    # Lettura delle password inserite
    password1 = password_entry.get()
    password2 = retype_password_entry.get()

    # Se le password inserite corrispondono
    if password1 == password2:
        messagebox.showinfo("Successo", "Password corrispondenti.")

        # Altro campi inseriti
        nome = nome_entry.get()
        cognome = cognome_entry.get()
        dataNascita = dataNascita_entry.get()
        luogoNascita = luogoNascita_entry.get()
        residenza = residenza_entry.get()
        email = email_entry.get()
        telefono = telefono_entry.get()

        # Dati Inseriti
```

```
        dati_completi = f'Hai inserito:\nNome: {nome}\nCognome: {
    cognome}\nDataNascita: {dataNascita}\nLuogoNascita: {luogoNascita
    }\nPaeseDiResidenza: {residenza}\nEmail: {email}\
    nRecapitoTelefonico: {telefono}'
28      dati_qr = "{\"title\":\"NFT Sanitario\",\n\t\"description
    \":\"Descrizione NFT generica\",\n\t\"year\":\"2023\",\n\t\"
    category\":\"Sanita' pubblica\",\n\t\"attributes\":[\n\t{\"type
    \":\"Name\",\n\t\t\"value\":\""+nome+"\"},\n\t{\"type\":\"Surname
    \",\n\t\t\"value\":\""+cognome+"\"},\n\t{\"type\":\"DataNascita
    \",\n\t\t\"value\":\""+dataNascita+"\"},\n\t{\"type\":\"
    LuogoNascita\",\n\t\t\"value\":\""+luogoNascita+"\"},\n\t{\"type
    \":\"Residenza\",\n\t\t\"value\":\""+residenza+"\"},\n\t{\"type
    \":\"Email\",\n\t\t\"value\":\""+email+"\"},\n\t{\"type\":\"
    Telefono\",\n\t\t\"value\":\""+telefono+"\"}]}"

30      # Riepilogo dati inseriti
        messagebox.showinfo("Dati Inseriti", dati_completi)
32
        # Oggetto QRCode
34      qr = qrcode.QRCode(
            version=1,
36          error_correction=qrcode.constants.ERROR_CORRECT_L,
            box_size=10,
38          border=4,
        )
40
        # Dati aggiunti al QR code
42      qr.add_data(dati_qr)
        qr.make(fit=True)
44
        # Genera l'immagine QR
46      img = qr.make_image(fill_color="black", back_color="white")

48      # Salva il codice QR
        img.save("qr_patient.png")
50
        # Mostra il codice QR
52      img.show()
    else:
54      messagebox.showerror("Errore", "Password non corrispondono.")
```

```
56
# Finestra principale
58 root = tk.Tk()
root.title("Inserimento Dati")
60
# Dimensioni della finestra
62 root.geometry("600x400")

64 # Campi di inserimento
nome = tk.Label(root, text="Nome:")
66 nome.pack()
nome_entry = tk.Entry(root, width=50, justify="center")
68 nome_entry.pack()

70 cognome = tk.Label(root, text="Cognome:")
cognome.pack()
72 cognome_entry = tk.Entry(root, width=50, justify="center")
cognome_entry.pack()
74
dataNascita = tk.Label(root, text="DataNascita(dd/mm/aaaa):")
76 dataNascita.pack()
dataNascita_entry = tk.Entry(root, width=50, justify="center")
78 dataNascita_entry.pack()

80 luogoNascita = tk.Label(root, text="LuogoNascita:")
luogoNascita.pack()
82 luogoNascita_entry = tk.Entry(root, width=50, justify="center")
luogoNascita_entry.pack()
84
residenza = tk.Label(root, text="PaeseDiResidenza:")
86 residenza.pack()
residenza_entry = tk.Entry(root, width=50, justify="center")
88 residenza_entry.pack()

90 email = tk.Label(root, text="Email:")
email.pack()
92 email_entry = tk.Entry(root, width=50, justify="center")
email_entry.pack()
94
```

```
telefono = tk.Label(root, text="RecapitoTelefonico:")
telefono.pack()
telefono_entry = tk.Entry(root, width=50, justify="center")
telefono_entry.pack()

password = tk.Label(root, text="Password:")
password.pack()
password_entry = tk.Entry(root, width=50, justify="center", show="*")
password_entry.pack()

retype_password = tk.Label(root, text="RidigitaPassword:")
retype_password.pack()
retype_password_entry = tk.Entry(root, width=50, justify="center",
    show="*")
retype_password_entry.pack()

# Pulsante "Salva" per procedere con i dati inseriti
salva_button = tk.Button(root, text="Salva", command=
    inserimento_dati_genera_qr)
salva_button.pack()

# Visualizzare il risultato
risultato_label = tk.Label(root, text="")
risultato_label.pack()

# Ciclo principale interfaccia grafica
root.mainloop()
```

7.4.3 Funzioni pazienti e medici

Ogni paziente, una volta ottenuto il token della propria cartella clinica all'interno del proprio address, potrà visualizzarlo tramite l'applicazione che ha usato per svolgere la registrazione precedente alla prima visita. All'interno della piattaforma ci saranno varie sezioni di controllo, che egli potrà gestire autonomamente.

Una delle sezioni presenti include un elenco di address ai quali si è data *l'autorizzazione di visualizzare* la propria cartella, che comprende una sottosezione di indirizzi che hanno anche *l'autorizzazione di apportarne delle modifiche*.
In entrambi i casi si può decidere di consentire l'autorizzazione per l'intera cartella clinica o esclusivamente per determinate sezioni di interesse. Per esempio, se si sta facendo una visita specialistica, da medico specialista in pneumologia, cioè nella cura delle malattie dell'apparato respiratorio, si può optare per la la visualizzazione e modifica solo di determinate sezioni, rispetto all'intera cartella.
Ovviamente, appena si riceve il proprio token e si accede per la prima volta alla piattaforma, questa sezione sarà vuota, ma una volta che il paziente si recherà ad una visita, potrà concedere al medico presente l'autorizzazione di visualizzare e/o modificare, inserendo il suo address all'interno dell'elenco. Per esempio, questo processo avverrà già durante la prima visita, quando il medico minterà il token al paziente, in quanto, dopo la creazione, il paziente stesso dovrà concedere al medico l'autorizzazione di concludere l'inserimento dei dati della propria cartella clinica.

Ogni volta che un paziente si reca ad una visita, quindi, consentirà al medico di visualizzare e/o modificare il token, se non è già presente nell'elenco di indirizzi autorizzati. Qualsiasi modifica apportata dal professionista sanitario al token della cartella verrà registrata all'interno della blockchain come una transazione, caratterizzata da un timestamp e dall'address del medico che ha fatto le modifiche, in modo tale che non sia ripudiabile in futuro.
Dopo aver completato la visita, il paziente può revocare al medico la possibilità di visualizzare e modificare la propria cartella clinica, direttamente dalla piattaforma, e poi riconcedere tale autorizzazione alla visita successiva; in alternativa, potrebbe anche decidere di non revocarla e lasciargliela. È importante notare che la non revoca non è un problema, in quanto eventuali modifiche non autorizzate sarebbero comunque legate da timestamp e address del medico, così da poterne tracciare la modifica.

Quando il paziente dovrà inserire l'address del medico nell'elenco di indirizzi che hanno l'autorizzazione sul proprio token, dovrà esclusivamente scansionare il codice QR del sanitario, che mostrerà i dati e offrirà la possibilità di decidere se

concedere l'autorizzazione alla sola visualizzazione o anche alla modifica di tutte le sezioni o esclusivamente di quelle di competenza specialistica dell'operatore. Dopo la scelta, in automatico, l'address del medico sarà aggiunto all'elenco indirizzi autorizzati.
Un altro possibile metodo per autorizzare il medico è quello di procedere tramite richiesta. Il paziente dovrà aspettare che egli faccia richiesta, tramite il suo portale, con la possibilità di richiedere una visione e/o modifica dei record in base al fine della visita, senza dover necessariamente richiedere la condivisione di tutte le informazioni. A questo punto il paziente dovrà esclusivamete accettare tale richiesta, che in automatico farà aggiungere l'address del medico nell'elenco indirizzi autorizzati.

Il paziente, quindi, oltre ad avere la possibilità di visualizzare il suo token con tutte le specifiche salvate nei vari metadata, può attivare altre funzioni all'interno del contratto che regola la governance dei token dei pazienti. Può dare e togliere il permesso di visualizzare e modificare il proprio token a operatori sanitari che siano in possesso di determinati NFT, ma non può modificarlo di sua spontanea volontà. Se volesse modificare, per esempio, alcune delle informazioni personali che ha inserito durante la registrazione, dovrebbe rivolgersi ad un operatore nel corso di una visita. I dati di accesso alla piattaforma, come per esempio la password, invece, sono modificabili, in quanto non sono contenuti nel token della cartella clinica.

Un operatore sanitario, pertanto, oltre alla possibilità di creare e visualizzare i token dei pazienti, dopo aver ottenuto l'autorizzazione, ha la possibilità di scrivere e quindi modificarne il contenuto.
Inoltre, gli operatori possono attivare una funzione di accesso di emergenza, se il paziente è impossibilitato e non può dare l'autorizzazione di visualizzazione e modifica della propria cartella. Ovviamente questa azione sarebbe tracciata con timestamp e address del'operatore che ha attivato la funzione, per evitare che medici la attivino senza esserci la necessità.
Ogni paziente può, inoltre, inserire come indirizzo di recupero un address di un familiare o amico, così in caso di problemi, se venisse attivata la funzione di emergenza da parte di un medico, arriverebbe direttamente una notifica all'indirizzo di recupero inserito dal paziente stesso.

Ogni cambiamento e ogni azione che un address consente è tracciato come transazione all'interno della blockchain, quindi anche se non si revoca la possibilità di modificare il proprio token ad un medico, se egli lo modifica senza il consenso, in ogni caso sarà tracciabile con timestamp di quando è stato modificato e dall'indirizzo di chi ha apportato la modifica. Questo è anche utile per la funzione di emergenza, in quanto, se un medico la attiva senza che effettivamente ci sia un'emergenza, sarebbe tutto segnato all'interno di una modifica di stato in blockchain.

A seguito dell'aggiornamento della sua cartella clinica, il paziente è avvisato. Questo quadro garantisce la privacy, impedendo alle parti interessate l'accesso alle cartelle cliniche senza il permesso del paziente, che, dopo le modifiche effettuate al token, potrà vedere l'NFT aggiornato tramite l'applicazione.

Di seguito si riporta un esempio di contratto per le funzioni definite tra pazienti e operatori sanitari.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.21;

// Import: Importare il contratto dei token degli operatori sanitari
import "./SmartDOCTtoken.sol";
// Import: Importare il contratto dei token dei pazienti
import "./SmartPATIENTtoken.sol";

contract GovernancePatient_Doctor {

    address public owner; // Owner del contratto, definito
    inizialmente dal costruttore
    SmartDOCTtoken public tokenDOCTContract; // Variabile contratto
    per l'uso delle funzioni interne
    SmartPATIENTtoken public tokenPATIENTContract; // Variabile
    contratto per l'uso delle funzioni interne

    // Modificatore: per definire l'identificatore onlyOwner
    modifier onlyOwner() {
```

```
        require(msg.sender == owner, "Solo l'owner puo' chiamare
    questa funzione.");
        _;
    }

    // Mapping: Lega id univoco dell'NFT del paziente, con gli
    address dei dottori autorizzati a visualizzare il token
    mapping(uint256 => address[]) private authorizedViewAddress;
    // Mapping: Lega id univoco dell'NFT del paziente, con gli
    address dei dottori autorizzati a modificare il token
    mapping(uint256 => address[]) private authorizedEditAddress;

    // Evento: Token modificato
    event TokenModified(uint256 indexed tokenId, address indexed
    _modifier);
    // Evento: Token visualizzato tramite funzione di emergenza
    event TokenViewEmergence(uint256 indexed tokenId, address indexed
     _address);
    // Evento: Token modificato tramite funzione di emergenza
    event TokenModifiedEmergence(uint256 indexed tokenId, address
    indexed _modifier);

    // Costruttore: Inizializza il contratto
    constructor(address _tokenDOCTContract, address
    _tokenPATIENTContract) {
        tokenDOCTContract = SmartDOCTtoken(_tokenDOCTContract);
        tokenPATIENTContract = SmartPATIENTtoken(
    _tokenPATIENTContract);
        owner = msg.sender;
    }

    // Funzione: Aggiungere un address di un dottore autorizzato a
    visualizzare i metadata del token del paziente
    function addAuthorizedViewAddress(uint256 tokenId, address
    _address) external {
        require(tokenPATIENTContract.ownerOf(tokenId) == msg.sender,
    "Solo il proprietario del token puo' aggiungere address.");
        require(tokenDOCTContract.balanceOf(_address) == 1, "Solo i
    possessori di token di operatori sanitari possono essere aggiunti.
    ");
```

```
44          require(!verifyViewAuthorization(tokenId, _address), "
    Operatore sanitario gia' autorizzato a visualizzare.");
            authorizedViewAddress[tokenId].push(_address);
46      }

48      // Funzione: Aggiungere un address di un dottore autorizzato a
    modificare i metadata del token del paziente
        function addAuthorizedEditAddress(uint256 tokenId, address
    _address) external {
50          require(tokenPATIENTContract.ownerOf(tokenId) == msg.sender,
    "Solo il proprietario del token puo' aggiungere address.");
            require(tokenDOCTContract.balanceOf(_address) == 1, "Solo i
    possessori di token di operatori sanitari possono essere aggiunti.
    ");
52          require(!verifyEditAuthorization(tokenId, _address), "
    Operatore sanitario gia' autorizzato a modificare.");
            authorizedEditAddress[tokenId].push(_address);
54      }

56      // Funzione: Rimuovere un address di un dottore autorizzato a
    visualizzare i metadata del token del paziente
        function removeAuthorizedViewAddress(uint256 tokenId, address
    _address) external {
58          require(tokenPATIENTContract.ownerOf(tokenId) == msg.sender,
    "Solo il proprietario del token puo' rimuovere address.");
            require(tokenDOCTContract.balanceOf(_address) == 1, "Solo i
    possessori di token di operatori sanitari possono essere rimossi."
    );
60          require(verifyViewAuthorization(tokenId, _address), "
    Impossibile rimuovere l'autorizzazione ad un operatore non
    autorizzato.");
            address[] storage allAddress = authorizedViewAddress[tokenId
    ];
62          for (uint256 i = 0; i < allAddress.length; i++) {
                if (allAddress[i] == _address) {
64                  allAddress[i] = allAddress[allAddress.length - 1];
                    allAddress.pop();
66                  break;
                }
68          }
```

```
    }

    // Funzione: Rimuovere un address di un dottore autorizzato a modificare i metadata del token del paziente
    function removeAuthorizedEditAddress(uint256 tokenId, address _address) external {
        require(tokenPATIENTContract.ownerOf(tokenId) == msg.sender, "Solo il proprietario del token puo' rimuovere address.");
        require(tokenDOCTContract.balanceOf(_address) == 1, "Solo i possessori di token di operatori sanitari possono essere rimossi.");
        require(verifyEditAuthorization(tokenId, _address), "Impossibile rimuovere l'autorizzazione ad un operatore non autorizzato.");
        address[] storage allAddress = authorizedEditAddress[tokenId];
        for (uint256 i = 0; i < allAddress.length; i++) {
            if (allAddress[i] == _address) {
                allAddress[i] = allAddress[allAddress.length - 1];
                allAddress.pop();
                break;
            }
        }
    }

    // Funzione: Visualizzare metadata di un token, essendone il paziente proprietario
    function getAllTokenMetadata(uint256 tokenId) external view returns (string memory) {
        require(tokenPATIENTContract.ownerOf(tokenId) == msg.sender, "Solo il proprietario del token puo' vederne le specifiche.");
        return (tokenPATIENTContract.getMetadata(tokenId));
    }

    // Funzione: Visualizzare metadata di un token, essendo stato autorizzato
    function viewTokenMetadataFromDoctor(uint256 tokenId, address _address) external view returns (string memory) {
        require(verifyViewAuthorization(tokenId, _address), "Address non autorizzato a visualizzare il token.");
```

```
        return (tokenPATIENTContract.getMetadata(tokenId));
    }

    // Funzione: Modifica metadata di un token, essendo stato autorizzato
    function editTokenMetadataFromDoctor(uint256 tokenId, address _address, string memory newMetadata) external {
        require(verifyEditAuthorization(tokenId, _address), "Address non autorizzato a modificare il token.");
        tokenPATIENTContract.updateMetadata(tokenId, newMetadata);
        emit TokenModified(tokenId, _address); // Evento di modifica token
    }

    // Funzione: Visualizzare metadata di un token, tramite funzione di emergenza
    function viewTokenEmergence(uint256 tokenId, address _address) external returns (string memory) {
        emit TokenViewEmergence(tokenId, _address); // Evento di visualizzazione di emergenza
        return (tokenPATIENTContract.getMetadata(tokenId));
    }

    // Funzione: Modifica metadata di un token, tramite funzione di emergenza
    function editTokenEmergence(uint256 tokenId, address _address, string memory newMetadata) external {
        tokenPATIENTContract.updateMetadata(tokenId, newMetadata);
        emit TokenModifiedEmergence(tokenId, _address); // Evento di modifica di emergenza
    }

    // Funzione: Controllo dell'autorizzazione a visualizzare
    function verifyViewAuthorization(uint256 tokenId, address _address) public view returns (bool) {
        address[] memory authorizedView = authorizedViewAddress[tokenId];
        for (uint256 i = 0; i < authorizedView.length; i++) {
            if (authorizedView[i] == _address) {
                return true;
```

```
            }
        }
        return false;
    }

    // Funzione: Controllo dell'autorizzazione a modificare
    function verifyEditAuthorization(uint256 tokenId, address
    _address) public view returns (bool) {
        address[] memory authorizedEdit = authorizedEditAddress[
    tokenId];
        for (uint256 i = 0; i < authorizedEdit.length; i++) {
            if (authorizedEdit[i] == _address) {
                return true;
            }
        }
        return false;
    }

    // Funzione: Trasferire il controllo del contratto
    function transferOwnership(address newOwner) public onlyOwner {
        require(newOwner != address(0), "Inserire un address corretto
    .");
        owner = newOwner;
    }

}
```

Capitolo 8

Conclusione

Il lavoro di tesi ha voluto esplorare le potenzialità della tecnologia Blockchain e mettere in luce le possibili applicazioni in ambito sanitario.
Attraverso l'analisi dell'architettura e il funzionamento delle principali blockchain esistenti e della crittografia che ne sta alla base, è stato realizzato uno studio che possa costituire fondamento per l'ottimizzazione della presente situazione sanitaria, non solo nazionale, ma anche europea o mondiale.

All'interno della tesi si sono volute indagare le possibili procedure per la definizione di una blockchain nazionale e realizzare un modello tipo che potrà essere utilizzato in futuro da enti e aziende diverse, come strumento di miglioramento tecnologico.

Sono stati definiti, tramite smart contracts, la DAO per l'amministrazione del servizio sanitario nazionale, i token per gli operatori sanitari e per i pazienti e gli standard e le regole per la visualizzazione e modifica degli stessi.

Per la definizione dei contratti, dal momento che lo Stato italiano non dispone attualmente di una blockchain nazionale, si è optato di beneficiare della rete esistente Ethereum e utilizzare il linguaggio di programmazione Solidity. I contratti realizzati potranno comunque costituire la base per una futura implementazione su qualsiasi sistema.

Ogni paziente avrà la piena proprietà della propria cartella clinica in formato di non-fungible token, all'interno del quale verrà registrata la completa documentazione personale sanitaria. In questo modo, si migliora la qualità della vita dei pazienti, facilitando l'uso futuro delle informazioni in qualsiasi struttura sanitaria.

La tecnologia Blockchain costituisce una possibile soluzione di archiviazione, garantendo elevati livelli di sicurezza e integrità dei dati. Questo è possibile grazie alle sue proprietà di immutabilità (i dati archiviati al suo interno sono inalterabili) e alla capacità di verificare informazioni relative a versioni passate.
Inoltre, attraverso questa metodologia di memorizzazione, la ricerca medica potrà usufruire, nel futuro e con le dovute autorizzazioni, della totalità dei fascicoli sanitari dei pazienti e utilizzarli per lo studio di nuove cure, sulla base di completi report storici.

Alcuni Stati hanno già attuato soluzioni impiegando questa tecnologia per migliorare la sicurezza e l'affidabilità. Per esempio, il Brasile, nel settembre 2023, ha iniziato ad implementare, in alcune città, una versione basata su Blockchain del registro condiviso del servizio nazionale delle entrate, per l'emissione di una nuova tipologia di carta d'identità, al fine di ridurre le frodi e le attività illegali (19).
Esistono aziende che stanno già attuando delle soluzioni a livello sanitario, per esempio Medicalchain (1), con sede a Londra, che permette al paziente di avere il controllo della condivisione dei propri dati sanitari.

Grazie alle opportunità offerte da questa tecnologia, con l'avanzare degli anni, si prevede che la sua diffusione continuerà a crescere in diversi settori.

I contratti presentati potranno esclusivamente fornire una base per una futura implementazione, non essendo possibile entrare nel dettaglio di decisioni che dovranno essere prese dagli enti sanitari.
Questa tesi, quindi, presenta una prima versione del progetto, alla quale dovranno essere apportate modifiche e migliorie legate principalmente ai dettagli richiesti dal sistema statale che ne farà uso.

Bibliografia

[1] Medicalchain. URL https://medicalchain.com/en/.

[2] Ethereum improvement proposals, . URL https://eips.ethereum.org/erc.

[3] Etherscan, . URL https://etherscan.io/.

[4] Hyperledger, . URL https://www.hyperledger.org/.

[5] Keccakteam, . URL https://keccak.team/.

[6] Openzeppelin, . URL https://www.openzeppelin.com/.

[7] Crossroads, . URL https://nftartwork.co.uk/nft-artwork/crossroads/.

[8] Coinmarketcap, . URL https://coinmarketcap.com/currencies/.

[9] Remix, . URL https://remix.ethereum.org/.

[10] Elliptic curves over finite fields, . URL https://graui.de/code/elliptic2/.

[11] Solidity. URL https://docs.soliditylang.org/en/v0.8.21/.

[12] Turintech. URL https://www.turintech.it/home.

[13] Ripemd-160, 2014. URL https://en.bitcoin.it/wiki/RIPEMD-160.

[14] Secp256k1, 2019. URL https://en.bitcoin.it/wiki/Secp256k1.

[15] Technical background of version 1 bitcoin addresses, 2021. URL https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses.

[16] Base58check, 2021. URL https://en.bitcoin.it/wiki/Base58Check_encoding.

[17] Ecdsa: Elliptic curve signatures, 2022. URL https://cryptobook.nakov.com/digital-signatures/ecdsa-sign-verify-messages.

[18] Nft dinamici, 2022. URL https://www.dz-techs.com/it/static-vs-dynamic-nfts-how-do-they-differ.

[19] Brazil develops blockchain network to support id rollout, 2023. URL https://www.forbes.com/sites/angelicamarideoliveira/2023/09/27/brazil-develops-blockchain-network-to-support-id-rollout/.

[20] Digital signature algorithm, 2023. URL https://en.wikipedia.org/wiki/Digital_Signature_Algorithm.

[21] Building a decentralized autonomous organization (dao), 2023. URL https://itnext.io/.

[22] Blockchain: cos'è, come funziona e come cambierà il business, 2023. URL https://www.zerounoweb.it/cio-innovation/blockchain-cose-come-utilizzarla-e-come-cambiera-il-business/.

[23] 7ecnologie. Evoluzioni: smart contract e nft. URL https://www.7ecnologie.it/15-blockchain-e-bitcoin/05-evoluzioni-smart-contract-e-nft.

[24] Giulia Adonopoulos. Smart contract: cosa sono e come funzionano, 2022. URL https://www.forbes.com/advisor/it/investire/criptovalute/smart-contract-cosa-sono-e-come-funzionano/.

[25] Oluwademilade Afolabi. Che cos'è una blockchain ibrida e in cosa differisce da una normale blockchain?, 2023. URL https://www.makeuseof.com/what-is-hybrid-blockchain-how-differ-from-regular-blockchain/.

[26] Andreas M. Antonopoulos. *MasteringBitcoin*. O'Reilly Media, first edition, 2017.

[27] Andreas M. Antonopoulos and Dr. Gavin Wood. *MasteringEthereum*. O'Reilly Media, first edition, 2018.

[28] Hany F. Atlam. Blockchain with internet of things: Benefits, challenges, and future directions. 2018. URL https://www.mecs-press.org/ijisa/ijisa-v10-n6/IJISA-V10-N6-5.pdf.

[29] Adam Back. Hashcash - a denial of service counter-measure. 1997. URL http://www.hashcash.org/papers/hashcash.pdf.

[30] Dave Bayer, Stuart Haber, and Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. 1992. URL https://www.math.columbia.edu/~bayer/papers/Timestamp_BHS93.pdf.

[31] Binance. Unspent transaction output (utxo). URL https://academy.binance.com/en/glossary/unspent-transaction-output-utxo.

[32] Bitpanda. Tutto quello che devi sapere sugli alberi di merkle. URL https://www.bitpanda.com/academy/it/lezioni/tutto-quello-che-devi-sapere-sugli-alberi-di-merkle/.

[33] Giovanni Capaccioli. Come nasce la blockchain, 2019. URL https://affidaty.io/blog/it/2019/04/come-nasce-la-blockchain/.

[34] Marco Cavicchioli. Blockchain, the differences between hard fork, soft fork and sidechain, 2022. URL https://en.cryptonomist.ch/2022/04/18/blockchain-differences-between-hard-fork-soft-fork-sidechain/.

[35] David Chaum. Blind signatures for untraceable payments. 1982. URL https://sceweb.sce.uhcl.edu/yang/teaching/csci5234WebSecurityFall2011/Chaum-blind-signatures.PDF.

[36] Wei Dai. B-money. 1998. URL http://www.weidai.com/bmoney.txt.

[37] Bazzanella Danilo and Gangemi Andrea. Materiale del corso blockchain e criptoeconomia.

[38] Bazzanella Danilo and Di Scala Antonio Jose'. Materiale del corso crittografia.

[39] Ethereum. Ethereum development documentation. URL https://ethereum.org/en/developers/docs/.

[40] Joep Crompvoets Evrim Tan, Stanislav Mahula. Blockchain governance in the public sector: A conceptual framework for public management. 2021. URL https://www.sciencedirect.com/science/article/pii/S0740624X21000617.

[41] Hal Finney. Reusable proofs of work, 2004. URL https://nakamotoinstitute.org/finney/rpow/index.html.

[42] Jake Frankenfield. Digicash: Meaning, history, implications, 2023. URL https://www.investopedia.com/terms/d/digicash.asp.

[43] GiorgioHerbie. Consensus mechanisms, 2023. URL https://ethereum.org/it/developers/docs/consensus-mechanisms/.

[44] Bharath H., Rahul N., Shylash S., and Vinny Pious. Patient data management using blockchain. 2020. URL https://www.researchgate.net/publication/343492725_Patient_Data_Management_Using_Blockchain.

[45] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. 1991. URL https://link.springer.com/article/10.1007/BF00196791.

[46] Eric Hughes. A cypherpunk's manifesto. 1993. URL https://nakamotoinstitute.org/static/docs/cypherpunk-manifesto.txt.

[47] Linas Kmieliauskas. Blockchain: cos'è e come funziona, 2021. URL https://it.cryptonews.com/guides/what-is-blockchain.htm.

[48] Vito Lavecchia. Definizione e differenza tra tecnologia dirompente e tecnologia di supporto, 2022. URL https://vitolavecchia.altervista.org/.

[49] Rebecca Lewis, John McPartland, and Rajeev Ranjan. Blockchain and financial market innovation. 2017. URL https://www.chicagofed.org/~/media/publications/economic-perspectives/2017/ep2017-7-pdf.pdf.

[50] Alex Lielacher. A history of bitcoin forks, 2018. URL https://www.bitcoinmarketjournal.com/bitcoin-forks/.

[51] Peter Lone, Kumar Nagarajan, Trish Supples, , and Paul Wong. Using distributed ledger technology for payment directories, 2022.

URL https://www.federalreserve.gov/econres/notes/feds-notes/using-distributed-ledger-technology-for-payment-directories-20220203.html.

[52] Crosby M., Nachiappan, Pattanayak P., Verma S., and Kalyanaraman V. Applied innovation review. 2016. URL https://scet.berkeley.edu/wp-content/uploads/AIR-2016-Blockchain.pdf.

[53] Fitzpatrick S. M. and Mckeon S. Banking on stone money: Ancient antecedents to bitcoin. 2019. URL https://www.researchgate.net/publication/333657766_Banking_on_Stone_Money_Ancient_Antecedents_to_Bitcoin.

[54] Damiano Di Francesco Maesa and Paolo Mori. Blockchain 3.0 applications survey. 2020. URL https://www.researchgate.net/publication/338406976_Blockchain_30_applications_survey.

[55] Vinodhini Mani, Prakash Manickam, Youseef Alotaibi, Saleh Alghamdi, and Osamah Ibrahim Khalaf. Hyperledger healthchain: Patient-centric ipfs-based storage of health records. 2021. URL https://www.researchgate.net/publication/356751386_Hyperledger_Healthchain_Patient-Centric_IPFS-Based_Storage_of_Health_Records.

[56] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. URL https://bitcoin.org/bitcoin.pdf.

[57] Figueiredo Novo and Isac Daniel. Property-based testing of erc-721 ethereum smart contracts. 2021. URL https://www.dcc.fc.up.pt/~edrdo/supervision/inovo_erc721_pbt.pdf.

[58] Wang Qin, Li Rujia, Wang Qi, and Chen Shiping. Non-fungible token (nft): Overview, evaluation, opportunities and challenges. 2021. URL https://www.researchgate.net/publication/351656444_Non-Fungible_Token_NFT_Overview_Evaluation_Opportunities_and_Challenges.

[59] Delton Rhodes. The double spending problem, explained, 2023. URL https://komodoplatform.com/en/academy/double-spending-problem/.

[60] Rosita Rijtano. Attacco ddos, 2018. URL https://www.cybersecurity360.it/nuove-minacce/ddos-cosa-sono-questi-attacchi-hacker-e-come-stanno-evolvendo/.

[61] River. What is segregated witness (segwit)? URL https://river.com/learn/what-is-segwit/.

[62] Alessio Salvetti. Bitcoin: Guide on the blockchain, 2020. URL https://en.cryptonomist.ch/2020/01/18/bitcoin-guide-on-blockchain/.

[63] Kaushal Shah, Uday Khokhariya, and Saumya Patel. Smart contract-based dynamic non-fungible tokens generation system. 2023. URL https://assets.researchsquare.com/files/rs-2796956/v1_covered.pdf?c=1681361469.

[64] Toshendra Kumar Sharma. Blockchain role of p2p network, 2022. URL https://www.blockchain-council.org/blockchain/blockchain-role-of-p2p-network/.

[65] Nick Szabo. The idea of smart contracts. 1997. URL https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html.

[66] Nick Szabo. Bit gold, 2005. URL https://unenumerated.blogspot.com/2005/12/bit-gold.html.

[67] José Luis Romero Ugarte. Distributed ledger technology (dlt): introduction. 2018. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3269731.

[68] Matteo Zinato. La crittografia a chiave pubblica e rsa, 2006. URL https://www.html.it/pag/16477/la-crittografia-a-chiave-pubblica-e-rsa/.